Universidade Estadual de Feira de Santana

Programa de Pós-Graduação em Ciência da Computação

Ferramentas de visualização e simulação de programas na compreensão de Funções de Ordem-Superior (Higher-Order Functions)

Marcos Rogério Martins

Feira de Santana

2024

Universidade Estadual de Feira de Santana

Programa de Pós-Graduação em Ciência da Computação

Marcos Rogério Martins

Ferramentas de visualização e simulação de programas na compreensão de Funções de Ordem-Superior (Higher-Order Functions)

Dissertação apresentada à Universidade Estadual de Feira de Santana como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Orientador: Rodrigo Silva Duran

Feira de Santana

2024

Ficha Catalográfica - Biblioteca Central Julieta Carteado - UEFS

Martins, Marcos Rogério
M344f Ferramentas de visualização e simulação de programas na compreensão de Funções de Ordem-Superior (Higher-Order Functions) / Marcos Rogério Martins. – 2024.
121 f. : il.

Orientador: Rodrigo Silva Duran.
Dissertação (mestrado) – Universidade Estadual de Feira de Santana, Programa de Pós-Graduação em Ciência da Computação, 2024.

1. Programação (Computadores). 2. Funções de Ordem-Superior. 3. Visualização de programa. 4. Softwares – Utilização. I. Título. II. Duran, Rodrigo Silva, orient. III. Universidade Estadual de Feira de Santana.

CDU 004.4

Renata Aline Souza Silva - Bibliotecária - CRB-5/1702

Marcos Rogério Martins

Ferramentas de visualização e simulação de programas na compreensão de Funções de Ordem-Superior (Higher-Order Functions)

Dissertação apresentada à Universidade Estadual de Feira de Santana como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Feira de Santana, 13 de setembro de 2024

BANCA EXAMINADORA

Documento assinado digitalmente
gov.br
RODRIGO SILVA DURAN
Data: 13/09/2024 16:53:47-0300
Verifique em https://validar.iti.gov.br

Rodrigo Silva Duran (Orientador(a))
IFMS

Documento assinado digitalmente
gov.br
ROBERTO ALMEIDA BITTENCOURT
Data: 14/09/2024 14:51:17-0300
Verifique em https://validar.iti.gov.br

Roberto Almeida Bittencourt
Universidade Estadual de Feira de Santana

Documento assinado digitalmente
gov.br
AYLA DEBORA DANTAS DE SOUZA REBOUCAS
Data: 14/09/2024 08:29:45-0300
Verifique em https://validar.iti.gov.br

Ayla Debora Dantas de Souza Reboucas
Universidade Federal da Paraíba

Abstract

There is a growing demand for programming aimed at a target audience generally referred to as non-developers: individuals who use programming to solve tasks such as product recommendation, fraud detection, disease diagnosis, process optimization, file management, among others, but whose primary goal is not professional software development. The activities carried out by non-developers are often more related to the demands of organizing, transforming, and automating data processing than other programming constructs, giving rise to a new paradigm called Data Centrality, where different skills are the focal point of this audience. Therefore, it is imperative to provide efficient and effective computational tools that simply perform tasks involving complex data. Higher-Order Functions (HOFs) have become popular tools among non-developers to carry out such tasks. Despite their simplicity, research shows that students still struggle to understand and use HOFs. Based on the evidence supporting the use of program visualizers for code comprehension, we believe that simulating the concepts inherent to HOFs may contribute to a better understanding of their semantics. However, no visualization and simulation system currently described in the literature explicitly supports HOFs, and little research has been dedicated to their development and use. This work proposed adapting a visualization and simulation tool to incorporate HOF animations and empirically investigating which types of simulation might offer a better understanding of programs using this concept. In addition to the visualization adaptations, educational material for teaching HOFs was produced, along with an assessment tool that attempts to capture potential misconceptions about HOFs. Twenty-one students from various higher education institutions with experience in Python participated in the investigation. When comparing groups that used different types of simulations (intermediate and pictographic) with a control group, we found no significant differences in program comprehension. Further research is needed to replicate these results and investigate more deeply the users' engagement with the visualization tool and their behavior when using it.

Keywords: Higher-Order Functions, Program visualization, Program simulation.

Resumo

Há uma demanda crescente por programação direcionada a um público-alvo geralmente denominado de não-desenvolvedores: indivíduos que utilizam a programação para resolver tarefas como recomendação de produtos, detecção de fraudes, diagnóstico de doenças, otimização de processos, gerenciamento de arquivos, entre outras, mas que não possuem como principal objetivo o desenvolvimento de software profissional. Frequentemente, as atividades realizadas por não-desenvolvedores estão relacionadas às demandas de organização, transformação e automação do processamento de dados, do que com outros construtos de programação, originando um novo paradigma denominado de Centralidade de Dados, onde diferentes habilidades são o ponto focal deste público. Desta forma, é imperativo prover ferramentas computacionais eficientes e eficazes que realizem tarefas envolvendo dados complexos de forma simples, e as Funções de Ordem-Superior (FOS) tornaram-se ferramentas populares entre não-desenvolvedores para realizar tais tarefas. Apesar de sua simplicidade, pesquisas mostram que estudantes ainda possuem problemas em compreender e utilizar FOS. Partindo das evidências que suportam o uso de visualizadores de programas para compreensão de código, acreditamos que simular os conceitos inerentes às FOS pode contribuir com uma melhor compreensão de sua semântica. No entanto, nenhum sistema de visualização e simulação descrito atualmente na literatura tem suporte explícito às FOS e pouca pesquisa tem sido dedicada ao seu desenvolvimento e uso. Este trabalho propôs a adaptação de uma ferramenta de visualização e simulação para incorporar animações de FOS e a investigação empírica de que tipos de simulação podem oferecer uma melhor compreensão de programas que se utilizam deste conceito. Além das adaptações da visualização, foi produzido material didático para o ensino de FOS e um instrumento avaliativo que tenta capturar potenciais concepções incorretas sobre FOS.Participaram da investigação 21 estudantes de diversas instituições de ensino superior com experiência em Python. Ao comparar grupos que utilizaram diferentes tipos de simulações (intermediária e pictográfica) com um grupo controle, não encontramos diferenças significativas na compreensão de programas. Pesquisas futuras são necessárias para replicar estes resultados e investigar mais profundamente o engajamento dos usuários com a ferramenta de visualização e seu comportamento ao utilizá-la.

Palavras-chave: Higher-Order Functions, Funções de Ordem-Superior, Visualização de programas.

Prefácio

Esta dissertação de mestrado foi submetida à Universidade Estadual de Feira de Santana (UEFS) como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

A dissertação foi desenvolvida no Programa de Pós-Graduação em Ciência da Computação (PGCC), tendo como orientador o Prof. Dr. **Rodrigo Silva Duran**.

Agradecimentos

Primeiro e antes de tudo, quero agradecer a Deus, que me concedeu a vida, a saúde, a força e a sabedoria para realizar este trabalho. Agradeço por Sua infinita bondade e misericórdia, que sempre me acompanharam em todos os momentos, mesmo nos mais difíceis.

Agradeço também à minha filha, Cecília, a qual é a luz da minha vida e a razão do meu sorriso. Ela me motiva a ser uma pessoa melhor, a alcançar meus objetivos e por me ensinar o verdadeiro significado do amor.

A minha esposa, Cláudia Cunha, meu porto seguro. Ela me apoia em tudo o que faço e me ajuda a superar os desafios. Obrigado por me amar, por me apoiar e por me fazer acreditar em mim mesmo.

A minha mãe, Guiomar, que me ensinou o valor da família, do trabalho e do amor, sem seu apoio e amor incondicional nada disso teria sido possível.

O meu pai, Salvador Martins(In memoriam), meu exemplo de vida, que está sempre presente em meus pensamentos. Ele me inspirou a ser um homem honesto, integro e trabalhador. Saudades eternas

Meus irmãos, Marluce, Marcelo e Marlicéia meus companheiros de vida. Sempre ao meu lado, nos bons e nos maus momentos.

Os meus colegas de trabalho, Fabiane, Laís, Danilla, Marcelo, Talita, Maria Clara, vocês são pessoas que admiro e respeito. Me ajudam a crescer profissionalmente e a aprender coisas novas. Vocês são mais do que colegas, são amigos, parceiros e uma fonte de inspiração.

Agradeço imensamente ao meu amigo e patrão, João Souza, por ter me proporcionado a oportunidade única de conciliar o trabalho com os estudos na UEFS. Sua confiança e incentivo foram fundamentais para que eu pudesse alcançar essa conquista. A ele, minha eterna gratidão.

Aos professores do PGCC, meus sinceros agradecimentos. Além de transmitir conhecimentos valiosos, vocês me inspiraram a buscar a excelência em tudo o que faço, mostrando-me o caminho para o crescimento profissional e pessoal. A cada um de vocês, minha admiração e respeito.

E, finalmente, quero agradecer ao meu orientador, Rodrigo Duran, agradeço imensamente por sua orientação, paciência e dedicação. Você foi mais do que um orientador acadêmico. Você foi um mentor, um amigo e um guia. Obrigado por tudo que você fez por mim.

Agradeço a todos que contribuíram para a realização deste sonho. Sem vocês, nada disso seria possível.

"Faça as coisas o mais simples que você puder, porém não se restrinja às mais simples"

– Albert Einstein

Sumário

Alinhamento com a Linha de Pesquisa

Linha de Pesquisa: Software e Sistemas Computacionais

Educação em Computação é um campo que estuda e exerce o ensino e a aprendizagem de conceitos, habilidades e ferramentas relacionados à computação. Funções de Ordem-Superior são funções que recebem ou retornam outras funções como argumentos ou resultados, permitindo definir padrões de computação comuns que podem ser facilmente reutilizados. E as Ferramentas de visualização e simulação de programas são ferramentas que permitem representar graficamente o funcionamento de um programa, facilitando a compreensão do seu comportamento, estrutura e lógica.

A compreensão de Funções de Ordem-Superior é uma habilidade importante para a programação funcional, um paradigma de programação que se baseia no uso de funções matemáticas para processar dados. A programação funcional é um tópico relevante para a Educação em Computação, pois oferece uma forma diferente e elegante de pensar sobre problemas computacionais, além de ter aplicações em diversas áreas, como inteligência artificial, processamento de linguagem natural, análise de dados, etc. As ferramentas de visualização e simulação de programas são úteis no ensino de ciência da computação porque podem facilitar o ensino e a aprendizagem de conceitos abstratos e complexos, como Funções de ordem-superior, por meio de representações visuais de fluxo de dados e controle de programas.

Lista de Tabelas

Lista de Figuras

Lista de Abreviações

Abreviação	**Descrição**
BJC	Beleza e Alegria da Computação(Beauty and Joy of Computy)
CD	Centralidade de Dados
CLT	Teoria da Carga Cognitiva(Cognitive Load Theory)
CS1	Cursos Introdutórios de Ciência da Computação
CSBC	Congresso da Sociedade Brasileira de Computação
CSS	Folhas de Estilo em Cascata (Cascading Style Sheets)
CS+X	Ciência da Computação com treinamento técnico/profissional em outras disciplinas.
DOM	Modelo de Documento por Objetos(Document Object Model)
EC	Educação em Computação
EduComp	Simpósio Brasileiro de Educação em Computação
HoF	Funções de Ordem-Superior (Higher Order-Function)
HTML	Linguagem de Marcação de Hipertexto (Hyper Text Markup Language)
IFMS	Instituo Federal do Mato Grosso do Sul
JSON	Notação de Objeto JavaScript (JavaScript Object Notation)
FOS	Funções de Ordem-Superior (Higher Order-Function)
UEFS	Universidade Estadual de Feira de Santana
SBC	Sociedade Brasileira de Computação
TCLE	Termo de Consentimento Livre e Esclarecido
...	...

Capítulo 1

Introdução

Uma questão central em Educação em Computação (EC) é a investigação de como se dá o aprendizado de programação, uma vez que, historicamente, a maioria dos cursos de graduação em Ciência da Computação foram desenhados para formar desenvolvedores de software (Guzdial, 2019). Embora seja vital preparar os estudantes que almejam o mercado profissional, cuja atividade principal em muitos casos envolve a atividade de desenvolvimento de software, existem outras necessidades e propósitos para a computação, e até mesmo para a programação em si. Além das necessidades do desenvolvimento de software para estudantes dos cursos de Computação, e de tecnologia de forma geral, existe uma crescente demanda da programação em outras áreas do conhecimento.

Alguns cursos são denominados CS+X "Ciência da Computação e X" (Brodley et al., 2022), onde nesses cursos, "X" representa uma diversidade de áreas do conhecimento, como negócios, engenharia, saúde, artes, entre outras. Esses cursos permitem que os alunos busquem um programa flexível de estudo incorporando uma base sólida em Ciência da Computação com treinamento técnico ou profissional nas artes e ciência, integrando transdisciplinarmente as habilidades de programação em contextos como biologia (Berger-Wolf et al., 2018), direito (Sloan et al., 2017), ciências sociais (Guzdial e Shreiner, 2021; Shreiner e Guzdial, 2022), artes, jornalismo (Plaue e Cook, 2015), entre outros. Esta demanda tem um público-alvo distinto dos cursos de Computação que possui, em muitos casos, diferentes objetivos. Guzdial (2019) cita duas razões centrais para ensinar programação para não-desenvolvedores: a programação como ferramenta para pensar, ou seja, uma ferramenta para auxiliar o desenvolvimento do raciocínio lógico e a solução de problemas; e a programação como ferramenta para aprender a fazer, onde a programação é o meio de apoio à aprendizagem ou ao desenvolvimento das tarefas do usuário. Em ambas as motivações, a programação é uma ferramenta para atingir um objetivo profissional ou pessoal que não está necessariamente ligado à atividade de desenvolvimento profissional de software. Um estudo realizado por Scaffidi et al. (2005) identificou que existe grande número de programadores não-desenvolvedores, os quais utilizam a programação para resolver problemas em suas tarefas diárias, tais como recomendação

de produtos, detecção de fraudes, diagnóstico de doenças, otimização de processos, gerenciamento de arquivos, processamento de imagens e processamento de dados.

O que estas atividades realizadas por estes não-desenvolvedores têm em comum é o que se convencionou chamar de Centralidade de Dados (Krishnamurthi e Fisler, 2020). Esse novo paradigma de computação coloca um foco muito maior no tratamento de dados em oposição ao uso de estruturas de controle de fluxo. As atividades realizadas com a (CD) em foco, em geral, necessitam de aplicações que visem atender demandas organizacionais de armazenamento, recuperação, movimentação e processamento de dados. Esse manuseio massivo de dados requer profissionais com conhecimento para implementar algoritmos que automatizem processos de análise de dados, auxiliando na tomada de decisão. Suas tarefas em geral consistem em transformar e limpar dados, geralmente em contextos complexos onde a expertise em outras áreas do conhecimento é necessária para interpretar tais dados e suas análises. Essa demanda dos não-desenvolvedores em conjunto com a Centralidade de Dados originou inclusive um novo ramo dos cursos de Tecnologia que se convencionou chamar de Ciência de Dados ou *Data Science* (Leidig e Cassel, 2020). Tal ramo visa responder perguntas sobre o mundo através da aplicação de métodos computacionais e estatísticos aos dados aplicados em contextos diversos e ricos (Krishnamurthi e Fisler, 2020).

Para trabalhar com dados, principalmente em grande escala e com dados que apresentam relacionamentos complexos, é fundamental prover ferramentas computacionais eficientes e eficazes que possam realizar tarefas de forma simples. Estudantes dos cursos de Ciência da Computação, até mesmo em seus cursos iniciais, realizam tais tarefas. Considere, por exemplo, a necessidade de filtrar uma coleção de dados de acordo com um determinado critério (p.e., apenas números positivos). Um estudante de Computação poderia produzir um código como o apresentado no exemplo 1.1:

Listing 1.1: Um filtro de números positivos em Python.

```
valores = [-1, 2, -3 , 4, -5]
filtrados = []
for n in valores:
    if (n > 0):
        filtrados.append(n)
print(filtrados)
```

Tal código, apesar de simples se avaliado em termos de número de linhas, tem embutido inúmeros constructos de programação, tais como loops e condicionais, que podem tornar o processo de escrita e compreensão do código difícil (Luxton-Reilly, 2016; Luxton-Reilly et al., 2018). Entretanto, alguns modelos teóricos (Duran et al., 2018) preconizam que o uso extensivo de funções prontas podem reduzir a carga cognitiva necessária para a execução de determinadas tarefas. Considere, por exemplo, a solução do mesmo problema anteriormente citado utilizando *Higher-*

Order Functions – HoF, ou Funções de Ordem-Superior-*FOS*, como apresentado no exemplo 1.2:

Listing 1.2: Um simples filtro de números positivos em Python usando FOS.

```
valores = [-1, 2, -3 , 4, -5]
filtrados = list(filter(lambda n: n>0, valores))
print(filtrados)
```

O código contido no exemplo 1.2 é mais conciso tanto em termos de tamanho, com potencial de ser menos complexo pelo seu uso de funções prontas (Duran et al., 2018), o que poderia auxiliar no processo de escrita de programas e solução de problemas sem que o público não-desenvolvedor tenha que aprender uma vasta gama de conceitos ligados à programação. As *FOS* são descritas como funções que operam em outras funções, seja tomando-as como argumentos ou retornando-as como resultado, além de permitirem composição de funções, ou seja, possuir pequenas funções que compõem funcionalidades que realizam tarefas mais extensas e complexas. As *FOS* são componentes que estão cada vez mais presentes nas linguagens de programação modernas tais como Java, Javascript e R e em bibliotecas como JQuery[1] (biblioteca Javascript amplamente utilizada por desenvolvedores Web) e a biblioteca Pandas[2] (biblioteca Python extremamente popular em análise de dados).

Dentre as inúmeras *FOS*, podemos destacar três funções amplamente utilizadas por desenvolvedores e não-desenvolvedores em virtude do seu poder de expressão para solucionar problemas que envolvam dados complexos eficientemente: *Map, Filter e Reduce*, utilizadas para transformar, filtrar e agregar dados, respectivamente. Estes métodos são potencialmente úteis para reduzir a carga cognitiva, trabalhar sem "efeitos colaterais" e, muitas vezes, tornar o código mais legível(Duran et al., 2018; Kather et al., 2021).

Contudo, pouca pesquisa tem sido dedicada ao aprendizado ou a compreensão de estudantes sobre *FOS*. Estudos recentes mostram que estudantes conseguem compreender os objetivos de uma *FOS*, conseguem reconhecê-las individualmente e utilizá-las para planejar soluções corretas para certos problemas de processamentos de dados (Krishnamurthi e Fisler, 2021). Entretanto, os estudantes, ainda têm dificuldades na compreensão de funções como o reduce (Krishnamurthi e Fisler, 2021) e na composição de funções (Rivera et al., 2022).

Dificuldades semelhantes na composição e compreensão de código enfrentadas pelos estudantes já foram observadas no passado em programas utilizando outros tipos de constructos (Sorva et al., 2012). Para mitigar tais dificuldades, estudos anteriores sugerem que a visualização e simulação de programas podem ser eficazes e melhorar o processo de compreensão de programas (Sorva e Sirkiä, 2010; Sorva et al., 2012).

[1] https://jquery.com/
[2] https://pandas.pydata.org/

A visualização e simulação de programas explora abordagens e métodos de representação de código e processamento gráfico estático e dinâmico de elementos deste código, tais como variáveis, chamadas de função e estado da *stack*. A apresentação destes elementos de código está relacionada principalmente à análise e ao desenvolvimento de programas, visando melhorar a compreensão do programa inerentemente invisível e intangível e apresentar concretamente a semântica de uma determinada linguagem de programação de forma simplificada e com objetivos educacionais, ou seja, a sua *Notional Machine* (Boulay, 1986). A *Notional Machine* pode ser considerada uma abstração idealizada do hardware do computador e outros aspectos do ambiente de tempo de execução dos programas utilizada na educação de programação para ajudar os alunos a entender os conceitos de variáveis, operadores, estruturas de controle e outros elementos da programação (Sorva, 2013; Duran et al., 2021). O principal desafio é encontrar traduções eficazes dos vários aspectos do código para representações gráficas usando metáforas visuais (Sorva et al., 2013).

Estudos anteriores exploraram tais representações de construtos como variáveis, loops, condicionais, objetos, entre outros, para dinamicamente simular a execução de um determinado código (Eliot et al., 1997; Lattu et al., 2000a; Levy et al., 2003; Moreno et al., 2004). Estas representações diferem em termos da *Notional Machine* utilizada e dos objetivos da visualização, bem como no nível de detalhe da simulação apresentada e nos elementos gráficos utilizados. Por exemplo, o JSVEE, uma biblioteca JavaScript desenvolvida para criar visualizações de programas educacionais para diversas linguagens de programação, que podem ser incorporadas em páginas web (Sirkiä, 2013), parte da premissa que os estudantes devem ser expostos a uma *Notional Machine* que apresenta todos os detalhes ao nível de avaliação de expressão, o que significa que ela especifica as operações básicas necessárias para avaliar expressões, bem como a ordem em que essas operações devem ser realizadas. JSVEE, inclui funções básicas comuns, como controle de animação passo a passo, mover para frente e para trás, mostrar textos explicativos, assim, como as etapas de visualização mais comuns, como buscar valores, avaliar operadores e atribuir valores a variáveis. Contudo, ela ainda não possui suporte às Funções de Ordem-Superior.

Não encontramos referências na literatura a sistemas de visualização que possuem suporte explicito às *FOS*. No entanto, o Python Tutor, outra ferramenta de visualização e simulação de programas baseada na Web (Guo, 2013), possui suporte limitado às *FOS*, oferecendo visualizações com recursos insuficientes para auxiliar programadores a entender códigos que contenham *FOS*. Por exemplo, não há uma representação da visualização da pilha de chamadas quando *FOS* são invocadas, recurso utilizado para demonstrar o fluxo de execução, auxiliando desenvolvedores a rastrear como as funções são aninhadas e executadas. Outra limitação apresentada pelo Python Tutor em nosso entendimento é a falta de elementos pictóricos, para ilustrar as relações entre funções e suas entradas e saídas, facilitando a compreensão de dependências funcionais complexas.

Assim, este trabalho pretende preencher esta lacuna da literatura e propor potenciais designs para sistemas de visualização e simulação de *FOS* e avaliar como tais designs

impactam a compreensão de programas que utilizam as funções Filter, Map e Reduce escritos em Python.

1.1 Objetivos e Questões de Pesquisa

O objetivo geral desse trabalho é analisar, por meio de um estudo empírico, o impacto das visualizações e simulações de Funções de Ordem-Superior adaptadas pelos autores, utilizando a biblioteca de simulação e visualização JSVEE, quanto à compreensão de programas que utilizam as Funções de Ordem-Superior: Filter, Map e Reduce escritos em Python. Como objetivos específicos pode-se listar:

- Prototipação de designs para sistemas de visualização e simulação de Funções de Ordem-Superior.
- Visando representar visualmente o comportamento de *FOS*, utilizaremos a biblioteca JSVEE para adaptar e personalizar animações a partir dos protótipos de designs.
- A construção de um material didático sobre Funções de Ordem-Superior utilizando a linguagem de programação Python.
- A construção de um questionário de avaliação sobre Funções de Ordem-Superior objetivando analisar a compreensão dos participantes quanto ao tema e identificar possíveis *misconceptions*.

Diante dos objetivos apresentados, as seguintes questões de pesquisa foram propostas:

1. **QP1**. Quais as potenciais características de design que podem ser utilizadas na simulação e nas representações visuais de Funções de Ordem-Superior tais como o Filter, Map e Reduce?
2. **QP2**. A partir destas características de design, quais melhoram a compreensão de programas utilizando Funções de Ordem-Superior escritos em Python?

1.2 Contribuições

Funções de Ordem-Superior se tornaram elementos essenciais em diversas linguagens de programação com ampla utilização em desenvolvimento de aplicações web e Ciência de Dados. Com uso crescente em vários domínios importantes, acompanhado pelo fato de diversos programadores não-desenvolvedores utilizarem habilidades de programação para tarefas específicas, visualizar os conceitos inerentes às Funções de Ordem-Superior pode contribuir com uma melhor compreensão das suas funcionalidades e aplicações.

A visualização de Funções de Ordem-Superior contribui para a abstração de código, permitindo que os programadores compreendam o funcionamento das funções sem a

necessidade de se aprofundar nos detalhes da implementação. Essa abstração facilita a manipulação e o reuso das Funções de Ordem-Superior em diferentes contextos, otimizando o desenvolvimento de software e promovendo a legibilidade do código.

Ao possibilitar a visualização das relações entre as Funções de Ordem-Superior e seus argumentos, a técnica contribui para o aumento da modularidade do código. Essa visualização torna evidente como cada Função de Ordem-Superior opera sobre os dados de entrada, facilitando a identificação de dependências e a modularização do código em unidades menores e reutilizáveis.

A visualização se configura como um recurso didático valioso para o ensino e aprendizado de Funções de Ordem-Superior. Ao fornecer representações gráficas dos conceitos abstratos envolvidos, as visualizações facilitam a compreensão dos alunos, dispensando a necessidade de memorização complexa e permitindo o foco na aplicação prática das Funções de Ordem-Superior.

Do ponto de vista do ensino, a visualização das Funções de Ordem-Superior auxilia os professores na comunicação clara e concisa dos conceitos relacionados às funções. Por meio de recursos visuais, os professores podem tornar as aulas mais dinâmicas e envolventes, capturando a atenção dos alunos e promovendo uma aprendizagem mais profunda e significativa.

O material instrucional desenvolvido para a pesquisa, incluindo as animações interativas e os códigos exemplos, podem auxiliar tanto estudantes quanto professores no aprendizado de Funções de Ordem-Superior:

- Animações interativas: permitem que os usuários explorem e manipulem os conceitos de forma lúdica e interativa, facilitando a compreensão do funcionamento das Funções de Ordem-Superior.
- Códigos exemplos: guiam os usuários através da aplicação real de Funções de Ordem-Superior em diferentes cenários, consolidando o aprendizado e promovendo a autonomia na resolução de problemas.

Visando complementar o material instrucional e avaliar a efetividade da aprendizagem de *FOS*, desenvolvemos uma avaliação final que integra questões de rastreio de código. Essas questões desafiam os alunos a analisar e interpretar o funcionamento de *FOS* em diversos contextos, promovendo o desenvolvimento do raciocínio lógico e da habilidade de depuração, essenciais para a programação. Além disso, a avaliação foi cuidadosamente elaborada para identificar e abordar possíveis *Misconceptions* comuns, contribuindo para uma compreensão mais profunda e precisa do tema.

1.3 Organização do Trabalho

Iniciamos este trabalho com uma revisão da literatura pertinente no Capítulo 2, a fim de situar o tema da pesquisa no contexto atual. Em seguida, no Capítulo 3, descrevemos o método utilizado para conduzir a pesquisa. Os resultados obtidos

foram apresentados e analisados nos Capítulos 4 e 5, respectivamente. Por fim, no Capítulo 6, sintetizamos as principais conclusões e discutimos as implicações dos resultados para futuras pesquisas. Materiais complementares podem ser encontrados no Apêndice A .

Capítulo 2

Revisão Bibliográfica

"You can't teach people everything they need to know. The best you can do is position them where they can find what they need to know when they need to know it."

– Seymour Papert

2.1 Centralidade de Dados

Programar é criar, pensar em coisas que nunca foram feitas antes, planejar novas soluções para problemas ou organizar informações. Programadores realizam tal tarefa escrevendo programas que executam algum comportamento desejado.

Ko et al. (2011) definem programação para não-desenvolvedores como "alcançar o resultado de um programa, principalmente para uso pessoal", o que difere da programação profissional cujo objetivo é produzir código para a utilidade de outros.

Existe uma diversidade de práticas de uso de software exibidas por usuários não-desenvolvedores. Scaffidi et al. (2005) salientam que muitos usuários não-desenvolvedores utilizam a programação para realizar tarefas em campos diversos e diferentes do desenvolvimento de software, mostrando que existem diferentes maneiras de usar habilidades de programação.

Por exemplo, designers gráficos e outros envolvidos na edição de mídia formam um grupo relativamente novo e crescente de programadores não-desenvolvedores (Dorn e Guzdial, 2006). Eles recorrem a pacotes de software profissionais como Adobe Photoshop e GIMP, implementando interfaces de script integradas por meio de linguagens de programação como JavaScript, Scheme e Python no processo de edição de imagens.

Professores do ensino médio e fundamental aprendem a programar, não com o objetivo principal de desenvolver software, mas para ensinar computação (Guzdial, 2019), ou para fornecer aos seus alunos simulações, animações ou outros objetos computacionais para fins especiais em suas salas de aula (Wiedenbeck, 2005).

Além destes profissionais da educação e computação gráfica, existe de forma geral uma crescente demanda para aprender programação por indivíduos que não são desenvolvedores de software, devido ao que se convencionou chamar de Centralidade de Dados (Krishnamurthi e Fisler, 2020). Esse novo paradigma de enxergar a computação, tem foco voltado a atividades que transformam, limpam e filtram dados para ser possível realizar análises estatísticas e visualizações computacionais que permitam uma interpretação mais clara dos dados. Esse paradigma vem sendo extensamente utilizado por profissões que demandam um conhecimento computacional suficiente, além de profundo conhecimento do domínio para analisar e manipular dados (Krishnamurthi e Fisler, 2020), caracterizando essa computação como intrinsecamente interdisciplinar.

Profissionais de áreas como administração, matemática e jornalismo estão em um processo de apropriação para o novo campo da Ciência da Computação que se dedica a investigar e prover métodos para esse novo paradigma de centralidade de dados: a Ciência de Dados (Cao, 2017; Force, 2021). Este campo requer habilidades particulares tais como analisar, manipular e entender os dados com rapidez e eficiência. Estas tarefas utilizam cada vez mais linguagens de programação modernas com semânticas e estruturas funcionais, tais como R e Python, ferramentas fundamentais e poderosas para tarefas que exigem análise e processamento de dados, por serem simples, robustas, eficientes e versáteis.

A centralidade de dados se tornou um paradigma fundamental na área da computação, permeando diversos aspectos da prática e da teoria. No ensino de programação, essa perspectiva também vem ganhando destaque, objetivando preparar os futuros desenvolvedores para lidar com os desafios da era da informação.

Um exemplo de abordagem que incorpora a centralidade dos dados no ensino de programação é o curso BJC *Beauty and Joy of Computy* (Goldenberg et al., 2020). Esse curso introdutório à ciência da computação, direcionado a estudantes do ensino médio e superior, explora a alegria e a profundidade da criação de programas e aplicativos, ao mesmo tempo, em que incentiva a reflexão crítica sobre os impactos das novas tecnologias.

O BJC utiliza a linguagem de programação visual Snap!, de fácil aprendizado, e aborda conceitos poderosos da ciência da computação, como recursão, Funções de Ordem-Superior e computabilidade. Os resultados obtidos com a implementação do BJC têm sido promissores, com aumento da participação de professores e diversidade de alunos nos cursos de ciência da computação (Goldenberg et al., 2020). Além disso, os estudantes do BJC demonstraram ganhos significativos em conhecimento de conteúdo, autoeficácia, habilidades de programação e fluência em conceitos computacionais (Goldenberg et al., 2020).

A abordagem inovadora do BJC, que enfatiza a centralidade dos dados e a programação como uma atividade divertida e acessível, representa um modelo promissor para o ensino introdutório de ciência da computação. Essa centralidade dos dados, aliada ao uso de ferramentas adequadas, pode facilitar a aprendizagem de conceitos complexos

Cursos introdutórios de Ciência da Computação (CS1) frequentemente buscam equilibrar a introdução de diferentes paradigmas de programação com o ensino de conceitos fundamentais. Linguagens como MuLE (Dümmel et al., 2023) , que suportam múltiplos paradigmas, têm se mostrado eficazes nesse sentido. O estudo de (Dümmel et al., 2023) demonstra que o uso de MuLE em cursos CS1 pode melhorar o desempenho dos alunos.

Um conceito fundamental na centralidade de dados e na programação funcional são as Funções de Ordem-Superior (FOS). O estudo de Krishnamurthi e Fisler (2020) evidência a importância das *FOS* em contextos multiparadigma, destacando como elas podem melhorar a escrita e compreensão de código. No entanto, a aprendizagem de *FOS* pode apresentar desafios, como apontado por Austin et al. (2018). Este último estudo sugere a utilização de pequenos problemas práticos para facilitar a compreensão desse conceito.

A conexão entre a centralidade de dados e o uso de *FOS* é evidente. As *FOS*, como ferramentas poderosas para manipular dados, podem ser utilizadas para implementar conceitos essenciais da centralidade de dados de forma eficiente e expressiva. Diante da crescente importância dos dados na sociedade contemporânea e do papel fundamental das *FOS* na programação funcional, é crucial que pesquisas continuem investigando as melhores práticas para o ensino e aprendizado desse conceito. A compreensão profunda das *FOS* não apenas capacita os programadores a escreverem código mais conciso e elegante, mas também contribui para a formação de profissionais mais aptos a lidar com os desafios da computação moderna.

2.2 Funções de Ordem-Superior - *Higher-Order Functions*

Linguagens que fornecem elementos e estruturas que podem prover comportamento funcional, como Python, possuem ferramentas computacionais para trabalhar com dados, principalmente com dados que apresentam relacionamentos complexos ou com grandes volumes, de forma simples e eficiente. A exemplo podemos citar algumas das bibliotecas mais utilizadas em Python como NumPy, Pandas, NLTK e BeautifulSoup. As Higher-Order Functions, ou Funções de Ordem-Superior, são uma dessas ferramentas amplamente utilizadas no tratamento e manipulação de dados, por apresentarem métodos úteis para reduzir a complexidade e aumentar a clareza do código (Duran et al., 2018).

Dentre as Funções de Ordem-Superior, três funções destacam-se em virtude de seu

poder de expressão: *Map, Filter e Reduce*. Tais funções são utilizadas para transformar, filtrar e agregar dados, respectivamente. A utilização dessas poderosas ferramentas em Python pode melhorar significativamente como os programas são construídos, por tornarem o código mais modular, permitem avaliar funcionalidades e simplificam a reutilização de código. Descrevemos a seguir o funcionamento das três funções que serão investigadas nesse estudo:

- *Map* possui dois parâmetros: o primeiro parâmetro corresponde a uma função de transformação que será aplicada a cada elemento de uma lista, que será mapeada (o segundo parâmetro). Retornando uma nova lista contendo os elementos do segundo parâmetro alterados pela função de transformação.
- A função *Filter* testa se cada elemento de uma lista (argumento) satisfaz a condição da função parâmetro. Retornando uma nova lista contendo apenas os elementos que satisfazem a condição da função parâmetro.
- E por fim temos a função *Reduce*, que diferentemente de *Map e Filter*, não é nativa no Python, precisando ser importada de uma biblioteca (functools, por exemplo). Essa função também se difere, pois não retorna (necessariamente) uma lista como resultado: ela pode retornar apenas um valor resultante da aplicação da função de redução a todos os elementos da lista. Para tanto, introduz um novo parâmetro que serve como acumulador para o resultado parcial a cada iteração com os elementos da lista.

Frequentemente utilizadas em campos como Desenvolvimento Web e Ciência de Dados, o uso das Funções de Ordem-Superior em ambientes educacionais é relativamente pouco explorado. Krishnamurthi e Fisler (2021) investigaram se estudantes de uma disciplina introdutória do curso de Ciência da Computação utilizando a linguagem Racket, recém-ingressos em uma instituição de elite nos EUA dos quais 10% não possuíam conhecimento prévio em computação, mas todos sendo considerados novatos em programação funcional, conseguiam identificar comportamentos de entrada e saída utilizando Funções de Ordem-Superior. Os autores realizaram um módulo de cerca de 30 dias, ensinando aos alunos programação funcional e ao final do módulo os alunos foram apresentados ao conteúdo de Funções de Ordem-Superior sendo submetidos logo depois à aplicação de uma série de instrumentos e métodos, com exemplos concretos de entrada/saída de programas.

Para esse estudo foram desenvolvidas duas representações diferentes de comportamentos de Funções de Ordem-Superior: a primeira utilizando trechos de código escritos na sintaxe de Racket, e a segunda visualização utilizando recursos pictográficos envolvendo símbolos e cores, com intuito de diminuir a carga cognitiva (Mayer, 2002). Os alunos realizaram atividades de agrupamento ou classificação de Funções de Ordem-Superior através dos comportamentos de cada conjunto de dados ao serem aplicados a uma função. Como resultado, os autores evidenciaram que um número considerável de alunos mostrou-se capaz de correlacionar exemplos de entrada/saída de programas às Funções de Ordem-Superior, porém manifestaram

problemas relacionados à classificação e compreensão de alguns exemplos, em particular da aplicabilidade da função *reduce*.

A partir do estudo de Krishnamurthi e Fisler (2021), Rivera et al. (2022) investigaram a aplicação de Funções de Ordem-Superior como ferramentas de planejamento na perspectiva da composição de planos, que segundo Soloway (1986), são representações estereotípicas de soluções algorítmicas para problemas ou sub-problemas recorrentes. Assim, buscando apresentar mais evidências acerca da capacidade dos alunos de compreender o comportamento de Funções de Ordem-Superior individualmente, entender quais características comportamentais das Funções de Ordem-Superior refletem sua compreensão, assim como investigar a capacidade de planejamento de soluções como a composição de Funções de Ordem-Superior.

Rivera et al. (2022) realizaram um estudo em um curso introdutório acelerado de computação (em um contexto de pesquisa similar ao utilizado por (Krishnamurthi e Fisler, 2021)) onde os estudantes foram instruídos a planejar soluções envolvendo Funções de Ordem-Superior e investigou-se posteriormente o entendimento dos alunos quanto às Funções de Ordem-Superior individuais, com foco principal em Map, Filter e Reduce através de análises quantitativas e qualitativas. Rivera et al. (2022) concluíram que os alunos são capazes de reconhecer Funções de Ordem-Superior individuais por meio de exemplos de entrada/saída, conseguem utilizar uma variedade de recursos para identificação das Funções de Ordem-Superior, e construir planos corretos utilizando composição de Funções de Ordem-Superior além de implementar as composições em código. Entretanto, os alunos apresentaram problemas quanto ao reconhecimento da composição de Funções de Ordem-Superior quando expostos a uma pequena quantidade de exemplos.

Os estudos realizados por Krishnamurthi e Fisler (2021) e Rivera et al. (2022), serviram como base para o estudo realizado por Rivera e Krishnamurthi (2022), no qual os autores investigaram dois padrões de composições de funções para resolver problemas de processamento de dados: estrutural e pipeline. Na composição estrutural, cada função é aplicada sequencialmente, a saída da função anterior, são geralmente utilizadas em conjunto com funções lambda (funções anônimas, sem necessidade de serem definidas antes de seu uso) enquanto a composição pipeline utiliza uma sintaxe mais fluida, encadeando as chamadas de função o que equivale a uma sequência de loops.

2.1:

Listing 2.1: Exemplo de Composição Estrutural.

```
def square(x):
    return x * x

def is_even(x):
    return x % 2 == 0

numbers = [1, 2, 3, 4, 5]
```

```
                                                                    8
even_numbers = filter(is_even, numbers)                             9
squared_even_numbers = map(square, even_numbers)                    10
result = sum(squared_even_numbers)                                  11
print(result)                                                       12
# Output: 20                                                        13
```

2.2:

Listing 2.2: Exemplo de Composição Pipeline.

```
def square(x):                                                      1
    return x * x                                                    2
                                                                    3
def is_even(x):                                                     4
    return x % 2 == 0                                               5
                                                                    6
numbers = [1, 2, 3, 4, 5]                                           7
                                                                    8
result = sum(map(square, filter(is_even, numbers)))                 9
print(result)                                                       10
# Output: 20                                                        11
```

Os autores, apresentaram as estruturas de composição de Funções de Ordem-Superior estrutural e pipeline para alguns professores de programação funcional da Universidade de Brown e para três educadores de outras instituições diferentes com o intuito de responder perguntas acerca da clareza das estruturas e de classificação quanto à usabilidade para resolução de problemas.

Posteriormente, os autores realizaram um experimento envolvendo alunos de um curso introdutório de computação onde os participantes precisavam, a partir de uma entrada concreta, adquirir uma noção do tipo de saída desejada, ficando a cargo da tarefa generalizar a solução para ser aplicada a qualquer conjunto de dados. Como resultado, os autores concluíram que os alunos se saíram melhor com os problemas que utilizavam a composição estrutural em relação aos problemas utilizando composição pipeline, corroborando com as respostas dos especialistas que sinalizaram a composição estrutural mais fácil de utilizar do que a composição pipeline.

(Krishnamurthi e Fisler, 2021; Rivera et al., 2022; Rivera e Krishnamurthi, 2022) demonstram o potencial das Funções de Ordem-Superior como ferramentas de ensino para auxiliar na compreensão de conceitos de programação funcional e na resolução de problemas complexos. No entanto, indicam que os alunos podem apresentar dificuldades em compreender plenamente o funcionamento e a aplicação das Funções de Ordem-Superior especialmente no que se refere à composição de funções.

É comum que alunos iniciantes desenvolvam *misconceptions* (ideias ou concepções errôneas) sobre a natureza das *FOS*, como a ideia de que elas são entidades comple-

xas e abstratas, dificultando a sua aplicação em problemas concretos. Além disso, a falta de experiência com programação funcional pode levar os alunos a utilizarem estratégias procedurais para resolver problemas que poderiam ser solucionados de forma mais elegante e concisa com *FOS*. A identificação e o tratamento dessas *misconceptions* podem ser cruciais para o sucesso do ensino de Funções de Ordem-Superior.

2.3 Misconceptions

A evolução constante das linguagens e paradigmas de programação ao longo da história da computação trouxe consigo novos desafios para o ensino e a aprendizagem da programação. Embora fundamental em diversas áreas, a programação apresenta obstáculos significativos, especialmente para iniciantes. A compreensão de conceitos abstratos, muitas vezes, é obscurecida por *misconceptions*, ou seja, modelos mentais incorretos, persistentes (e frequentemente consistentes) acerca do funcionamento de um determinado aspecto do funcionamento de programas (Sorva et al. (2012); Duran et al. (2021)). A literatura sobre o tema tem se dedicado a identificar e categorizar esses obstáculos, buscando oferecer subsídios para a criação de estratégias de ensino mais eficazes (para uma extensa lista, veja Sorva et al. (2012)).

A literatura sobre *misconceptions* na programação tem se dedicado a identificar e categorizar os principais obstáculos enfrentados por estudantes. Estudos como os de (Kaczmarczyk et al., 2010) e (Lewis et al., 2019) evidenciam a persistência desses equívocos, mesmo após a instrução formal, e a necessidade de abordagens pedagógicas mais eficazes para superá-los.

Lewis et al. (2019) destacam a influência do conhecimento prévio dos alunos e do "ponto cego do especialista" na aprendizagem de computação. Ao analisar as dificuldades dos estudantes, os autores enfatizam a importância de diagnosticar e corrigir os *misconceptions* para otimizar o processo de ensino-aprendizagem.

Swidan et al. (2018) ampliaram a investigação sobre *misconceptions*, utilizando o ambiente visual Scratch para analisar as dificuldades de crianças e adolescentes. Os resultados indicam que esses equívocos não se restringem aos iniciantes e podem estar relacionados a diversos fatores, como o conhecimento prévio de matemática e a interpretação de comandos.

Krishnamurthi e Fisler (2021) investigaram a compreensão de Funções de Ordem-Superior por estudantes de computação, identificando dificuldades em correlacionar o comportamento de entrada e saída dessas funções e em generalizar seus comportamentos. Esses resultados evidenciam a necessidade de abordagens pedagógicas mais eficazes para o ensino de *FOS*.

Identificar *misconceptions* na aprendizagem de programação é fundamental para o desenvolvimento de ferramentas e recursos pedagógicos mais eficientes. Sistemas de visualização e simulação de programas emergem como uma promissora alternativa

para auxiliar na identificação e correção desses equívocos. Ao proporcionar uma representação visual do código, esses sistemas podem facilitar a compreensão de conceitos abstratos e a detecção de erros lógicos.

2.4 Sistemas de Visualização de Programas

Um grande desafio para iniciantes em programação é se familiarizar com o conceito de máquina fictícia ou máquina nocional (*Notional Machine*) (Boulay, 1986), ou seja, a abstração do computador no papel de executor de programas, que pode ter diferentes linguagens de programação e paradigmas associados a diferentes máquinas nocionais (Sorva et al., 2013).

O uso de imagens como recurso didático é muito comum em livros e em salas de aula em todo o mundo, não se limitando ao ensino de computação (Sorva et al., 2013) e instrutores que atuam na área de educação de computação frequentemente utilizam algum tipo de visualização em seus cursos (Naps et al., 2002). A visualização de uma *Notional Machine* pode tornar processos e tempo de execução ocultos em focais, explícitos e controláveis, servindo como modelos conceituais que ajudam os alunos a construir um conhecimento de programação possível (Naps et al., 2002).

Nas últimas décadas, muitas ferramentas de visualização e simulação de programas foram desenvolvidas, tais como o Jeliot, uma das ferramentas com maior durabilidade e amplamente estudada na Educação em Computação. Desenvolvida inicialmente como Eliot (Eliot et al., 1997) tendo como finalidade animar graficamente variáveis em programas utilizando a linguagem de programação C definidas pelo usuário, o Eliot (Figura 2.1) permitia que o usuário selecionasse quais variáveis eram animadas e também definir parcialmente sobre como era a visualização de um programa executado por meio da escolha de cores e locais para as caixas que representavam tais variáveis. Os objetivos de Eliot estavam principalmente no lado da visualização de algoritmos, e era usado de forma mais eficiente por programadores que possuíam alguma experiência.

Em sua primeira atualização denominada Jeliot I, implementação para a linguagem Java de Eliot (Haajanen et al., 1997), desenvolvido com intuito de levar animações de programas para a Web, essa ferramenta foi avaliada e definida em dois cursos introdutórios de programação como útil e adequada para iniciantes (Lattu et al., 2000b). Apesar das experiências positivas (Lattu et al., 2000b, 2003), foram observados problemas ao utilizar Jeliot I para o ensino de programação introdutória: uma interface muito complexa para alguns iniciantes usarem e muitos aspectos da execução do programa, considerados úteis por iniciantes, não tiveram visualização implementadas (por exemplo, objetos e classes).

Jeliot 2000 (Levy et al., 2003) uma evolução do Jeliot apresentado na Figura 2.2, foi projetado para oferecer uma experiência mais intuitiva para iniciantes na programação. Com automação completa da visualização, a ferramenta dispensava a

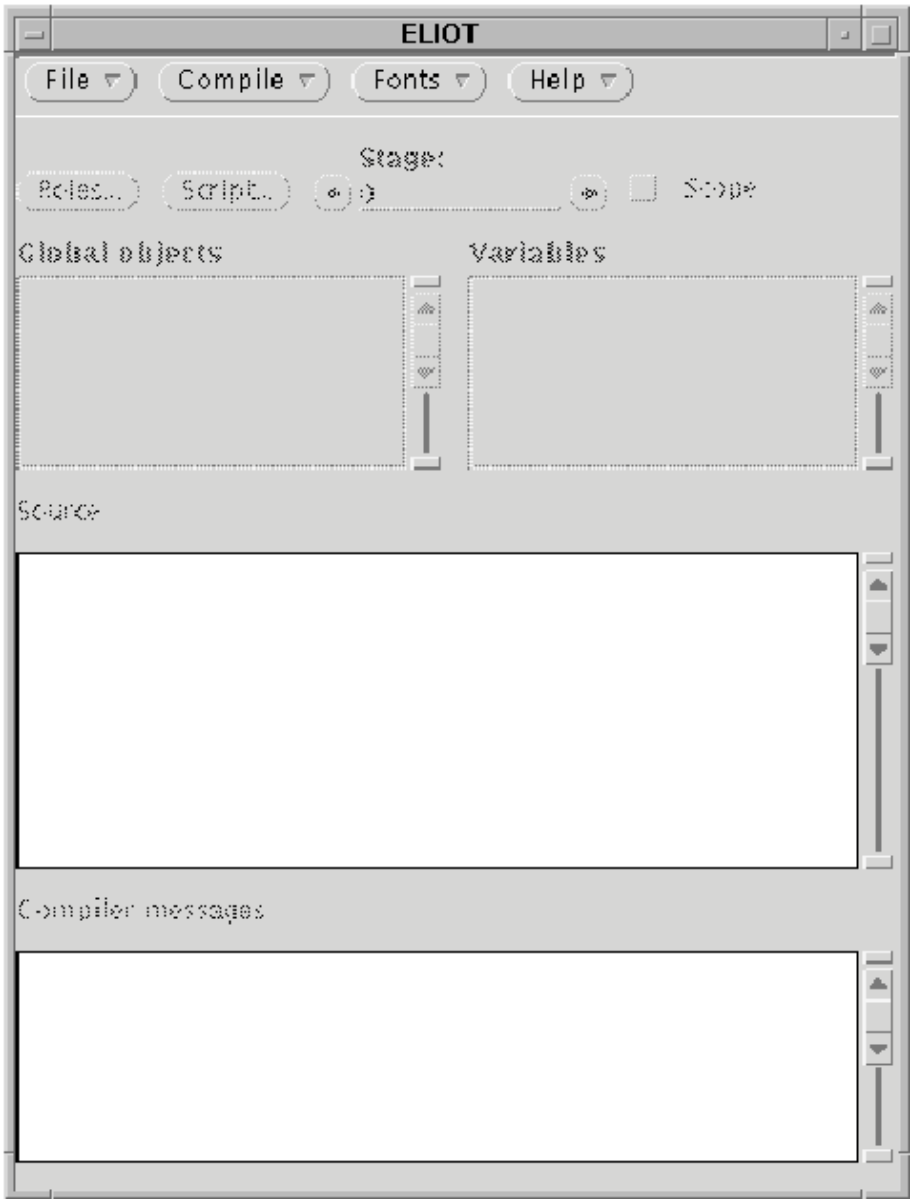


Figura 2.1: Eliot.

intervenção do usuário na definição da aparência ou dos aspectos a serem animados durante a execução do programa. Essa abordagem proporcionava uma visualização detalhada e consistente do código Java, incluindo a avaliação de expressões. No entanto, o Jeliot 2000 apresentava uma limitação ao não suportar conceitos de programação orientada a objetos, embora oferecesse uma representação gráfica das referências a objetos de matriz por meio de setas.

Com o crescente uso da linguagem Python, em especial em cursos introdutórios de programação, em diversas universidades e livros digitais, o Python Tutor (Guo, 2013) foi desenvolvido como uma ferramenta de visualização e simulação de pro-

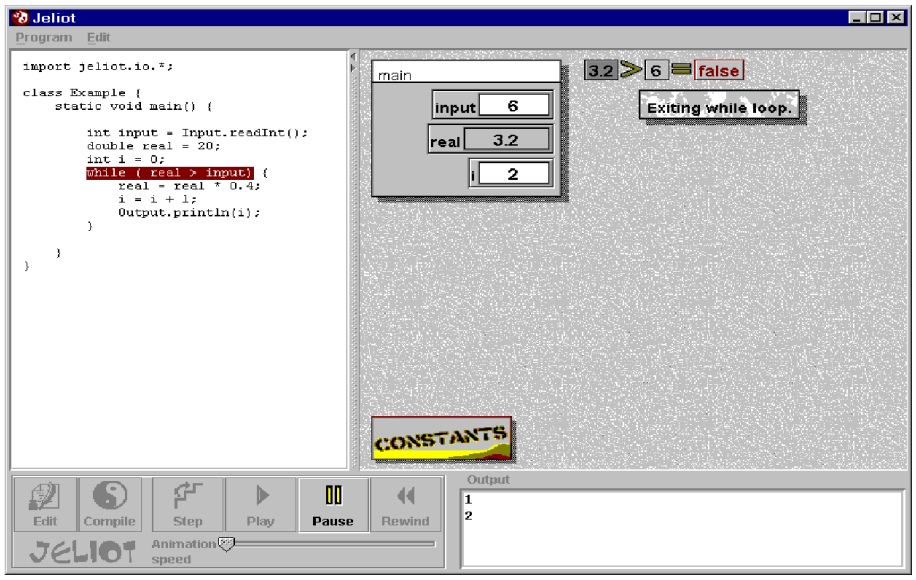


Figura 2.2: Jeliot 2000.

gramas baseado na Web. O design do Python Tutor Figura 2.3 objetivava gerar visualizações de códigos escritos em Python sem necessidade de instalação de programas ou plugins, executado direto do navegador.

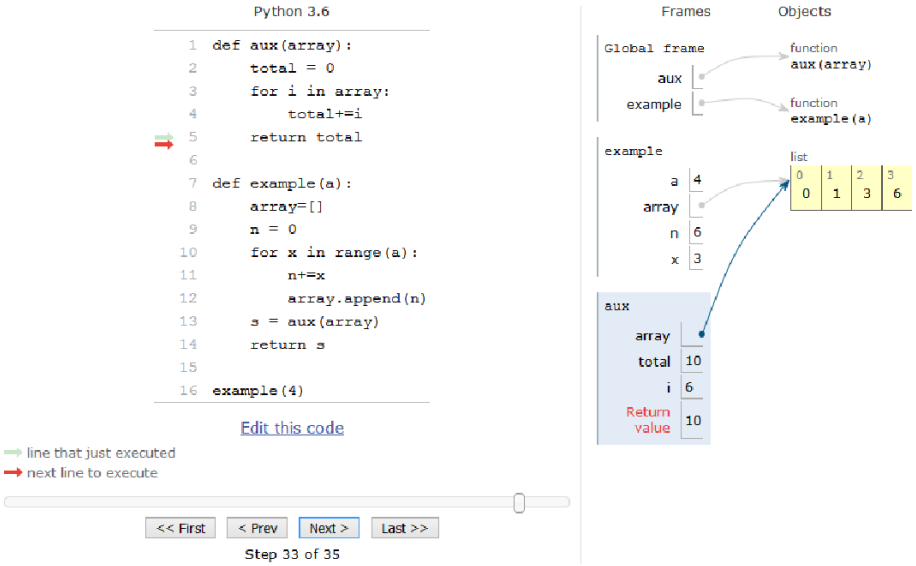


Figura 2.3: Python Tutor.

Os objetivos e design do Python Tutor foram motivados pelas experiências de instrutores com o ensino de Python, que frequentemente desenhavam manualmente diagramas confusos no quadro (Guo, 2013). Desenvolvida em Python, HTML, CSS e JavaScript, essa ferramenta transforma o código-fonte de um programa Python em uma visualização detalhada de sua execução. A partir do código como entrada,

ela gera um rastreamento passo a passo, permitindo a análise do comportamento do programa. Seu backend executa o programa de entrada sob a supervisão do módulo depurador padrão do Python, que interrompe a execução após cada linha executada e registra o estado de tempo de execução do programa. O rastreamento é uma lista ordenada de pontos de execução, onde cada ponto contém o estado logo antes de uma linha de código estar prestes a ser executada. Após o término da execução, o backend codifica o rastreamento no formato JSON, serializando os tipos de dados (Python) em tipos JSON nativos com tags de metadados extras. O backend interrompe a execução após 300 linhas executadas como forma de evitar loops infinitos e rastreamentos excessivamente longos.

A visualização da execução do programa é realizada através do site do Python Tutor[1] onde o usuário por meio de uma caixa de texto pode digitar (ou colar) um código Python, e ao clicar no botão "Visualizar execução", o frontend envia o código python digitado para o backend efetuar todo o processo de codificação para visualização e então devolver um rastreamento de execução JSON, que o frontend renderiza em uma interface gráfica.

Guo (2013) sinalizou necessidades de outros estudos para determinar possíveis adições de novos recursos e funcionalidades ao Python Tutor, como, por exemplo, a incorporação de prompts de perguntas, mini-testes, anotações textuais e tópicos de discussão diretamente nos elementos de visualização, e uma visualização da depuração mais refinada, pois o Python Tutor utiliza-se de uma *Notional Machine* que mostra a execução uma linha por vez como um depurador e não mostra os detalhes da avaliação da expressão e da sub expressão.

Outra ferramenta de visualização e simulação de programas escritos em Python foi desenvolvida por (Sirkiä, 2016) denominado JSVEE, uma biblioteca JavaScript cujo objetivo é fornecer recursos básicos comuns necessários nas animações do programa, tais como controlar e criar um layout para a animação. A biblioteca fornece as partes técnicas e o usuário da biblioteca dedica-se mais ao desenvolvimento do conteúdo. A biblioteca consiste em cerca de 50 operações prontas que controlam a *Notional Machine* implementada na biblioteca. Combinando essas operações, chamadas de sequência de operação, o resultado é uma animação passo a passo na qual as operações são executadas visualmente.

Essas operações contêm, por exemplo, acesso a variáveis, avaliação de operações aritméticas e lógicas, saltos condicionais, chamadas de funções com parâmetros e valores de retorno, bem como definição de classes com métodos e criação de instâncias e acesso de suas variáveis de instância. O JSVEE (veja Figura 2.4) produz animações consistindo em elementos HTML manipulados com JavaScript e estilizados com CSS, que podem ser embutidas em qualquer página web.

A biblioteca JSVEE fornece uma *Notional Machine* que executa visualmente as operações fornecidas. Ao usar a visualização ao nível de expressão, os alunos podem

[1] https://pythontutor.com/

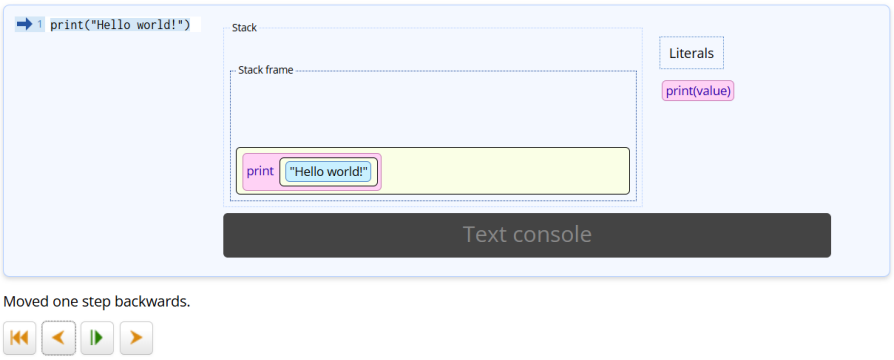


Figura 2.4: JSVEE.

entender melhor a *Notional Machine* e aprender uma maneira precisa de pensar como as expressões são avaliadas e como a *Notional Machine* funciona (Sorva et al., 2012). Isso beneficia o aprendizado dos novatos e permite uma melhor compreensão da execução dinâmica dos programas.

Sirkiä (2016) relatou problemas de usabilidade no JSVEE, tais como a posição inadequada de animações em telas pequenas, e a falta de um tradutor para tornar a produção de animações mais conveniente. Seu estudo posterior (Sirkiä et al., 2017) demonstrou que o uso do JSVEE pode auxiliar estudantes na compreensão de programas escritos em Python. Ferramentas de anotação e quizzes integrados às visualizações de código foram posteriormente adicionadas ao ecossistema do JSVEE, aprimorando ainda mais sua utilidade (Sirkiä et al., 2017).

Ferramentas como o JSVEE evidenciam o poder da visualização e simulação no aprendizado de programação. Para explorar esse potencial no ensino de Funções de Ordem-Superior, propomos a integração dos princípios da Aprendizagem Multimídia e Pistas Visuais no desenvolvimento de sistemas de visualização e simulação de programas com suporte a Funções de Ordem-Superior.

2.5 Aprendizagem Multimídia e Pistas Visuais

As animações servem a muitos propósitos na educação, como chamar a atenção dos alunos, demonstrar técnicas concretas ou abstratas e ajudar os alunos a entender os sistemas dinâmicos (Surjono e Muhtadi, 2017). Os ambientes de aprendizagem multimídia ainda têm grande potencial para melhorar o modo como as pessoas aprendem (Mayer, 1999; van Merrienboer, 1997; Sweller, 1999). Mutlu-Bayraktar et al. (2019) afirmam que a estrutura de trabalho da cognição humana deve ser considerada no design de aprendizagens multimídia para permitir que o aprendizado progrida conforme os processos cognitivos de cada aluno, pois a aprendizagem pode ser mais eficiente quando representações linguísticas e pictóricas são integradas de forma complementar (Mayer, 2002).

Portanto, ao projetar animações ou visualizações, os designers instrucionais devem

considerar como os alunos processam novas informações e como técnicas de design específicas facilitam o aprendizado (Renkl e Scheiter, 2017; Schroeder e Cenkci, 2018). Sorva et al. (2012) e (Sweller, 1988; Duran et al., 2022) enfatizam como a Teoria da Carga Cognitiva (*Cognitive Load Theory* - CLT) , que destaca a necessidade de controlar a carga cognitiva de materiais instrucionais de forma que não sobrecarreguem a capacidade de processamento cognitivo do estudante, pode ser utilizada como guia para design de sistemas de aprendizado de programação, tais como visualizadores e simuladores de código. De acordo com a CLT, a memória de trabalho é pequena e possui curta duração e qualquer informação processada deve estar dentro de seus limites (aproximadamente 4 elementos distintos armazenados por aproximadamente 10 segundos – Cowan (2001)). Portanto, pistas visuais ou artefatos de cognição externa podem atuar como atenuadores da carga cognitiva ao liberar "espaço" na memória de trabalho.

Como exemplo deste argumento, a CLT aponta que, resolver tarefas como cálculos, exige menos esforço mental quando utilizamos de artefatos auxiliares para armazenar resultados parciais, como uma folha de papel, principalmente quando a expressão aritmética envolve vários elementos e lembrar os resultados parciais da expressão torna-se parte da tarefa cognitiva (Beckmann, 2010). Algumas estratégias de controle de carga cognitiva consistem em apresentar explicitamente os passos envolvidos na tarefa como soluções prontas, tais como os chamados Worked Examples (Muldner et al., 2022), a exemplo da *Notional Machine* utilizada pelo JSVEE. Designers instrucionais também devem observar aspectos relacionados com estas Pistas Visuais. Wang et al. (2020) descrevem Pistas Visuais como "dicas, cores, palavras e setas, ou informações adicionadas que podem fornecer andaimes visuais" e ajudar a direcionar a atenção dos usuários para os aspectos que são importantes nos materiais de aprendizagem e ajudar a orientar os processos cognitivos (Mayer, 1999).

de Koning et al. (2010) atribuem os efeitos das pistas visuais aos seus efeitos nos processos perceptivos e cognitivos. As limitações perceptivas permitem que os humanos se concentrem em apenas uma pequena parte da exibição visual de cada vez com base nas propriedades do elemento, como contraste visuoespacial (por exemplo, cor) e contraste dinâmico (por exemplo, movimento repentino). Kong et al. (2017) argumenta que estas técnicas reduzem as buscas visuais e, portanto, "menos recursos visuoespaciais são necessários para direcionar a execução dos movimentos oculares", definindo Pistas Visuais como "todo elemento visual adicionado ou modificado usado para atrair a atenção de um público para uma área visual específica, estendendo o conceito de adição de não-conteúdo à manipulação de componentes visuais existentes".

Capítulo 3

Metodologia

O objetivo deste capítulo é descrever o percurso metodológico deste estudo experimental, desde a elaboração dos protótipos, a abordagem metodológica utilizada, a coleta de dados; além dos sujeitos participantes e os recursos utilizados para a sua realização.

3.1 Desenho do Estudo

Esta pesquisa pretende investigar a efetividade de ferramentas de visualização e simulação de programas na compreensão de Funções de Ordem-Superior por parte de estudantes de programação. Para tanto, adotamos uma abordagem exploratória - quantitativa, combinando o desenvolvimento baseado em experts com um experimento empírico. Para facilitar a leitura desta seção, a seguir apresentamos um breve resumo do percurso metodológico da pesquisa, que será entendida em detalhes nas seções posteriores.

Natureza da Pesquisa. Com caráter exploratório e comparativo, este estudo objetiva investigar o impacto de visualizações e simulações de programas na aprendizagem de Funções de Ordem-Superior (FOS). A pesquisa busca aprofundar o conhecimento sobre como essas representações visuais podem facilitar a compreensão de um conceito complexo e frequentemente desafiador para iniciantes em programação.

Abordagem. A abordagem metodológica é mista, combinando elementos quantitativos de análise e exploratórios no desenvolvimento. A revisão bibliográfica permitiu identificar as principais ferramentas e metodologias utilizadas em pesquisas anteriores no desenvolvimento de aspectos da simulação e visualização de programas. Os protótipos foram avaliados iterativamente por experts na área e refinados até o seu design final (Nielsen e Molich, 1990). O experimento empírico, de caráter quantitativo, coletou dados sobre o desempenho dos participantes ao utilizar diferentes tipos de visualizações. Através da aplicação de um pós-teste consistido em 8 questões de rastreio de código (Izu et al., 2019).

Revisão da literatura. A revisão da literatura foi estruturada em cinco eixos principais. Inicialmente, investigamos estudos sobre Centralidade de Dados e a importância do aprendizado de programação por uma extensa comunidade de profissionais não-desenvolvedores. Em seguida, aprofundamos nossa compreensão sobre Funções de Ordem-Superior, buscando identificar possíveis melhores práticas para a visualização de conceitos abstratos. Paralelamente, realizamos uma revisão da literatura sobre *misconceptions* em programação, a fim de identificar os principais desafios enfrentados pelos aprendizes e como as visualizações podem auxiliar na sua superação. Também foram revisados os artigos relacionados às dificuldades relacionadas ao uso e aprendizagem de *FOS*. Na quarta etapa, analisamos diversos sistemas de visualização e simulação de programas, comparando suas características e funcionalidades. Por fim, exploramos os princípios da aprendizagem multimídia e o papel das pistas visuais na aprendizagem de programação, buscando integrar esses conhecimentos ao design das nossas visualizações. O objetivo final dessa revisão foi adquirir uma base sólida para o desenvolvimento de visualizações e simulações que explorem o potencial das Funções de Ordem-Superior, visando otimizar a aprendizagem de programação.

Desenvolvimento de Protótipos. Com base nos achados da revisão bibliográfica, foram desenvolvidos dois protótipos de visualização e simulação de programas com *FOS*, utilizando a ferramenta JSVEE como base. Intitulados "Array Intermediário" e "Cores", esses protótipos foram desenvolvidos com o intuito de atender especificamente às necessidades da pesquisa. O desenvolvimento se deu por meio de refinamentos sucessivos do protótipo com avaliação de experts em simulação e visualização de programas. Atuaram como experts nesta avaliação os autores do JSVEE (Sirkiä e Sorva, 2015).

Adaptação das Animações. Adaptação das animações desenvolvidas na etapa de prototipação utilizando a ferramenta *JSVEE* para representar concisamente o comportamento de Funções de Ordem-Superior.

Desenvolvimento do Material Instrucional e do Instrumento de Avaliação. Elaboramos um material instrucional com o propósito de facilitar a compreensão de Funções de Ordem-Superior, na linguagem de programação Python contemplando as animações desenvolvidas utilizando a ferramenta *JSVEE*, além da criação de um instrumento de avaliação composto por 8 questões de rastreio de código, visando avaliar a compreensão dos participantes sobre os conceitos abordados. O material é uma adaptação feita pelo autor deste trabalho de materiais do orientador da tese.

Experimento. Para avaliar o impacto de diferentes abordagens visuais no aprendizado de *FOS* em Python, realizamos um experimento com estudantes de computação. Os materiais instrucionais variavam na presença e tipo de animação (sem animação, array intermediário, pistas visuais com cores). A análise quantitativa dos dados da avaliação buscou identificar diferenças no desempenho entre os grupos.

Avaliação de compreensão de programas utilizando FOS. Para avaliar o desempenho dos estudantes, foi criado um instrumento para avaliar a capacidade dos estudantes em compreender programas que utilizam *FOS*. Como metodologia para

desenvolvimento deste instrumento, foi utilizada uma abordagem mista: foram apresentadas *misconceptions* relatadas anteriormente nos trabalhos de Krishnamurthi e Fisler (2021) e Rivera et al. (2022); posteriormente, os experts em simulação e visualização de programas (Sirkiä e Sorva, 2015) participaram de entrevistas para refinar o instrumento, inclusive sugerindo novos tipos de questões e potenciais *misconceptions* não apresentadas anteriormente na literatura.

Análise dos Dados. Os dados coletados foram analisados de forma quantitativa, utilizando estatística descritiva e inferencial para comparar o desempenho dos participantes em cada grupo. Um teste de ANOVA comparou a diferença de desempenho entre o grupo controle, array intermediário e pistas visuais com cores.

3.2 Participantes

Amostragem. A pesquisa contou com a participação voluntária de estudantes de graduação em diversas áreas da computação, como Ciência da Computação, Engenharia da Computação, Sistemas da Informação e Análise de Sistemas. Inicialmente, a pesquisa se concentraria em alunos da UEFS e IFMS, recrutados nos cursos de programação da instituição. A fim de aumentar a generalizabilidade dos resultados e considerar a diversidade de perfis de estudantes, a pesquisa foi expandida para incluir participantes de outras instituições. Essa decisão também foi motivada pela necessidade de compensar as dificuldades encontradas na coleta de dados devido a interrupções nas atividades acadêmicas das instituições UEFS e IFMS durante períodos distintos da pesquisa. Estes participantes foram recrutados por meio de listas de transmissão de grupos de pesquisa (tais como a lista da comunidade de Educação em computação), divulgação em eventos como o Congresso da Sociedade Brasileira de Computação (CSBC) e convite durante aulas da disciplina de Jogos Digitais na UEFS. Para incentivar a participação, divulgamos um sorteio de livros sobre a história da computação [1]. A participação era anônima e volutária e os estudantes deveriam enviar o seu e-mail apenas se desejassem participar do sorteio.

Critério de Inclusão e exclusão. Como critério de participação na pesquisa pontuamos a necessidade dos estudantes terem conhecimentos compatíveis com a conclusão de disciplinas de programação introdutória ou equivalente em Python, possuindo assim conhecimentos básicos de programação tais como: variáveis, condicionais, loops, funções e listas. Para avaliar tais conhecimentos e auxiliar na alocação dos estudantes em faixas de grupos, eles foram submetidos a um pré-teste de conhecimentos de programação em Python de Duran et al. (2019), o qual consiste em um pequeno questionário de autoavaliação, onde o participante indica o seu nível de proficiência nos constructos presentes em cursos introdutórios. O instrumento pré-teste foi validado em outros contextos, mostrando uma boa capacidade em mensurar o co-

[1] "Ada Lovelace: A condessa curiosa" (`https://www.editorainverso.com.br/pagina-de-produto/ada-lovelace-a-condessa-curiosa`) e "Alan Turing: Suas máquinas e seus segredos" (`https://edufscar.com.br/alan-turing-506903106`)

nhecimento prévio dos estudantes sem a necessidade de realização de testes. Em geral, o instrumento é rápido e pode ser respondido de forma completamente online em menos de 15 minutos. Para assegurar a validade dos resultados, a pesquisa considerou critérios de exclusão participantes que indicaram um nível de proficiência nos constructos de Python muito baixo, participantes que não responderam todas as questões contidas no instrumento de pós-teste e participantes que indicaram no pré-teste possuir um conhecimento prévio em *FOS* acima do nível intermediário.

Amostra. Composta por 25 participantes, a amostra foi coletada de forma assíncrona em um estudo online realizado entre junho e agosto de 2024. Destes, 12 se identificaram como homens (48%), 5 se identificaram como mulheres (20%) e 1 estudante se identificou como não-binário. A idade média dos respondentes é de 23 anos (mínimo 18, máximo 42), e 40% destes se identificaram como brancos, 16% como pretos e 36% como pardos. Cerca de 60% dos estudantes cursam Engenharia da Computação, 28% dos estudantes cursam Ciência da Computação e 12% cursam Engenharia de Software. Estudantes da UEFS compõem 76% da amostra, seguido por estudantes da Universidade Federal do Amazonas (UFAM) com 28%. Devido à greve ocorrida no IFMS no período da pesquisa, nenhum estudante participou da pesquisa.

Aplicando o critério de exclusão, 4 participantes foram excluídos: 3 responderam apenas uma questão do pós-teste e 1 apontou não ter os conhecimentos mínimos sobre Python. **A amostra final consistiu em 21 respondentes.**

3.2.1 Versões dos Materiais de Apoio

A fim de investigar o efeito de diferentes representações visuais na aprendizagem de *FOS*, os participantes foram expostos a materiais de apoio digitais personalizados, elaborados pelo orientador em conjunto com o autor (veja Seção 3.4). Os materiais, com duração estimada de 40 minutos cada, foram disponibilizados online por meio de um link seguro, juntamente com instruções detalhadas. O grupo controle recebeu uma versão sem animações, enquanto os grupos experimentais foram expostos a materiais com animações baseadas nos protótipos "array intermediário" e "pistas visuais", respectivamente. Todas as versões, incluindo o material de pré-teste e pós-teste, foram disponibilizadas em um único formulário, eliminando a necessidade de encaminhamento para outro local.

Neste estudo, os participantes foram expostos a diferentes versões de um material de apoio, visando avaliar o impacto de diferentes abordagens visuais no aprendizado de conceitos de *FOS* em Python. As versões apresentavam variações em relação à utilização de animações e elementos visuais.

- **Versão sem Animações:** nesta versão, o conteúdo era apresentado de forma tradicional, com textos, códigos estáticos e alguns vídeos narrativos. O foco estava na explicação teórica dos conceitos de *FOS* (map, filter, reduce) e composição de funções.

- **Versão "Array Intermediário":** nesta versão, foram introduzidas animações para visualizar a execução dos códigos e o conceito de "array intermediário". Esse elemento visual buscava auxiliar os participantes a acompanhar o fluxo de dados nas operações e a entender o papel de cada função.
- **Versão "Pistas Visuais":** similar à versão anterior, esta versão também utilizava animações, porém com o objetivo de destacar elementos e passos importantes do código através de cores como marcadores visuais. A ideia era fornecer pistas visuais para guiar os participantes durante o aprendizado.

3.2.2 Contexto experimental de pesquisa

Alocação dos Participantes. Embora a amostragem utilizada tenha sido por conveniência, os participantes foram alocados aleatoriamente em três grupos, seguindo um delineamento experimental fatorial (Backstrom e Kleinberg, 2011) com dois grupos experimento e um grupo controle. A aleatorização simples na alocação do experimento garante que cada participante tenha a mesma probabilidade de ser atribuído a qualquer um dos grupos, minimizando vieses e aumentando a confiabilidade dos resultados.

Coleta de dados. Aprovada pelo Comitê de Ética Número do Parecer: 6.961.536, a pesquisa foi conduzida integralmente em ambiente online de forma assíncrona entre os meses de junho e agosto de 2024, por meio da plataforma Jotform, escolhida por sua interface intuitiva e recursos de coleta de dados. Todos os materiais do experimento, incluindo um termo de consentimento detalhando os objetivos e benefícios da pesquisa, questionário socioeconômico, questionário de autoavaliação de proficiência em constructos de Python, material instrucional e o instrumento de avaliação, foram disponibilizados em um único link seguro [2]. Em média, os participantes levaram 50 minutos para concluir o experimento.

Análise de Dados. A análise dos dados foi realizada utilizando a linguagem de programação R, versão 4.4.1, no ambiente de desenvolvimento integrado RStudio, versão 2024.04.2+764. Foram empregados os pacotes tidyverse, ggplot2, scales e dplyr para manipulação, visualização e análise dos dados. A fim de comparar as médias entre os diferentes grupos, foram realizados testes de análise de variância que verificaram os pressupostos para a aplicação de um teste de análise de variância (ANOVA). A variável dependente (score dos estudantes no instrumento de pós-teste) atende ao pressuposto de uma variável contínua quantitativa e o design do experimento garantiu a independência da amostra. Como o tamanho da amostra não atende ao teorema do limite central (Stevens, 2013), foi realizado o teste de Shapiro-Wilk para determinar se a distribuição da amostra segue uma distribuição normal. O teste de Levene foi realizado para determinar se a variância em grupos diferentes é similar. Uma inspeção visual através de plots foi feita para determinar a necessidade de remoção de *outliers*. O α de significância adotado em todos os testes é

[2] `https://www.jotform.com/pt/build/240555359200653`

de ($p < 0.05$). Os tamanhos de efeito das diferenças entre os grupos foram estimados utilizando o índice de Cohen's d, permitindo uma quantificação da magnitude dessas diferenças, caso existam e sejam significativas.

3.3 Elaboração dos protótipos

Após realizar estudos acerca dos guias de designs para sistemas de visualização e aprendizado multimídia por meio de uma revisão da literatura do tema (veja Seção 2.4), elaboraram-se protótipos de visualização de Funções de Ordem-Superior, utilizando o software de criação/edição e exibição de apresentações gráficas PowerPoint. Embora limitado aos recursos do software na construção do background dos protótipos, recriou-se a aparência do JSVEE, replicando o mesmo esquema de cores, e mantendo o passo-a-passo das animações presentes na ferramenta o mais similar possível.

Com o layout do JSVEE replicado no PowerPoint, dois tipos de animações foram criados. O primeiro protótipo visou contemplar em seu design a representação intermediária dos arrays construídos durante a execução das Funções de Ordem-Superior Map, Filter e Reduce, visto que a animação é um componente multimídia, com um papel importante em auxiliar os alunos a entender tópicos complexos e abstratos, podendo simplificar um processo longo e complexo, apresentando-o passo-a-passo e tornando-o fácil de aprender (Surjono e Muhtadi, 2017). Assim, o array intermediário funciona como auxiliar externo de cognição (veja Seção 2.4) que reduz a carga cognitiva da compreensão de programas com Funções de Ordem-Superior ao apresentar concretamente a execução, fazendo com que o estudante não necessite armazenar e recobrar mentalmente o estado do array resultante, liberando recursos cognitivos anteriormente alocados à memorização do estado do array para a atividade de compreender efetivamente o processo realizado pela *FOS*.

Para cada ação, o elemento original do array de entrada será avaliado e uma saída apresentada. Por exemplo, no Map será apresentado qual o elemento do array original está sendo avaliado no momento, aplica-se a função de transformação e o valor resultante é inserido no array intermediário para o usuário poder visualizar este processo iterativo. No Filter, cada elemento avaliado será inserido ou não no array intermediário conforme a avaliação da função de filtragem (retorna true ou false). Como mostram as pesquisas de Krishnamurthi e Fisler (2020), no Reduce é fundamental que se apresente o valor do acumulador e o estado do valor resultante intermediário. A Figura 3.1 apresenta o processo de construção do protótipo do "array intermediario" no software PowerPoint. A imagem destaca os seguintes pontos: (1) id: 1 um identificador único da lista valores na memória, (2) ref: 1 referencia ao objeto com id 1,(3) id: 7 identificador único da lista filtrados na memória, (4) ref: 7 referencia ao objeto com id 7 e (5) a construção do array intermediário, representada por filter, demonstra como os resultados temporários da aplicação da função de filtragem são armazenados.

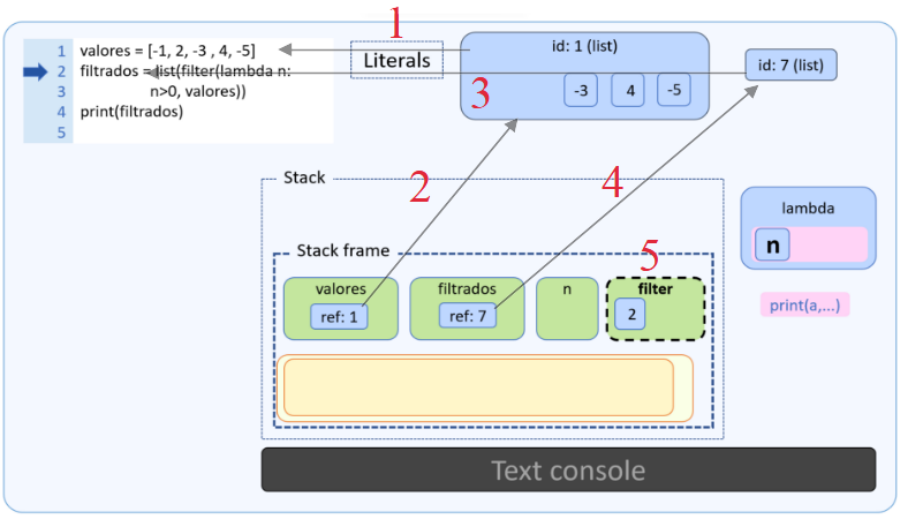


Figura 3.1: Protótipo Array Intermediário - Função Filter.

No segundo protótipo, optamos por uma visualização mais simplificada, eliminando o passo-a-passo da inserção de dados no array intermediário. Em vez disso, utilizamos pistas visuais, como cores vibrantes, para destacar elementos que satisfazem as condições das Funções de Ordem-Superior. Essa escolha se baseia na premissa de que sinais visuais podem otimizar a aprendizagem, direcionando a atenção para os aspectos mais relevantes da tarefa, como evidenciado por estudos como (Wang et al., 2020; Kong et al., 2017; de Koning et al., 2010). Ao adotar essa abordagem, introduzimos uma nova *Notional Machine* que difere daquela utilizada no protótipo anterior. Enquanto a abordagem do array intermediário enfatiza a avaliação individual de cada elemento, as pistas visuais sugerem uma aplicação simultânea da função a toda a coleção. Essa simplificação pode facilitar a compreensão do processo, mas também pode obscurecer detalhes importantes e levar a uma representação mental incompleta da operação. Embora não tenhamos encontrado na literatura *Misconceptions* específicas relacionadas a essa abordagem, nossa pesquisa busca identificar possíveis limitações e desafios cognitivos que podem surgir.

Para exemplificar, no Filter, utilizamos a convenção de cores verde e vermelho para indicar elementos que atendem ou não ao critério de filtragem. No Map, adotamos um esquema de elementos com cores distintas no array de origem, preservando a cor original de cada elemento ao longo da transformação. No Reduce, não encontramos um sistema de representação visual por cores que atendesse aos requisitos específicos da nossa proposta. A Figura 3.1 apresenta o processo de construção do protótipo ”Pistas Visuais” no software PowerPoint. A imagem destaca os seguintes pontos: (1) id: 1 um identificador único da lista valores na memória, (2) ref: 1 referencia ao objeto com id 1,(3) id: 7 identificador único da lista filtrados na memória, (4) ref: 7 referencia ao objeto com id 7 e (5) o resultado da aplicação da função de filtragem são destacados em verde para os elementos que atendem ao critério da função e em vermelho os elementos que não atendem o critério.

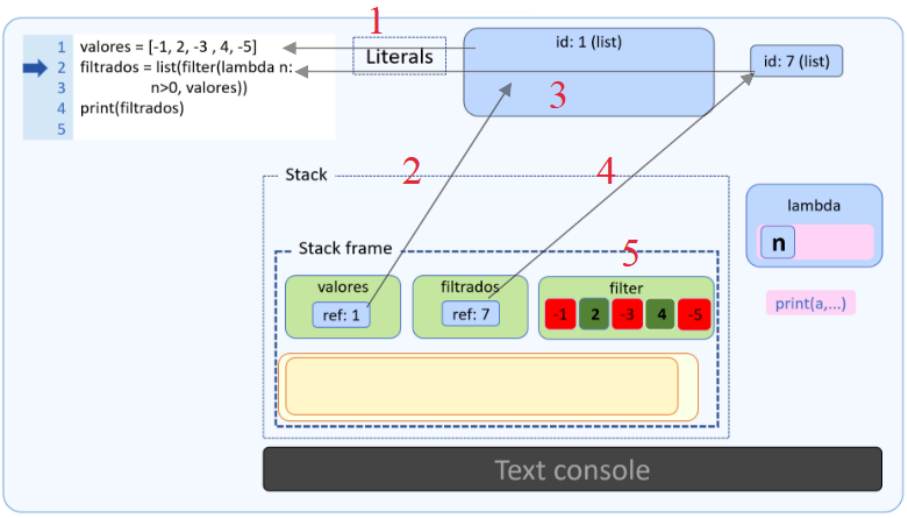


Figura 3.2: Protótipo Pistas Visuais - Função Filter.

Ao fim da prototipagem inicial no software PowerPoint, o orientador e o autor do trabalho adotaram a abordagem de desenvolvimento por experts para avaliar o processo exploratório de design. A abordagem de desenvolvimento por experts consiste em um método de criação e aprimoramento de produtos, serviços ou sistemas que se baseia no conhecimento profundo e na experiência de profissionais altamente qualificados em uma determinada área (Nielsen e Molich, 1990). Esses experts, por meio de suas habilidades técnicas e domínio do assunto, são capazes de oferecer soluções inovadoras e eficazes para os desafios complexos que surgem no processo de desenvolvimento. Assim apresentamos o protótipo a dois experts em ferramentas de visualização e simulação de programas, que atuaram como avaliadores dos protótipos: os autores da proposta original do JSVEE Juha Sorva e Teemo Sirkiä (Sirkiä e Sorva, 2015). Após uma análise detalhada, os expert's sugeriram alterações como a utilização exclusiva da cor verde para marcar elementos nos protótipos com pistas visuais. Essa recomendação visa garantir a acessibilidade do material, evitando, problemas para pessoas com deficiências visuais. Este processo de análise e refinamento repetiu-se com outras versões do protótipo.

O primeiro protótipo viável, foi apresentado no Simpósio Brasileiro de Educação em Computação (EduComp). Realizado anualmente pela Sociedade Brasileira de Computação (SBC), o EduComp é um fórum de destaque para a comunidade de Educação em Computação, reunindo pesquisadores e educadores para compartilhar novas ideias e resultados. O protótipo foi apresentado no Laboratório de Ideias, um espaço do evento dedicado à troca de ideias inovadoras e em desenvolvimento (Martins e Duran, 2023). Em uma sessão, composta por 22 pessoas, proporcionou a oportunidade de obter feedback qualificado da comunidade acadêmica, o que foi fundamental para refinar o protótipo e direcionar os próximos passos da pesquisa na confecção das animações utilizando a ferramenta de visualização e simulação de programas JSVEE (vide Seção 3.5).

3.4 Elaboração do material de Apoio a aprendizagem

Os materiais didáticos desenvolvidos neste estudo foram elaborados com o propósito de proporcionar uma compreensão aprofundada das Funções de Ordem-Superior *FOS* em Python, com ênfase nas funções Map, Filter e Reduce. Adicionalmente, os materiais exploram a Composição de Funções de Ordem-Superior, uma técnica poderosa para o processamento de dados em programação funcional, demonstrando sua aplicabilidade na manipulação concisa e expressiva de dados. A fim de facilitar a aprendizagem, os materiais incluem exemplos práticos e códigos em Python que ilustram o uso dessas funções em diversos cenários. Essa abordagem pedagógica foi cuidadosamente estruturada para atender às necessidades dos três grupos participantes da pesquisa (veja Seção 3.1).

Elaborados pelo orientador e adaptados pelo autor para a pesquisa em formato de página web, cada versão do material, apresentado para os grupos, contém os mesmos conteúdos básicos, uma pequena introdução sobre o conceito mais amplo sobre Funções de Ordem-Superior , uma introdução sobre Funções de Ordem-Superior na linguagem Python, suas propriedades mais importantes e particularidades tais como o comportamento da aplicação de Funções de Ordem-Superior em uma lista vazia. Dando seguimento é introduzido o conceito base das três funções destaques da pesquisa, Map, Filter e Reduce, seguindo de uma explicação individual de cada função, detalhando suas implementações e apresentando exemplos de aplicabilidade. Buscou-se, utilizar exemplos que se aproximassem o máximo possível de um cenário real da utilização de Funções de Ordem-Superior no processamento de dados.

Para facilitar a compreensão do conceito de Funções de Ordem-Superior, os materiais iniciais apresentados a todos os grupos incluíam uma introdução simplificada a essas funções, sem o uso de listas, conforme o exemplo 3.1. Esses exemplos introdutórios permitiram aos participantes familiarizarem-se com a ideia de funções como argumentos e valores de retorno de outras funções, preparando-os para as aplicações mais complexas com listas.

Listing 3.1: Exemplo de aplicação de Funções de Ordem-Superior sem Listas.

```
# Higher-Order Functions
#Funcao que calcula o dobro de um valor
def  dobro (valor):
  return 2 * valor

#Funcao que calcula o quadrado de um valor
def quadrado (valor):
  return valor * valor

#Funcao que calcula o negativo de um valor
def negativo (valor):
  return -1 * valor
```

```
#Funcao que aplica a funcao func ao valor
def mat_func (func, valor):
  return func(valor)

#Exemplos de Uso
print(mat_func(dobro,3))
6
print(mat_func(quadrado,3))
9
print(mat_func(negativo,3))
-3
```

Ao grupo controle apresentou-se o material básico descrito acima, acrescido de exemplos estáticos de código, os quais poderiam ser compreendidos pelos estudantes ou executados para verificar seus resultados como o exemplo 3.2. Para complementar o material do grupo controle, foram incluídos vídeos explicativos narrando a execução dos exemplos de código. Esses vídeos visavam auxiliar os participantes na visualização do processo de execução das funções e na compreensão do conceito de array intermediário.

Ao segundo grupo, o material apresentado difere do material apresentado ao grupo controle, apenas nos exemplos, os quais ao invés do código estático contém animações criadas utilizando a biblioteca JSVEE contendo a representação do Protótipo "array intermediário". Para ilustrar o funcionamento do JSVEE, um vídeo demonstrativo foi elaborado, apresentando os principais componentes da ferramenta, como os controles de animação e as áreas de visualização do código e dos dados. Para complementar a aprendizagem, ao final de cada sessão, os participantes foram convidados a responder a um quiz sobre os conceitos abordados. Enquanto para o terceiro grupo o material apresentado era composto pela animação que simplifica o passo-a-passo, contendo pistas visuais Kong et al. (2017); de Koning et al. (2010); Wang et al. (2020), utilizando cores na marcação dos elementos baseado no protótipo "pistas visuais".

Listing 3.2: Exemplo do código estático da função de Ordem-Superior Map constante nas versões do material de Apoio sem animações.

```

# temperaturas em Fahrenheit
temp_fahrenheit = [ 78, 80, 100, 98 ]

#funcao para realizar a conversao de temperatura para Celsius
def celsius (T):
   return round((float (5) / 9) * (T-32),2)
#utilizando a funcao map para converter cada temperatura da lista
 valores = list(map(celsius,temp_fahrenheit))

#Saida
```

```
print(valores)                                                    12
[25.56, 26.67, 37.78, 36.67]                                      13
```

3.5 Implementando as Animações utilizando a Biblioteca JSVEE

Em conjunto com a elaboração dos materiais didáticos, iniciou-se a fase das implementações das animações elaboradas como protótipos (vide Seção 3.3). Nesta etapa utilizamos a biblioteca JSVEE[3], uma ferramenta de código aberto que consiste em vários módulos internos que provêm funcionalidades à biblioteca, por exemplo: ações para controlar a *Notional Machine*, explicações textuais para as ações, utilitários para manipular a interface do usuário. O módulo principal ou núcleo fornece a *Notional Machine* que executa operações como buscar valor de uma determinada variável ou avaliar um operador lógico ou aritmético. A implementação real dessas operações, no entanto, encontra-se dentro de outro módulo, assim como a lógica para criar o layout da interface do usuário. O núcleo fornece meios para executar a sequência de operação na ordem correta e a capacidade de retroceder as animações usando um buffer de desfazer. Todas as informações relevantes e o estado do programa estão no modelo de objeto de documento (DOM) do navegador e, portanto, o buffer de desfazer é implementado armazenando a árvore DOM para cada etapa. Ao retroceder, a visualização atual é substituída pela árvore DOM salva anteriormente no buffer de desfazer (Sirkiä, 2016). Outras informações acerca do layout da biblioteca, utilização e outros detalhes estão disponíveis em seu repositório do GitHub.

O JSVEE produz animações que consistem em elementos HTML manipulados com JavaScript e estilizados com CSS, o que torna essas animações capazes de ser incorporadas em qualquer página da web. Como o JSVEE não oferece nenhum ambiente de execução real para nenhuma linguagem de programação, o principal desafio enfrentado no uso da biblioteca é a criação da sequência de operações que alteram o estado do programa. Existem duas opções de produção da sequência de operações: manualmente ou através da utilização de um transpiler[4], uma ferramenta que reconhece a estrutura do programa e cria a sequência de operações correspondente baseada puramente na análise estática do código, mapeando a semântica do código do programa com as operações que o JSVEE fornece. Entretanto, ainda não existe uma implementação do transpiler capaz de mapear as ações necessárias para gerar animações de Funções de Ordem-Superior. É exatamente a criação de tais ações que podem ser fornecidas posteriormente a um transpiler que é o objetivo desta etapa.

As animações das Funções de Ordem-Superior foram implementadas manualmente através da combinação das aproximadamente 50 operações existentes no JSVEE.

[3] https://github.com/Aalto-LeTech/jsvee

[4] `https://acos.cs.hut.fi/jsvee-transpiler-pythonn`

Essas operações prontas controlam a *Notional Machine* implementadas na biblioteca, como, por exemplo, acessar variáveis, avaliar operações aritméticas e lógicas, saltos condicionais, chamar funções com parâmetros e valores de retorno, bem como definir classes com métodos e criar instâncias e acessar suas variáveis de instância (Sirkiä, 2016).

As operações existentes na biblioteca são basicamente objetos JSON, e através das combinações desses objetos foi possível gerar as ações necessárias ao JSVEE definindo a aparência e o fluxo das animações contendo o passo-a-passo das operações representando cada *Notional Machine* inerente às Funções de Ordem-Superior Map, Filter e Reduce contempladas nos protótipos "array intermediário" e "pistas visuais" (vide Seção 3.3).

A construção das animações baseadas no protótipo "array intermediário", ocorreu por meio da combinação de objetos JSON, uma estratégia que se mostrou eficaz para a maioria dos exemplos do material didático. Contudo, para atender às especificidades de alguns casos, foi preciso estender as capacidades da biblioteca JSVEE. Essa expansão envolveu a criação de funções JavaScript customizadas, como upper(), lower(), capitalize(), startswith(x) (vide Listing 3.3) e outras, que mimetizam comportamentos de manipulação de strings e listas encontrados em Python. Essas funções foram integradas ao namespace JSVEE, ampliando seu conjunto de ferramentas e possibilitando a implementação completa dos exemplos propostos, os quais demandavam um nível de flexibilidade e expressividade que a biblioteca original não oferecia.

Listing 3.3: Através do código JavaScript a biblioteca JSVEE expandiu suporte a funcionalidades típicas do Python como a função startswith.

```
else if (element.hasClass('jsvee-function') && element.attr('data-name')
    === 'startswith') {

  var param = element.find('.jsvee-value').eq(0).text();
  var arg = element.find('.jsvee-value').eq(1).text().slice(0,
     param.length); // Obtem o argumento como uma string
  var result = param.startsWith(arg); // Adicione o argumento a funcao
     'startsWith()'

  var newElement = JSVEE.utils.ui.findOrCreateValue(area, result, 'str');

  ready(newElement);
```

Para criar as animações baseadas no protótipo "pistas visuais", combinamos objetos JSON e manipulamos as cores dos elementos. Nosso objetivo era tornar as visualizações mais intuitivas e destacar as particularidades de cada função. Na função Map, ao invés de usar o azul-claro padrão do JSVEE para todos os elementos de uma lista, atribuímos cores diferentes a cada elemento, facilitando sua identificação

visual. Na função Filter, à medida que a função iterava pela lista, os elementos encontrados tinham suas cores alteradas para verde, evidenciando o resultado da filtragem. Para a função Reduce, função a qual na etapa de prototipagem (vide Seção 3.3) não encontramos um sistema utilizando cor capaz de se encaixar em nossa proposta, desenvolvemos uma visualização customizada: a cor do acumulador, que representa o resultado parcial a cada iteração, sofria uma transição gradual e monocromática. Proporcionando uma representação visual da acumulação. Para implementar essas modificações visuais, utilizamos regras CSS específicas para cada Funções de Ordem-Superior de nosso estudo.

Abaixo são fornecidos alguns detalhes da implementação da animação criada para a Funções de Ordem-Superior Filter baseada no protótipo "array intermediário" (vide Seção 3.3), utilizando como exemplo um programa que filtra números positivos contidos em uma lista de números.

As animações consistem em mostrar ao usuário o código estático das Funções de Ordem-Superior, e ao seu lado a execução de cada operação contida nele. Embora na animação a sintaxe do código apresentada para o usuário seja a da linguagem Python, na biblioteca JSVEE, a implementação dessa sintaxe é feita através das operações mostradas no código em 3.4, gerando para o usuário final uma interface visual do código estático vista em 3.5.

Listing 3.4: Código utilizado no JSVEE para apresentar ao usuário o código estático em Python de um Filter de numeros positivos.

```
lines:[
      "def numeros_positivos(numeros):",
      "    return numeros > 0",
      "",
      "numeros = [-2, 12, 0, 2]",
      "",
      "filtrados = filter(numeros_positivos,numeros)",
      "\t",
      "filtrados = list(filtrados)",
      "\t",
      "print(filtrados)"
    ],
```

Listing 3.5: Código Python apresentado ao usuário como parte da animação Filter .

```
def numeros_positivos(numeros):
   return numeros > 0

numeros = [-2, 12, 0, 2]

filtrados = filter(numeros_positivos,numeros)
```

```
filtrados = list(filtrados)

print(filtrados)
```

A configuração do layout da interface do usuário para as animações, exemplificada em 3.6, segue um padrão comum em todo o material didático. A propriedade code: 'left' garante que o código, como em 3.5, seja posicionado à esquerda da interface, facilitando a visualização simultânea do código e de sua respectiva execução. A ativação do console (console: true) permite a depuração e a exibição de resultados, enquanto os controles (controls: true) possibilitam a interação do usuário com a animação. A definição de altura para a pilha (stackHeight: 100), largura dos painéis à direita (rightPanelsWidth: 150) e altura do heap (heapHeight: 80) otimiza a visualização de dados e estruturas de memória. A configuração da largura como automática para a interface (width: 'auto') e para a área do código (codeWidth: 'auto') garante uma adaptação dinâmica aos diferentes tamanhos de tela, melhorando a experiência do usuário. A disposição das classes à esquerda (classesLeft: true) e do console à esquerda e acima do heap (consoleBelowHeap: false, consoleLeft: true) contribui para uma organização visual mais clara.

Listing 3.6: Código da implementação da interface de animação JSVEE.

```
settings: {
 'code': 'left',
 'console': true,
 'controls': true,
 'stackHeight': 100,
 'rightPanelsWidth': 150,
 'heapHeight': 80,
 'width': 'auto',
 'codeWidth': 'auto',
 'classesLeft': true,
 'consoleBelowHeap': false,
 'consoleLeft': true
 },
```

O principal módulo utilizado no JSVEE é o animations. Nele encontram-se várias operações utilizadas para controlar as ações realizadas nas animações. Para mostrar o passo-a-passo da *Notional Machine* das Funções de Ordem-Superior conforme idealizamos no protótipo "array intermediário", combinamos algumas operações prontas da biblioteca as quais são utilizadas para criar funções (createFunction), para adicionar valores as váriaveis (addValueFromVariable), para avaliar o operador em uma posição determinada e substituí-lo pelo resultado (evaluateOperator "," "posição"), e até mesmo desabilitando partes de animações (disableAnimations) de modo a ocultar algumas animações que não contemplavam a visualização proposta (3.7). Por exemplo, ao copiar, valores do array original para o array intermediário, é interna-

mente executado o comando *append* no Python, que é suprimido da animação de modo a apresentar apenas implicitamente que um novo elemento é inserido no array intermediário.

Listing 3.7: Trecho do código JSVEE para controlar ações das animações.

```
steps:[
     ["createParameterVariables",["numeros"]],
     ["assignParameters",["numeros"]],
     ["setLine",1],
     ["createInstance","list"],
     ["addReference","-1","0"],
     ["assign","filter"],
     ["setLine",2],
     ["_createIterator","i1","@numeros"],
     ["disableAnimations"],
     ["runForward",18],
     ["takeNext","i1","0"],
     ["assign","numeros"],
     ["setLine",2],
     ["addValueFromVariable","numeros","0"],
     ["addOperator",">","1"],
     ["addValue","0","2","int"],
     ["evaluateOperator","1"],
     ...
     ],
```

Para cada exemplo estático de código apresentado no material de apoio, desenvolvemos animações interativas baseadas nos protótipos "array intermediário" e "pistas visuais" (detalhados na Seção 3.4). A Figura 3.3 ilustra um exemplo da animação do "array intermediário", enquanto a Figura 3.4 apresenta a animação das "pistas visuais".

Essas animações, que envolvem uma quantidade considerável de operações, foram criadas de forma personalizada para cada exemplo, resultando em uma variedade de sequências animadas.

3.6 Elaboração do Instrumento de Avaliação

O desenvolvimento do pós-teste é uma etapa importante na avaliação do aprendizado dos participantes. Para criar um pós-teste eficaz, nos inspiramos em conceitos extraídos da literatura (Krishnamurthi e Fisler, 2021; Rivera et al., 2022). Nosso objetivo foi criar um teste com a capacidade de avaliar a compreensão e a aplicação de composições de Funções de Ordem-Superior, ao mesmo tempo, em que explora a

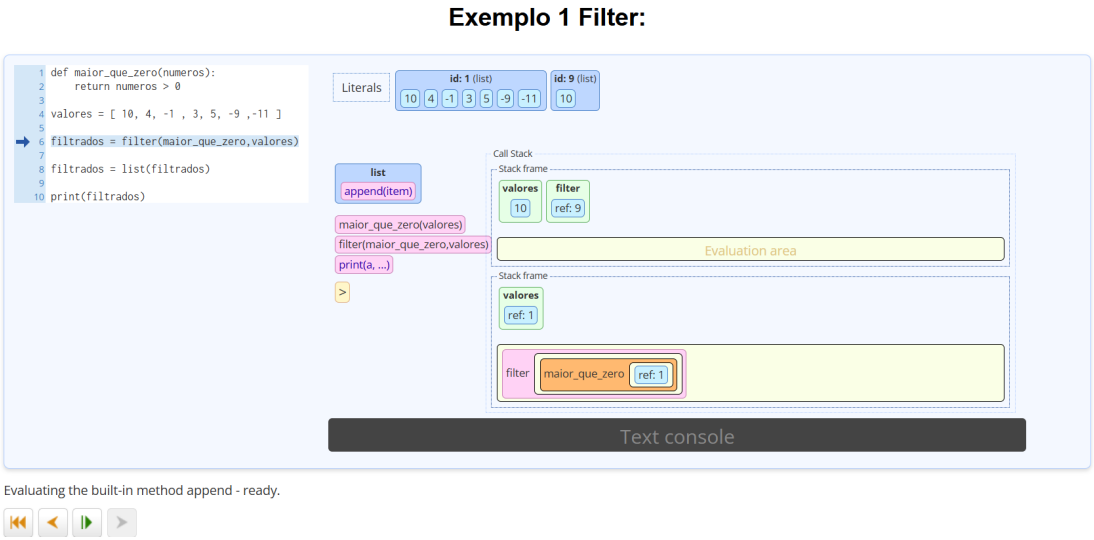


Figura 3.3: Animação do "array intermediário" ilustrando a filtragem de números positivos.

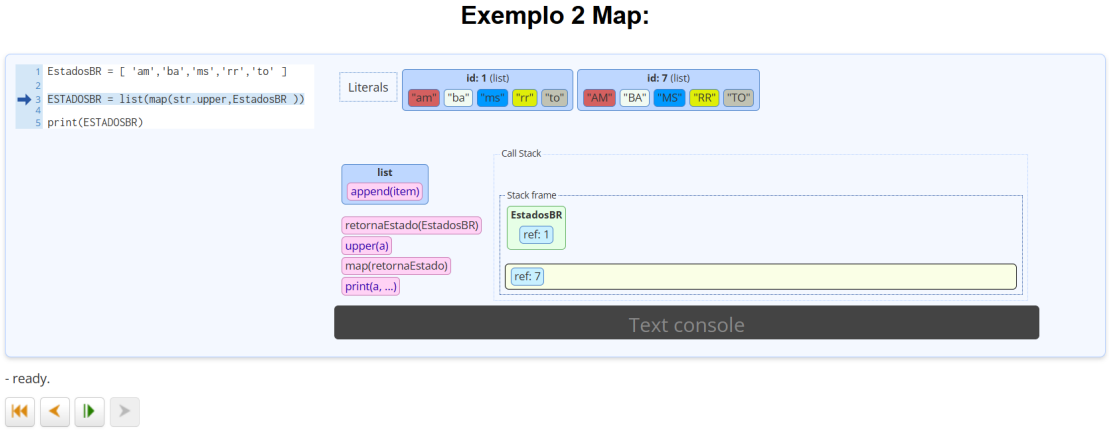


Figura 3.4: Animação das "pistas visuais" demonstrando o mapeamento dos elementos de uma lista.

capacidade da memória de trabalho dos participantes (Sweller, 1988; Duran et al., 2022) .

Nosso trabalho consistiu na elaboração de um conjunto de 8 questões para o pós-teste, objetivando avaliar a evolução dos participantes na compreensão e aplicação de Funções de Ordem-Superior(FOS). Em colaboração com professores da Aalto University autores do JSVEE, buscamos construir questões que abrangessem um espectro de dificuldade, desde as mais simples até as mais complexas.

Abordamos pelo menos duas questões para cada *FOS* contida em nosso material didático, map, filter e reduce. Além disso, elaboramos quatro questões sobre a com-

posição de *FOS*, um conceito fundamental para a programação funcional. Para avaliar a compreensão dos participantes, utilizamos exclusivamente questões de rastreio de código (Izu et al., 2019). Essas questões desafiam os estudantes a acompanhar passo-a-passo a execução de um programa, identificando as mudanças nas variáveis e o fluxo de controle. No entanto, a elaboração de questões que capturem de forma precisa os *misconceptions* (conceitos errôneos) relacionados às *FOS* mostrou-se um desafio considerável.

A revisão da literatura, combinada com nossas hipóteses, revelou uma série de *Misconceptions* sobre *FOS* que não foram devidamente reportadas em estudos anteriores. Essas *Misconceptions*, que emergiram como hipóteses a partir de nossa exploração em conjunto com os experts, sugerem a necessidade de aprofundar a investigação sobre as dificuldades específicas que os estudantes enfrentam ao aprender esse tópico. A identificação dessas novas concepções pode contribuir para a construção de um quadro mais completo das dificuldades de aprendizagem e, consequentemente, para o desenvolvimento de intervenções pedagógicas mais eficazes.

Os scores de cada estudante foram computados como a soma parcial dos acertos de cada alternativa em cada questão. Desta forma, o score de cada questão não era dicotômica, mas sim uma variável contínua resultante da soma parcial no intervalo de [0, 1]. Cada acerto era computado como o percentual de questões acertadas pelo estudante dividido pelo total de alternativas corretas. Não foram computadas penalidades para erros, apenas deixou de somar pontos. Nenhum candidato tentou burlar esta metodologia marcando todas as alternativas de uma questão.

Nossas questões foram projetadas para explorar os seguintes *Misconceptions*:

- Generalização de funções: a tendência a generalizar uma função para qualquer tipo de entrada, sem considerar casos específicos.
- Compreensão do comportamento de *FOS*: dificuldades em compreender o comportamento exato de *FOS* como “map” e “filter” em diferentes contextos.
- Valor inicial em “reduce”: a suposição de que o valor inicial do acumulador em “reduce” não afeta o cálculo da média.
- Ordem de aplicação de *FOS*: a crença de que a ordem em que as *FOS* são compostas não influencia o resultado final.

***Misconceptions* Específicos Relacionados a “filter”:**

- Compreensão do que é filtrado: o observador pode erroneamente acreditar que “filter” remove elementos da lista original, em vez de criar uma nova lista com os elementos que satisfazem uma determinada condição. A lista original permanece inalterada.
- Compreensão sobre o critério de filtragem pode variar amplamente, sendo definido por uma função que retorna um valor booleano (Verdadeiro ou Falso). Isso pode ser desafiador para iniciantes.

***Misconceptions* Específicos Relacionados a "map":**

- Natureza da função "map": "map" não modifica a lista original, mas cria uma nova lista aplicando uma função a cada elemento. A função fornecida como argumento define a transformação a ser realizada.
- Resultados possíveis: a lista de saída de "map" pode ter um tamanho diferente da lista original e pode conter elementos duplicados, dependendo da função aplicada.
- Preservação da ordem: a ordem dos elementos é mantida na lista de saída, mesmo após as transformações.

3.7 Considerações Éticas

A presente pesquisa garante a liberdade de participação, a integridade dos participantes e a preservação dos dados que possam identificá-los, assegurando a privacidade, sigilo e confidencialidade. Para tanto, as seguintes medidas foram tomadas:

1. **Consentimento Livre e Esclarecido:**

Todos os participantes receberam um Termo de Consentimento Livre e Esclarecido (TCLE) detalhado, em linguagem acessível, contendo todas as informações relevantes sobre a pesquisa, incluindo seus objetivos, métodos, riscos e benefícios. A participação na pesquisa ocorreu de forma totalmente voluntária e os participantes estavam livres para se retirar a qualquer momento, sem qualquer prejuízo.

2. **Anonimato e Confidencialidade:**

Todos os dados coletados foram anonimizados, o que significa que os participantes não foram identificados em nenhuma etapa da pesquisa. Não foram coletados dados em nenhum momento de aplicação da pesquisa capaz de identificar o participante, tais como nome, numero de matrícula ou e-mail. Os dados foram armazenados em local seguro e confidencial, acessível apenas aos pesquisadores responsáveis pelo projeto. As informações coletadas serão utilizadas apenas para fins de pesquisa e não serão divulgadas a terceiros sem a autorização expressa dos participantes.

3. **Sigilo e Segurança:**

Os dados coletados nesta pesquisa foram armazenados em ambiente seguro e confidencial na plataforma online Jotform, de propriedade da Jotform Inc. durante o tempo de 5 anos salvaguardado. A plataforma Jotform é compatível com a Carta Circular 001/2021, que estabelece os requisitos de segurança da informação para dados sensíveis O acesso aos dados será restrito aos pesquisadores diretamente envolvidos na pesquisa, que se comprometem a manter o sigilo e a confidencialidade das informações. A plataforma JotForm possui diversas medidas de segurança para garantir a confidencialidade e integridade dos dados, tais como:

- Criptografia de dados: todos os dados armazenados na plataforma são criptografados, tanto em repouso quanto em trânsito.
- Controle de acesso: o acesso à plataforma é restrito a usuários autorizados, mediante autenticação multifator.
- Monitoramento de segurança: a plataforma é monitorada 24 horas por dia, 7 dias por semana, por uma equipe especializada em segurança da informação.
- Atualizações de software: a plataforma é atualizada regularmente com as últimas correções de segurança.

4. **Princípios Éticos:**

A pesquisa utilizou os seguintes métodos e procedimentos:

A aplicação do questionário da pesquisa " Ferramentas de Visualização de Programas na Compreensão de Funções de Ordem-Superior" conforme os princípios e diretrizes da Carta Circular 001/2021 do Conselho Nacional de Saúde (CONEP), que trata da pesquisa com seres humanos em ambiente virtual. O questionário on-line foi divulgado durante as aulas presenciais da disciplina ministrada pelo Orientador da Pesquisa o Prof. Dr. Rodrigo Duran no IFMS Campus Nova Andradina no curso de nível superior de Tecnologia em Análise e Desenvolvimento de Sistemas, durante as aulas do Prof. Dr. Vitor Sarinho regente da disciplina de Jogos Digitais do curso de nível superior de Engenharia da Computação na UEFS, em listas de transmissão de grupos de pesquisa (tais como a lista da comunidade de Educação em computação) e em eventos como o Congresso da Sociedade Brasileira de Computação (CSBC), a qual a supervisão da aplicação ficou sob responsabilidade do autor da pesquisa.

Apresentação Clara e Acessível: O TCLE (Registro de Consentimento Livre e Esclarecido) apresentado no corpo da consulta está claro, conciso e acessível aos participantes. Fornecendo todas as informações necessárias para que os participantes tomem uma decisão consciente sobre sua participação na pesquisa, incluindo:

Objetivo da pesquisa: A pesquisa visa investigar o potencial de ferramentas de visualização para facilitar a compreensão de Funções de Ordem Superior (HOFs) na programação.

Metodologia: A pesquisa envolve a aplicação de um questionário online com duração estimada de cinco minutos, seguido de uma breve avaliação de conhecimentos de programação.

Riscos e Benefícios: O TCLE informa que a pesquisa não apresenta riscos identificados e destaca os benefícios potenciais para os participantes, comunidade e sociedade.

Direitos dos Participantes: O TCLE esclarece os direitos dos participantes, incluindo o direito de recusar-se a participar, retirar o consentimento a qualquer momento e solicitar a exclusão de seus dados.

Armazenamento Seguro dos Dados: O TCLE informa que os dados coletados serão armazenados em local seguro e confidencial, acessível apenas aos pesquisadores responsáveis, por um período de cinco anos.

Coleta Eletrônica Segura: O questionário será aplicado através de uma plataforma online segura, a JotForm, que atende aos requisitos de segurança da informação da UEFS e da Carta Circular 001/2021.

Armazenamento Confidencial: Os dados coletados serão armazenados em ambiente seguro na plataforma JotForm e acessíveis apenas aos pesquisadores responsáveis.

Exclusão de Dados: Os dados coletados serão excluídos de forma segura e definitiva após cinco anos, conforme a Carta Circular 001/2021. Análise de dados: os dados coletados serão analisados de forma rigorosa e científica, utilizando métodos estatísticos e qualitativos apropriados

6. **Considerações Éticas Adicionais:**

A pesquisa será conduzida conforme os princípios éticos estabelecidos na Resolução CNS nº 466/2012 e na Norma Operacional 001/2013 do Conselho Nacional de Saúde (CNS). O Comitê de Ética em Pesquisa (CEP) da Universidade Estadual de Feira de Santana será responsável por avaliar e aprovar o projeto de pesquisa antes de sua execução. Os pesquisadores se comprometem a manter um diálogo aberto e transparente com os participantes durante todo o processo de pesquisa.

7. **Monitoramento e Avaliação:**

O projeto de pesquisa será monitorado e avaliado periodicamente para garantir que os princípios éticos estejam sendo cumpridos.

8. **Responsabilidade dos Pesquisadores:**

Os pesquisadores responsáveis pelo projeto assumem a responsabilidade de garantir a ética e a segurança da pesquisa, zelando pela proteção dos direitos e da integridade dos participantes.

9. **Divulgação dos Resultados:**

Os resultados da pesquisa serão divulgados de forma ampla e transparente, por meio de publicações científicas, apresentações em eventos acadêmicos e outros meios de comunicação.

10. **Impacto Social:**

A pesquisa espera contribuir para o avanço do conhecimento na área de Educação em Computação e gerar resultados que possam ter um impacto positivo na sociedade.

11. **Considerações Finais:**

A presente pesquisa está comprometida com a condução ética e responsável da investigação, assegurando a proteção dos direitos e da integridade dos participantes.

Capítulo 4

Resultados

O presente estudo objetiva investigar principalmente o impacto da visualização e simulação de *FOS* em Python, especificamente das funções Filter, Map e Reduce, e da composição de *FOS*, sobre a compreensão e aplicação desses conceitos por indivíduos iniciantes em programação. Para tanto, foi desenvolvido um material didático e um conjunto de animações interativas, implementadas na biblioteca JSVEE, com base em protótipos elaborados durante a pesquisa. Após a aplicação do material didático, ambos os grupos realizaram uma avaliação desenvolvida para compor esse estudo com intuito de medir a compreensão dos conceitos. Os resultados obtidos permitiram avaliar a efetividade dessas ferramentas visuais na aprendizagem dos conceitos propostos.

4.1 Material Didático para FOS em Python

Como parte deste estudo resultou a criação de um material didático projetado para facilitar a compreensão de Funções de Ordem-Superior *FOS* em Python por iniciantes em programação. O material explora de forma detalhada as funções map, filter e reduce, além de apresentar a técnica de composição de funções como uma ferramenta essencial para escrever código mais conciso e eficiente.

Uma das principais inovações do nosso material é a utilização de exemplos práticos e animações para ilustrar conceitos complexos de forma clara e intuitiva. Além disso, o material foi cuidadosamente elaborado para atender às necessidades de iniciantes em programação, com uma linguagem acessível e códigos bem comentados.

Disponível em formato de página web, o material inclui 14 exemplos práticos que cobrem uma ampla variedade de aplicações das *FOS*. Ao explorar esses exemplos, os usuários poderão consolidar seus conhecimentos e desenvolver habilidades práticas para utilizar *FOS* em seus próprios projetos. A seguir, apresenta-se um exemplo 4.1, que demonstra o uso da função “map” para extrair os nomes de usuários a partir de seus endereços de e-mail:

Listing 4.1: Exemplo da Função Map contida no Material Didático.

```
def obter_nome(email):
    return email.split(''@'')[0]

emails = [ ''joaosilva@email.com'', ''mariagomes@email.com.br'',
    ''pedrodutra@email.com'' ]

nomes = list(map(obter_nome,emails))

print(nomes)
#Saida
['joaosilva','mariagomes','pedrodutra']
```

Neste exemplo, a função "map" demonstra sua utilidade ao aplicar a função obter_nome a cada endereço de email na lista emails. O resultado é uma nova lista contendo apenas os nomes dos usuários, evidenciando a capacidade do "map" de transformar cada elemento de uma lista de acordo com uma função específica.

Além da sintaxe, o material aprofunda os conceitos por trás da *FOS*, enfatizando sua natureza imutável. É explicado que o "map" não modifica a lista original, mas cria uma nova lista com os elementos transformados, preservando a mesma quantidade de elementos. A explicação também aborda o comportamento do "map" quando aplicado a listas vazias, destacando que o resultado será uma lista vazia.

O material demonstra a potência da composição de Funções de Ordem-Superior *FOS* ao abordar problemas mais complexos. Um exemplo ilustrativo 4.2 aborda a combinação das funções "map" e "filter" para identificar temperaturas superiores a 30°C em uma lista de valores em Fahrenheit. Essa composição permite resolver o problema de forma elegante e concisa, evidenciando a flexibilidade e expressividade das *FOS*.

Listing 4.2: Exemplo de Composição de Funções contida no Material Didático.

```
# temperaturas em Fahrenheit
temp_fahrenheit = [ 78, 80, 100, 98 ]
# funcao para realizar a conversao de temperatura para Celsius
def celsius (T):
   return round((float (5) / 9) * (T-32),2)

# funcao para encontrar altas temperaturas
def temperaturas_altas(x):
   return x > 30

resultado = list(filter(temperaturas_altas,map(celsius,temp_fahrenheit)))

#Saida
print(resultado)
```

```
[37.78, 36.67]
```

Neste exemplo, a função “map” é utilizada para converter as temperaturas de Fahrenheit para Celsius, e a função “filter” é aplicada para selecionar apenas as temperaturas superiores a 30°C.

Espera-se que este material sirva como base para outras pesquisas relacionadas ao uso de *FOS*, inclusive com sua tradução para outros idiomas, o que nos permitiria atingir outros contextos e uma amostra maior de participantes.

4.2 Animações de Funções de Ordem-Superior

Uma contribuição relevante deste estudo reside no desenvolvimento de animações interativas que visualizam o funcionamento das *FOS* “map”, “filter” e “reduce”, bem como da composição de funções. Essas animações, baseadas em códigos que simulam situações reais, podem ser facilmente incorporadas em qualquer material web e servem como base para uma futura implementação de um transpiler[1] (Ferramenta que analisa código Python 3 e retorna o código em formato que a biblioteca JSVEE entende).

A Figura 4.1 apresenta uma sequência de animações que detalham o funcionamento da função “filter” em Python. A Figura 4.1(a) inicia a sequência, ilustrando a chamada da FOS “filter” e demonstrando como uma função pode ser utilizada como argumento dentro de outra função. Em seguida, a Figura 4.1(b) mostra a criação de um array intermediário que armazenará os resultados parciais da filtragem. A Figura 4.1(c) visualiza o processo de inclusão dos elementos que atendem ao critério de filtragem nesse array intermediário. Por fim, a Figura 4.1(d) apresenta o resultado final da aplicação da *FOS* “filter”, exibindo a lista com os números positivos filtrados. Essa sequência de animações proporciona uma visualização clara e passo a passo do processo de filtragem, facilitando a compreensão do conceito.

[1] https://github.com/acos-server/acos-jsvee-transpiler-python

(a) Chamada da Função Filter.

(b) Criando Array Intermediário

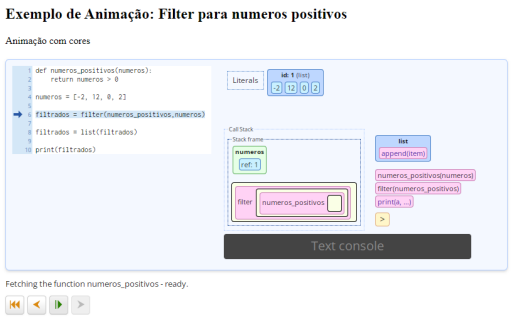
(c) Adicionando resultados ao array intermediário

(d) Apresentando o resultado ao usuário

Figura 4.1: Animação Array Intermediário *FOS* Filter.

As animações desenvolvidas para o protótipo "Pistas Visuais" foram simplificadas intencionalmente, com um número reduzido de frames. Essa decisão visou priorizar a pista visual baseada em cores, como exemplificado na Figura 4.2. Ao omitir detalhes menos relevantes, conseguimos destacar a cor como o elemento central da animação, visando facilitar a compreensão do conceito por parte do usuário.

Nessa sequência de animação destacamos, a Figura 4.2(a) focando na chamada da *FOS* "filter" utilizando outra função como parâmetro ou argumento da *FOS*, em seguida a Figura 4.2(b) demonstra a aplicação de "Filter" à lista de elementos, porém sem destaque para um elemento auxiliar como ocorre na animação "array intermediário", na Figura 4.2(c) evidenciamos o resultado da aplicação a todos os elementos (ocorrendo de uma só vez na animação) aplicando o verde aos elementos filtrados e por fim a figura 4.2(d) exemplifica a apresentação do resultado ao usuário.



(a) Chamada da Função Filter.

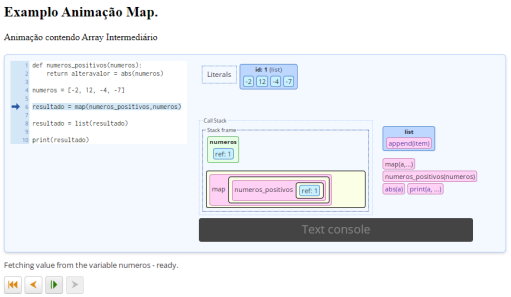
(b) Aplicando Filter a lista numeros

(c) Marcando resultados filtrados com cores

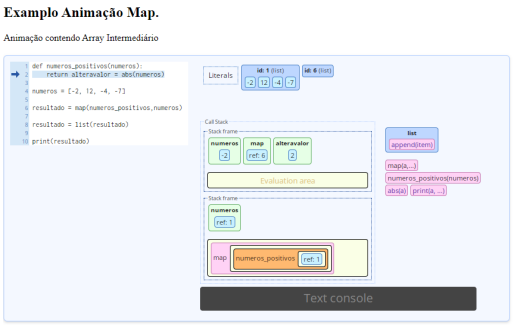
(d) Apresentando o Resultado ao usuário

Figura 4.2: Animação Pistas Visuais *FOS* Filter.

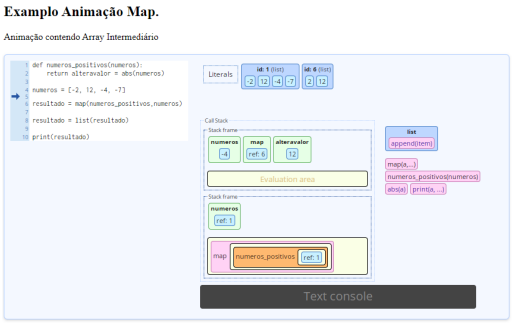
Para demonstrar que as animações contemplando o "Array Intermediário" seguem um padrão independente da *FOS*, a sequência de animações para "map", é apresentada na figura 4.3. Os passos detalhados se assemelham com os já demonstrados na figura 4.1 , destacando a chamada da *FOS* "map" e seus parâmetros na figura 4.3(a). A criação do array intermediário na figura 4.3(b), a adição dos resultados parciais ao array intermediário figura 4.3(c) e a apresentação do resultado da execução do código ao usuário na figura 4.3(d) .



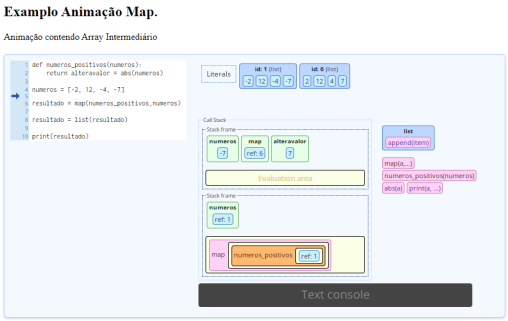
(a) Chamada da Função Map.



(b) Criando Array Intermediário



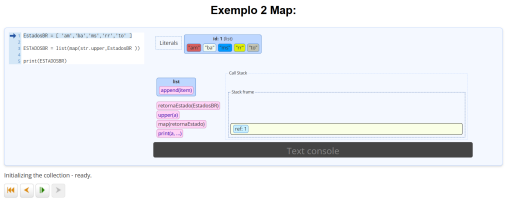
(c) Adicionando resultados ao array intermediário



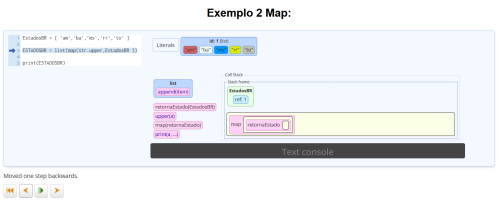
(d) Resultados adicionados ao array intermediário

Figura 4.3: Animação Array Intermediário *FOS* Map.

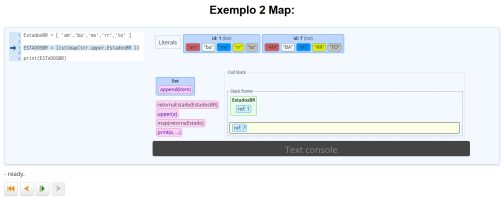
Entretanto, as animações desenvolvidas contemplando as "pistas visuais", possuem detalhes diferentes como mencionados em 3.5. Como se observa na sequência de animações retratada na figura 4.4 destacamos a apresentação dos elementos da lista original, sendo apresentados cada um mediante uma cor distinta conforme a figura 4.4(a), mantendo o padrão das demais animações. A chamada da *FOS* "map" semelhante às demais animações, aqui também evidenciamos a utilização de outra função como argumento ou parâmetro da *FOS* figura 4.4(b), em seguida mostramos o processo de transformação sendo aplicado ao mesmo tempo, em todos os elementos da lista original e o resultado da transformação mantendo o mesmo esquema de cores distintas de cada elemento na nova lista conforme figura 4.4(c) e a apresentação do resultado da execução do código ao usuário,conforme Figura 4.4(d).



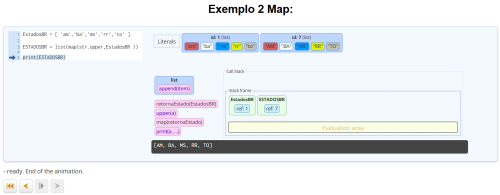
(a) Apresentando elementos da lista original.



(b) Chamada da Função Map



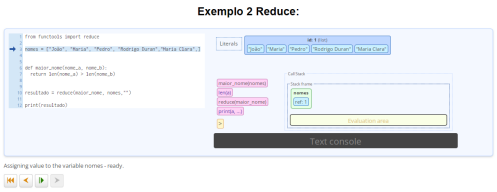
(c) Elementos transformados mantendo as cores da lista original



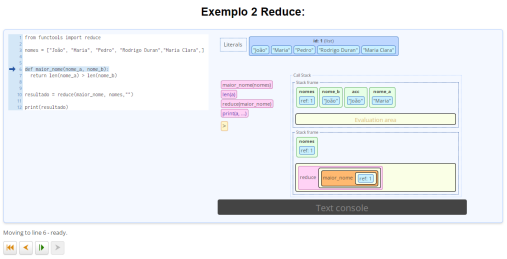
(d) Apresentação do Resultado ao usuário

Figura 4.4: Animação Pistas Visuais *FOS* Map.

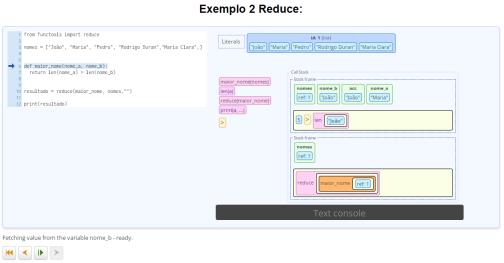
As animações criadas a partir do protótipo "Array Intermediário", que demonstram o funcionamento da *FOS* Reduce, seguem a seguinte sequência: Inicialmente, é apresentada a lista original de elementos Figura 4.5(a), seguido da chamada da *FOS* Reduce com seus argumentos uma função de redução, a lista de elementos e o acumulador Figura 4.5(b), na Figura 4.5(c) destacamos a criação do array intermediário que na *FOS* Reduce é o parametro acumulador "acc" utilizado para armazenar os resultados parciais da execução da *FOS* e por fim a Figura 4.5(d) apresenta o resultado ao final da execução da *FOS* Reduce ao usuário.



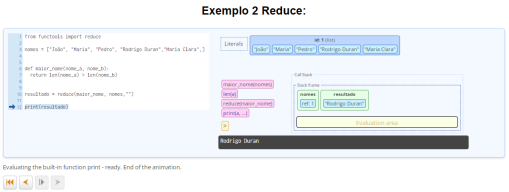
(a) Apresentando elementos da lista original.



(b) Chamada da Função Reduce



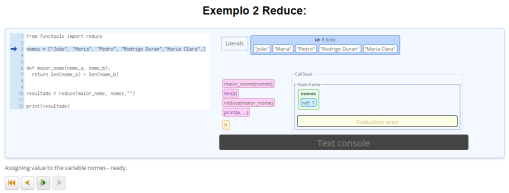
(c) Destaque para a criação do array intermediário e do array do acumulador (ACC)



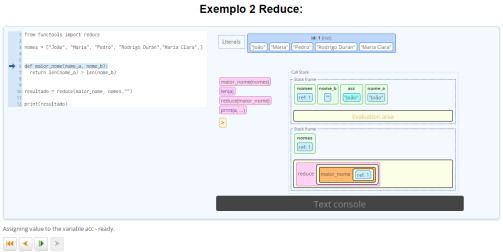
(d) Apresentação do Resultado ao usuário

Figura 4.5: Animação Array Intermediário *FOS* Reduce.

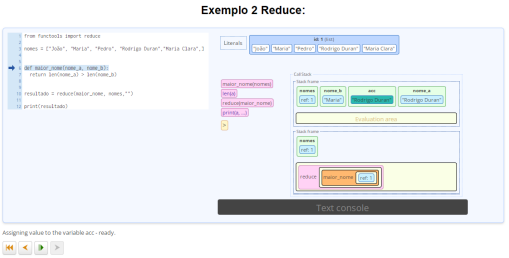
Já as animações desenvolvidas para a *FOS* Reduce contemplando "Pistas Visuais", seguem a seguinte sequência: Inicialmente, é apresentada a lista original de elementos Figura 4.6(a), seguido da chamada da *FOS* Reduce com seus argumentos uma função de redução, a lista de elementos e o acumulador Figura 4.6(b), na Figura 4.6(c) destacamos o parâmetro acumulador "acc" utilizado para armazenar os resultados parciais da execução da *FOS* que nessa animação sofre um aumento gradativo da tonalidade azul a cada iteração com a lista e por fim a Figura 4.5(d) apresenta o resultado ao final da execução da *FOS* Reduce ao usuário.



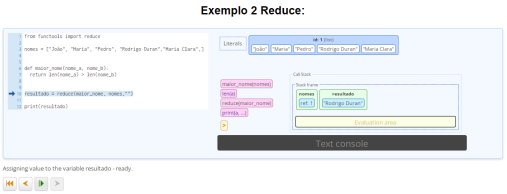
(a) Apresentando elementos da lista original.



(b) Chamada da Função Reduce e seus argumentos



(c) Acumulador com cor destacada



(d) Apresentação do Resultado ao usuário

Figura 4.6: Animação Pistas Visuais *FOS* Reduce.

4.3 Instrumento de Avaliação sobre FOS

Com o objetivo de verificar a efetividade do material didático e identificar possíveis *misconceptions* relacionadas a Funções de Ordem Superior *FOS*, aplicamos um pós-teste (Instrumento de avaliação) composto por 8 questões com diferentes graus de dificuldade, envolvendo as Funções de Ordem-Superior contidas no material didático e a técnica de composição de funções.

O pós-teste foi estruturado para avaliar a capacidade dos participantes de:

- **Interpretar visualmente.** Duas questões utilizaram representações visuais (formas geométricas coloridas) para avaliar a compreensão intuitiva da Funções de Ordem-Superior "filter" em um contexto mais abstrato.
- **Análise de código.** As seis questões restantes apresentaram trechos de código para avaliar a habilidade de raciocinar sobre a execução de programas e prever os resultados, essas questões englobaram as *FOS* "map", "filter" e "reduce" isoladamente ou abordaram a técnica de composição de funções.

Visando avaliar de forma abrangente a compreensão dos participantes sobre os conceitos de Funções de Ordem Superior *FOS* apresentados no material didático, aplicamos um pós-teste composto por oito questões. As questões foram cuidadosamente elaboradas para abordar, de forma isolada, cada uma das *FOS* estudadas, "map", "filter" e "reduce", além da técnica de composição de funções. A dificuldade das questões foi mesclada para contemplar os diferentes níveis de proficiência dos participantes. A seguir, apresentamos os enunciados de cada questão e as alternativas

contidas em nosso instrumento avaliativo, as alternativas foram identificadas de A1 a A11, para fins de análise.

Questão 1: Analise a figura 4.7 e assinale qual imagem representa a saída do código "filter"

Figura 4.7: Questão 1. Pós-Teste

Questão 2: Analise a figura 4.8 e assinale qual imagem representa a saída do código "filter"

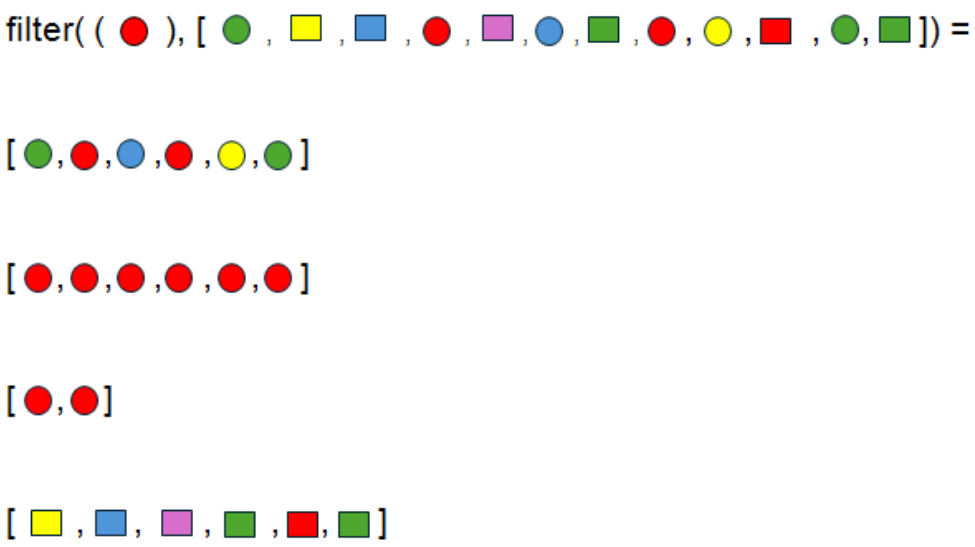


Figura 4.8: Questão 2. Pós-Teste

Questão 3. Analise o código e responda:

Observação: A função split transforma uma string em array. Quando não é usado nenhum parâmetro como separador, o padrão é separar as strings pelo espaço em branco. O índice -1 em Python retorna o último elemento de uma lista.

Listing 4.3: Código Contido na Questão 3.

```
def fn(nc):                                                    1
  ns = nc.split()                                              2
  return f''{ns[-1]}, {ns[0]}''                                3
                                                               4
nomes = [''Joao da Silva'', ''Maria Oliveira'', ''Pedro Souza'']  5
                                                               6
nf = list(map(fn, nomes))                                      7
print(nf)                                                      8
```

Alternativas Questão 3: Sobre o código 4.3, assinale as opções corretas:

A1 O conteúdo impresso na linha 8 (print(nf)) é uma lista contendo cada nome completo formatado no estilo ”Sobrenome, Nome”.

A2 A variável nomes contém uma lista vazia após a execução da linha 7.

A3 Após a execução da linha 8, será impresso os valores [’Silva, Joao’, ’Oliveira, Maria’, ’Souza, Pedro’].

A4 Após a execução da linha 7, a variável nomes contém os valores [’Silva, Joao’, ’Oliveira, Maria’, ’Souza, Pedro’].

A5 Após a execução da linha 7, a variáveis nomes contém os valores [”Joao da Silva”, ”Maria Oliveira”, ”Pedro Souza”].

A6 O código não roda, pois existe um erro no código.

A7 Nenhuma das anteriores

Questão 4. Analise o código e responda:

Listing 4.4: Código Contido na Questão 4.

```
def vc(produto):
  return produto[''categoria''] == ''Camisa''

def p(produto):
  return 50 <= produto[''preco''] <= 80

def ad4(produto):
  return produto[''avaliacao''] > 4

def inc10(produto):
  produto[''avaliacao''] = produto[''avaliacao''] * 1.1
  return produto

produtos = [
    {''nome'': ''Camisa Verde'', ''categoria'': ''Camisa'', ''preco'':
       55, ''avaliacao'': 4.7},
    {''nome'': ''Camisa Manga longa'', ''categoria'': ''Camisa'',
       ''preco'': 65, ''avaliacao'': 4},
    {''nome'': ''Calca Jeans'', ''categoria'': ''Calca'', ''preco'': 75,
       ''avaliacao'': 4.2},
    {''nome'': ''Tenis Running'', ''categoria'': ''Tenis'', ''preco'':
       90, ''avaliacao'': 4.8},
    {''nome'': ''Bone Esportivo'', ''categoria'': ''Acessorios'',
       ''preco'': 20, ''avaliacao'': 3.9},
    {''nome'': ''Camisa Polo'', ''categoria'': ''Camisa'', ''preco'': 68,
       ''avaliacao'': 3.9},
]

pf1 = list(filter(vc, filter(p,filter(ad4, produtos))))
print(pf1)
pf2 = list(filter(p, filter(ad4,filter(vc, produtos))))
print(pf2)
pf3 = list(filter(vc, filter(p,filter(ad4, map(inc10,produtos)))))
print(pf3)
```

Alternativas Questão 4: Sobre o código 4.4, assinale as opções corretas:

A1 A saída impressa após a execução da linha 24 é a média de preços dos produtos

que são camisas, com preços entre 50 e 80 reais.

A2 A saída impressa após a execução da linha 24 é ['nome': 'Camisa Verde', 'categoria': 'Camisa', 'preço': 55, 'avaliação': 4.7].

A3 A saída impressa após a execução da linha 26 é ['nome': 'Camisa Verde', 'categoria': 'Camisa', 'preço': 55, 'avaliação': 4.7, 'nome': 'Camisa Manga longa', 'categoria': 'Camisa', 'preço': 65, 'avaliação': 4.0].

A4 O array produtos está vazio após a execução da linha 25.

A5 O array produtos contém os mesmos elementos de sua inicialização (linha 14) após a execução da linha 25.

A6 As variáveis pf1 e pf2 possuem os mesmos valores.

A7 As variáveis pf1 e pf3 possuem os mesmos valores.

A8 A saída impressa após a execução da linha 28 é ['nome': 'Camisa Verde', 'categoria': 'Camisa', 'preço': 55, 'avaliação': 5.687000000000001, 'nome': 'Camisa Manga longa', 'categoria': 'Camisa', 'preço': 65, 'avaliação': 4.840000000000001, 'nome': 'Camisa Polo', 'categoria': 'Camisa', 'preço': 68, 'avaliação': 4.719].

A9 A saída impressa após a execução da linha 28 é ['nome': 'Camisa Verde', 'categoria': 'Camisa', 'preço': 55, 'avaliação': 5.170000000000001, 'nome': 'Camisa Manga longa', 'categoria': 'Camisa', 'preço': 65, 'avaliação': 4.4, 'nome': 'Camisa Polo', 'categoria': 'Camisa', 'preço': 68, 'avaliação': 4.29].

A10 O código não roda, pois existe um erro no código.

A11 Nenhuma das anteriores.

Questão 5. Analise o código e responda:

Listing 4.5: Código Contido na Questão 5.

```
from functools import reduce

def calcular_media(acc, nota):
    return acc + nota

notas = [10, 8, 7, 9, 6]
media_notas = reduce(calcular_media, notas, 0) / len(notas)

print(''Media das notas:'', media_notas)
```

Alternativas Questão 5: Sobre o código 4.5, assinale as opções corretas:

A1 O valor do parâmetro acc é inicializado com 0.

A2 O valor do parâmetro acc é inicializado com -1.

A3 O valor do parâmetro acc é inicializado com o primeiro elemento da lista.

A4 Ao executar a função calcular_media pela primeira vez, o valor do parâmetro acc é 0 e do parâmetro nota é 10.

A5 Ao executar a função calcular_media pela última vez, o valor do parâmetro acc é 34 e do parâmetro nota é 6.

A6 A variável media_notas armazena uma lista com os valores [0, 10, 18, 25, 34].

A7 A variável notas está vazia após a execução da linha 7.

A8 A variável notas possui os mesmos elementos com que foi inicializada (na linha 6) após a execução da linha 7.

A9 Quando a função calcular_media possui o parâmetro nota o valor 7, o parâmetro acc possui o valor 10.

A10 O código não roda, pois contém um erro.

A11 Nenhuma das anteriores.

Questão 6. Analise o código e responda:

Listing 4.6: Código Contido na Questão 6.

```
def transforma(num):                                    1
  if num > 0:                                           2
    return num * -1                                     3
  else:                                                 4
    return num                                          5
                                                        6
numeros = [1, 2, 3, -4, 5, 6, 7, -8, 9, 10]             7
                                                        8
resultado = list(map(transforma,numeros))               9
                                                        10
print(resultado)                                        11
```

Alternativas Questão 6: Sobre o código 4.6, assinale as opções corretas:

A1 O conteúdo impresso na linha 11 é [-1, -2, -3, -5, -6, -7, -9, -10].

A2 O conteúdo impresso na linha 11 é [-1, -2, -3, -4,-5, -6, -7, -8, -9, -10].

A3 A variável números está vazia após a execução da linha 9.

A4 A variável números contém 10 elementos após a execução da linha 9.

A5 O código não roda, pois contém um erro.

A6 Nenhuma das anteriores.

Questão 7. Analise o código e responda:

Listing 4.7: Código Contido na Questão 7.

```
def positivo(num):
  return num > 0

def adiciona(num):
  return num + 1

numeros = [-2, -1, 0, 1, 2]


resultado = list(filter(positivo,(map(adiciona,numeros))))

print(resultado)

resultado2 = list(map(adiciona,(filter(positivo,numeros))))

print(resultado2)
```

Alternativas Questão 7: Sobre o código 4.7, assinale as opções corretas:

A1 O conteúdo impresso na linha 12 é [2, 3].

A2 O conteúdo impresso na linha 12 é [1,2, 3].

A3 O conteúdo impresso na linha 16 é [2, 3].

A4 O conteúdo impresso na linha 16 é [1,2, 3].

A5 Na linha 10, todos os elementos da lista números são filtrados e depois alterados.

A6 Na linha 10, todos os elementos da lista números são alterados e depois filtrados.

A7 O código não roda, pois contém um erro.

A8 Nenhuma das alternativas.

Questão 8. Analise o código e responda:

Listing 4.8: Código Contido na Questão 8.

```
from functools import reduce

def s(a, v):
    return a + v

def f(v):
    return v >= 0

#A funcao listaAteOSentinela percorre toda a lista at encontrar o
    sentinela
#e retorna uma lista com todos os elementos anteriores como resultado
```

```
def listaAteOSentinela(lst, sentinela):                                    11
    result = []                                                            12
    for item in lst:                                                       13
        if item == sentinela:                                              14
          break                                                            15
        result.append(item)                                                16
    return result                                                          17
                                                                           18
valores = [0, 20, -3, 10, 0, -1, -5, 13, -3, -9, 7, 10, 20, 99999, -2, 4, 19
    3]
                                                                           20
sl = list(filter(f, listaAteOSentinela(valores, 99999)))                   21
                                                                           22
if sl: #verifica se a lista sl no est vazia                                23
    a = reduce(s, sl, 0) / len(sl)                                         24
    print(a)                                                               25
else:                                                                      26
    print(''No non-negative elements found in the list.'')                 27
```

Alternativas Questão 8: Sobre o código 4.8, assinale as opções corretas:

A1 A variável sl contém os valores [0, 20, 10, 0, 13, 7, 10, 20] após a execução da linha 21.

A2 A variável sl contém os valores [20, 10, 13, 7, 10, 20] após a execução da linha 21.

A3 Na linha 25 é impresso o valor 10.

A4 Na linha 25 é impresso o valor 13.33333.

A5 Na linha 25 são impressos os valores [0, 20, 10, 0, 13, 7, 10, 20].

A6 Na linha 25 são impressos os valores [20, 10, 13, 7, 10, 20].

A7 O código não roda, pois contém um erro.

A8 Nenhuma das anteriores.

4.4 Resultados da Aplicação do Experimento

Para avaliar o impacto de diferentes simulações e visualizações de programas na compreensão de *FOS* e identificar possíveis equívocos, conduzimos um experimento com 21 participantes. Os participantes foram divididos aleatoriamente em três grupos: o Grupo 0 (controle), com 8 participantes, que não recebeu nenhum tipo de visualização; o Grupo 1 (Array Intermediário), com 11 participantes, que recebeu material contendo animações contemplando as representações intermediárias dos resultados; e o Grupo 2 (Pistas Visuais), com 2 participantes, que recebeu animações contendo

cores como pistas visuais mais diretas. Infelizmente, devido ao tamanho da amostra não foi possível garantir um tamanho uniforme de amostras em cada grupo através do método de alocação randômica utilizado no formulário.

A média de score do experimento foi de relativamente baixa (m = 4.92, mediana = 4.06, desvio padrão = 2.44, min = 1.25, max = 9.6). Ao analisarmos os dados do grupo controle, temos um desempenho levemente superior ao grupo utilizando arrays intermediários, mas ambos foram inferiores ao score do grupo com pistas visuais em cores (vide Tabela 4.1).

Grupo	média	mediana	desvio padrão	range
0	5.39	3.95	2.84	[2.29, 9.6]
1	4.1	4.06	2.02	[1.25, 8.12]
2	6.97	6.97	0.43	[6.66, 7.27]

Tabela 4.1: Estatísticas descritivas por grupo de análise.

Para analisar os dados, realizamos uma análise de variância (ANOVA). Antes de aplicar essa técnica, verificamos se as premissas para sua utilização estavam sendo atendidas. Avaliamos a normalidade dos dados mediante análise gráfica (veja Figura 4.9) e do teste de Shapiro-Wilk, o qual apresentou um p-value de 0.26 ($W = 0.94$), superior ao nível de significância de $\alpha = 0.05$, então nós *não* rejeitamos a hipótese de que os residuais seguem uma distribuição normal.

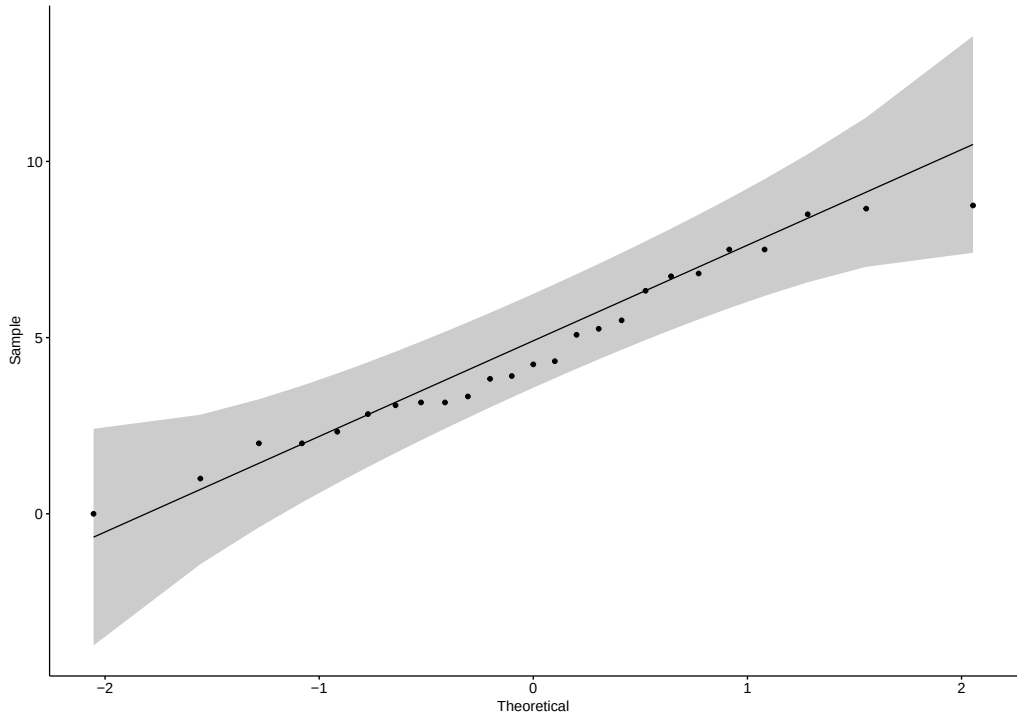


Figura 4.9: Distribuição de residuais.

A premissa de homogeneidade de variâncias é fundamental para a validade de muitos testes estatísticos paramétricos, como a ANOVA. Para avaliar essa premissa em nossos dados, empregamos uma abordagem combinada de análise gráfica e testes estatísticos. Inicialmente, realizamos uma análise exploratória dos dados por meio de gráficos de boxplot e dotplot (Figuras 4.10 e 4.11). A inspeção visual desses gráficos sugeriu, preliminarmente, a possibilidade de homogeneidade de variâncias entre os grupos, evidenciada por diferenças na dispersão dos dados.

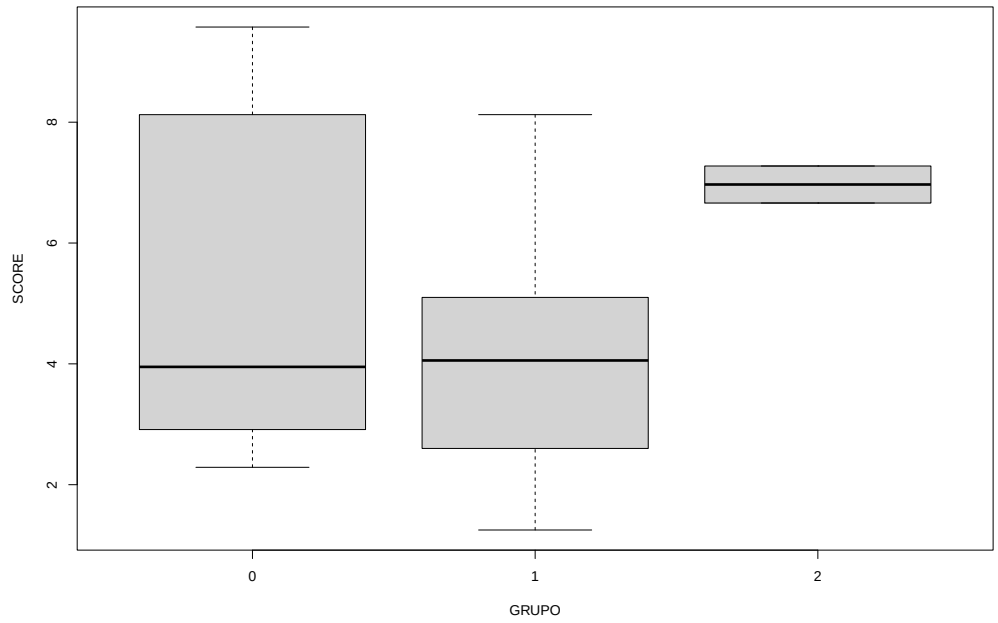


Figura 4.10: Boxplot Homogeneidade dos Grupos

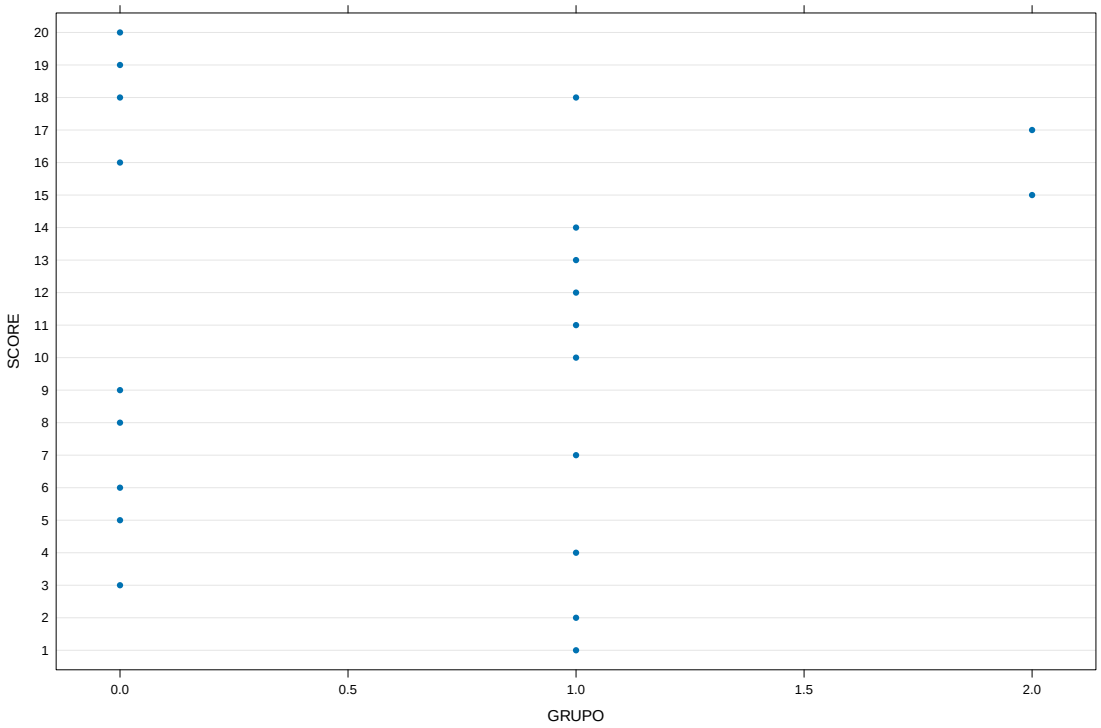


Figura 4.11: DotPlot Homogeneidade dos Grupos

Para confirmar ou refutar essa suspeita inicial, conduzimos os testes de Bartlett e Levene, ambos projetados para verificar formalmente a homogeneidade de variâncias. O teste de Bartlett, embora sensível à violação da normalidade, não forneceu evidências suficientes para rejeitar a hipótese nula de homogeneidade de variâncias (p-K-squared = 2.85, value = 0.24). Similarmente, o teste de Levene, mais robusto a desvios da normalidade, também não detectou diferenças significativas entre as variâncias dos grupos (F = 1.63, p-value = 0.22).

Considerando os resultados tanto da análise gráfica quanto dos testes estatísticos, concluímos que não há evidências empíricas suficientes para rejeitar a hipótese de homogeneidade de variâncias. Essa conclusão é crucial, pois valida a aplicação de testes estatísticos paramétricos que assumem essa premissa, como a ANOVA, para analisar as diferenças entre os grupos.

Ao comparar os diferentes grupos de visualização, a análise de variância (ANOVA) não revelou diferenças significativas na pontuação dos participantes (F = 1.5869, p-value = 0.25). Esses resultados sugerem que para as condições experimentais deste estudo, as animações utilizadas não apresentaram um efeito significativo sobre o desempenho dos participantes. É importante ressaltar que outros fatores, não considerados nesta pesquisa, podem ter influenciado os resultados.

Listing 4.9: Código R da aplicação da análise de variância (ANOVA) ao DataSet.

```
One-way analysis of means

data:  SCORE and GRUPO
F = 1.5869, num df = 2, denom df = 18, p-value = 0.2319
```

Visando compreender as dificuldades encontradas pelos participantes na aprendizagem das *FOS*, realizamos uma análise aprofundada dos dados coletados, com destaque para o desempenho no pós-teste (Seção 4.3). Embora a média geral do pós-teste tenha sido de 5.93, a análise por questão, apresentada na Figura 4.12, revelou-se a existência de tópicos que apresentaram maior dificuldade para os participantes. Para identificar os principais *misconceptions* e suas possíveis causas, realizamos uma análise individualizada das respostas dos participantes, buscando padrões de erros e dificuldades. Na Figura 4.12 apresentaremos uma análise detalhada de cada questão ao observar a amostra na totalidade. As alternativas com frame em preto denotam as alternativas corretas de cada questão. A Figura também apresenta o percentual de respondentes que assinalou cada uma das alternativas.

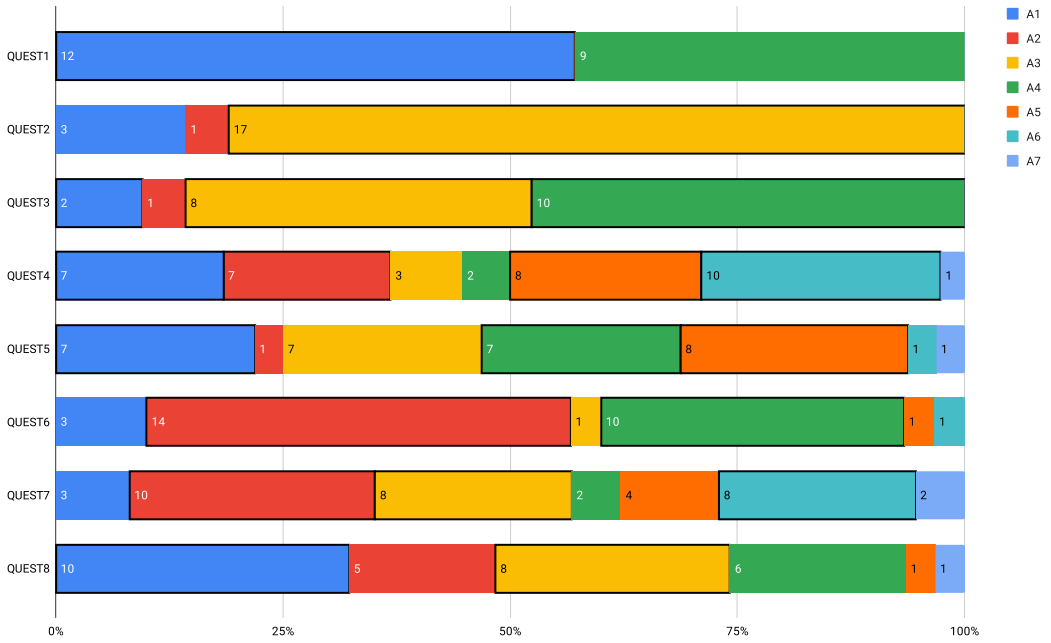


Figura 4.12: Gráfico Desempenho Geral

Na primeira questão, os participantes foram solicitados a interpretar a aplicação da função "filter" a uma lista de formas geométricas coloridas (quadrados e círculos). O objetivo era filtrar e retornar apenas os elementos circulares. Para representar os círculos a serem filtrados, utilizamos um círculo sem preenchimento como parâmetro da função. No entanto, 42% dos participantes interpretaram erroneamente que

a ausência de preenchimento no círculo do parâmetro significaria que nenhum elemento seria encontrado, indicando uma lista vazia como resposta (alternativa A4). Enquanto a maioria (57%) demonstrou compreender corretamente a aplicação da função, escolhendo a alternativa A1, que correspondia à lista correta contendo todos os círculos.

Na segunda questão, os participantes foram novamente desafiados a utilizar a função "filter". O objetivo, porém, era mais específico: filtrar e retornar apenas os círculos vermelhos de uma lista de formas geométricas coloridas. Para isso, um círculo vermelho foi utilizado como parâmetro da função. A análise das respostas revelou diferentes níveis de compreensão. A maioria dos participantes (81%) demonstrou domínio da função, selecionando a alternativa correta (A3). No entanto, 14% dos participantes interpretaram a função de forma mais ampla, indicando que todos os círculos seriam retornados, independentemente da cor (alternativa A1). Um pequeno grupo (4%) apresentou uma interpretação mais complexa, sugerindo que a função não apenas filtraria os círculos vermelhos, mas também alteraria a cor de todos os círculos para vermelho (alternativa A2).

A partir da terceira questão, investigamos a compreensão dos participantes sobre Funções de Ordem-Superior e composição de funções por meio de perguntas envolvendo rastreio de código, cada questão possuia mais de uma alternativa correta e o objetivo era que o participante assinalasse todas as alternativas corretas por questão.

A terceira questão explorava a compreensão dos participantes sobre o funcionamento da função "map" em Python. O código fornecido aplicava essa função a uma lista de nomes completos, invertendo a ordem de nome e sobrenome. A análise das respostas revelou um *misconception* comum: 47% dos participantes acreditavam que a função "map" modificava a lista original, ou seja, que após a aplicação da função, a lista nomes conteria os nomes já formatados no estilo "Sobrenome, Nome". Essa crença levou-os a marcar a alternativa A4. Alguns participantes (38%) demonstraram compreender corretamente o comportamento da função "map", indicando que a lista original permanece inalterada e a função retorna uma nova lista com os nomes formatados (alternativa A3). Um pequeno grupo (9,5%) também identificou a nova lista como resultado da função, mas de forma menos específica (alternativa A1). Por fim, 5% dos participantes apresentaram um equívoco mais fundamental, acreditando que a função "map" remove todos os elementos da lista original (alternativa A2).

A quarta questão, que introduz o conceito de composição de funções, apresentou um código cujo objetivo era filtrar e manipular uma lista de produtos com base em critérios específicos. A análise dos resultados revelou que:

- 26% dos participantes demonstraram compreender a combinação de múltiplos filtros, concordando com a alternativa A6 que afirma que as variáveis pf1 e pf2 possuem os mesmos valores, indicando que a ordem dos filtros não afeta o resultado neste caso específico.
- 21% demonstraram entendimento correto sobre a imutabilidade das funções de

ordem superior FOS, assinalando a alternativa A5, que corretamente afirma que a lista original de produtos permanece inalterada após a execução das operações.

- 36% evidenciaram compreensão da aplicação da composição de funções, dividindo-se entre aqueles que escolheram a alternativa A1 (18%), que corretamente indica que a saída da linha 24 é a média dos preços das camisas filtradas, e aqueles que escolheram a alternativa A2 (18%), que corretamente identifica uma das camisas filtradas na saída.

No entanto, observamos que:

- 8% dos participantes escolheram a alternativa A3, indicando uma compreensão parcial da filtragem, pois a saída apresentada nessa alternativa não inclui todas as camisas que atendem aos critérios.
- 5% demonstraram um equívoco fundamental, acreditando que as FOS modificam a lista original, como indicado na alternativa A4.
- 2% não conseguiram distinguir entre a combinação de múltiplos filtros e a combinação de filtragem com modificação dos dados, marcando a alternativa A7.

Para avaliar a compreensão dos participantes sobre a *FOS* "reduce", a quinta questão do pós-teste propôs o cálculo da média de uma lista de notas. Essa questão visava, especificamente, avaliar a compreensão do funcionamento do acumulador e da inicialização do "reduce". Com quatro alternativas corretas, a questão buscava identificar se os participantes haviam internalizado o conceito de redução de uma lista a um único valor.

Ao analisar as respostas, observamos que:

- 25% dos participantes demonstraram compreender a execução passo a passo do "reduce", identificando corretamente os valores dos parâmetros acc e nota na última iteração (alternativa A5).
- 42% demonstraram entendimento sobre a inicialização do acumulador, dividindo-se entre aqueles que escolheram a alternativa A1 (21%), que indica que o acumulador é inicializado com 0, e aqueles que escolheram a alternativa A4 (21%), que identifica o valor da primeira nota.

No entanto, observamos alguns equívocos mais fundamentais:

- 25% apresentaram um equívoco comum, acreditando que o acumulador é inicializado com o primeiro elemento da lista (alternativa A3).
- 3% dos participantes acreditaram que o "reduce" retornaria uma lista, em vez de um valor escalar (alternativa A6).
- 3% demonstraram um equívoco sobre a imutabilidade das listas em relação à aplicação das FOS, acreditando que o "reduce" alteraria a lista original (alternativa A7).

A questão 6, foi outra questão que abordava a aplicação da função “map”, ela apresentou duas alternativas corretas e permitiu avaliar a compreensão dos participantes sobre a transformação de listas e o conceito de imutabilidade. A análise das respostas revelou que:

- 46% dos participantes demonstraram entendimento claro sobre o objetivo da função “map” nesse contexto, identificando corretamente que a nova lista contém todos os números com o sinal invertido (alternativa A2).
- 33% demonstraram compreender o conceito de imutabilidade, indicando que a lista original não é alterada pela função “map” (alternativa A4).

No entanto, observamos algumas dificuldades:

- 10% dos participantes apresentaram um equívoco ao considerar que alguns números positivos não foram transformados (alternativa A1).
- 3% demonstraram um equívoco mais fundamental, acreditando que a lista original seria esvaziada (alternativa A3).

A sétima questão, com suas três alternativas corretas, visava avaliar a compreensão dos participantes sobre a composição de Funções de Ordem-SuperiorFOS “map” e “filter”, e como a ordem dessas operações afeta o resultado final.

A análise das respostas indicou que:

- 69% dos participantes demonstraram um bom entendimento da composição de “map” e “filter”, acertando pelo menos uma das três alternativas corretas (A2, A3 e A6). Esses participantes compreenderam que a ordem em que essas funções são aplicadas influencia diretamente o resultado final.
- 27% dos participantes, especificamente, acertaram a alternativa A2, demonstrando entender a ordem correta das operações na primeira parte do código.
- 21% dos participantes acertaram a alternativa A3, demonstrando entender a ordem correta das operações na segunda parte do código.
- 21% dos participantes acertaram a alternativa A6, demonstrando entender que a ordem das operações influencia o resultado final, mas sem especificar a ordem correta em cada caso.

Porém, as dificuldades observadas foram:

- 10% dos participantes inverteram a ordem das operações, acreditando que a ordem da execução das funções ocorre da função mais externa para a mais interna e não ao contrário (alternativa A5).
- 8% dos participantes apresentaram uma resposta incorreta para o conteúdo impresso na linha 12 (alternativa A1).
- 5% dos participantes apresentaram uma resposta incorreta para o conteúdo impresso na linha 16 (alternativa A4).

- 5% dos participantes acreditavam que o código continha um erro, impedindo sua execução (alternativa A7).

A oitava e última questão apresentou um código mais complexo, com o objetivo de calcular a média dos valores não negativos em uma lista até encontrar um valor sentinela específico. A análise das respostas revelou um bom nível de compreensão por parte dos participantes, mas também identificou alguns equívocos.

- 57% dos participantes acertaram pelo menos uma das duas alternativas corretas (A1 e A3), demonstrando um bom entendimento do funcionamento do código.
- 32% dos participantes acertaram a alternativa A1, indicando que compreenderam como a função listaAteOSentinela filtra os elementos da lista até encontrar o sentinela e como a função filter remove os valores negativos.
- 25% dos participantes acertaram a alternativa A3, indicando que compreenderam o cálculo da média dos valores não negativos e o resultado final.

No entanto, observamos algumas dificuldades:

- 16% dos participantes não incluíram o valor zero na lista sl, indicando uma possível confusão sobre o que é considerado um valor não negativo (alternativa A2).
- 19% dos participantes calcularam a média de forma incorreta, possivelmente dividindo pela quantidade total de elementos da lista original ao invés da quantidade de elementos não negativos na lista filtrada (alternativa A4).
- 3% dos participantes acreditavam que o resultado seria a própria lista sl (alternativa A5), demonstrando uma falta de compreensão sobre o cálculo da média.
- 3% dos participantes acreditavam que o código continha um erro (alternativa A7), indicando uma possível dificuldade em entender o código como um todo.

Para aprofundar nossa identificação dos principais *misconceptions*, realizamos uma análise detalhada do desempenho por grupo, complementando a análise geral. Ao comparar os resultados dos grupos 0 (sem animações), grupo 1 (array intermediário) e grupo 2 (pistas visuais), buscamos identificar possíveis padrões.

A figura 4.13 apresenta um panorama do desempenho de cada grupo por questão.

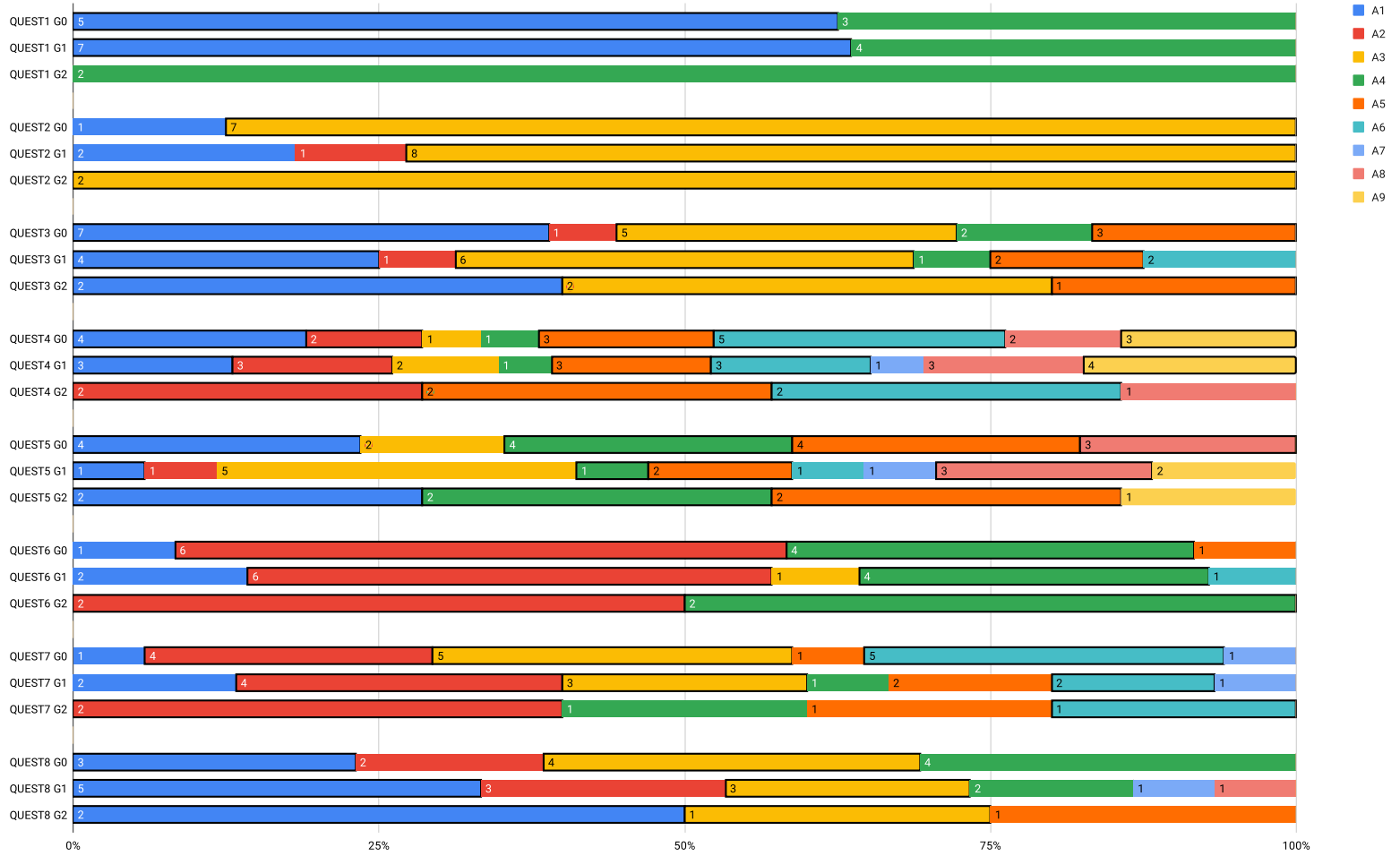


Figura 4.13: Gráfico Desempenho por Grupo

Ao analisar a primeira questão, observamos um contraste interessante nos resultados: os grupos 0 (controle) e 1 (array intermediário) apresentaram um desempenho superior ao grupo 2 (pistas visuais), com taxas de acerto de 62% e 63%, respectivamente. Em contrapartida, o grupo 2 não obteve nenhum acerto nessa questão. A análise dos *misconceptions* revela um padrão similar: os grupos 0 e 1 apresentaram taxas de equívocos semelhantes (37% e 36%, respectivamente), enquanto o grupo 2 apresentou uma taxa de 100

Ao passarmos para a segunda questão, observamos uma melhora no desempenho dos grupos. Enquanto na primeira questão o grupo 2 apresentou o pior resultado, na segunda questão ele obteve o melhor desempenho, com 100% de acertos. Os grupos 0 e 1 mantiveram um desempenho consistente, com taxas de acerto de 87% e 72%, respectivamente. Em relação ao *misconception* A1, os grupos 0 e 1 apresentaram taxas de escolha semelhantes, com 12% e 18%, respectivamente. Existindo ausência de dados para o grupo 2 nesse *misconception*.

Ao analisar a terceira questão, observamos uma maior variação nos resultados entre os grupos em comparação às questões anteriores. O grupo 0 (controle) obteve um desempenho ligeiramente superior na alternativa A1 (38%), enquanto o grupo 1 (array intermediário) apresentou a menor taxa de acerto nessa alternativa (25%). No entanto, o grupo 2 (pistas visuais) obteve um desempenho similar ao grupo 0. Quanto às alternativas A3 e A5, o grupo 1 apresentou os melhores resultados, com 37% e 12% de acertos, respectivamente. O grupo 2 manteve um desempenho consistente, com 40% de acertos em ambas as alternativas. O grupo 0 apresentou os menores índices de acerto nessas alternativas, com 27% e 16%, respectivamente. Analisando os *misconceptions* relacionados às alternativas A2 e A4, identificamos que os grupos 0 e 1 apresentaram taxas de escolha similares para a alternativa A2 (5% e 6%, respectivamente), indicando uma compreensão equivocada sobre o conteúdo da variável "nomes" após a execução da linha 7. O grupo 2, por sua vez, não apresentou respostas para essa alternativa. Em relação à alternativa A4, apenas o grupo 1 apresentou uma taxa de escolha de 12%, sugerindo que os demais grupos não demonstraram tal equívoco.

Na quarta questão, os grupos apresentaram diferentes padrões de respostas, em relação às alternativas corretas. Enquanto o grupo 2 demonstrou maior tendência a escolher a alternativa A2, os grupos 0 e 1 distribuíram suas escolhas entre as alternativas A1, A5, A6 E A9. Enquanto as respostas relacionadas às alternativas contendo possíveis *misconceptions*, a análise demonstrou que:

- Alternativa A4: uma pequena parcela dos grupos 0 (4%) e 1 (4%) acreditava que o array "produtos" estaria vazio após a execução da linha 25, indicando uma possível falta de compreensão sobre as operações realizadas no array.
- Alternativa A7: apenas 4% do grupo 1 escolheu essa alternativa incorreta, sugerindo uma confusão entre as variáveis "pf1" e "pf3" por parte de alguns participantes.

A análise da quinta questão revelou que 6% dos participantes do grupo 1 demonstraram um equívoco, acreditando erroneamente que o método reduce retornaria uma lista, em vez de um único valor (alternativa A6). Além disso, 6% dos mesmos participantes pensavam que o reduce modificaria a lista original (alternativa A7), demonstrando uma compreensão equivocada sobre a imutabilidade de listas em relação à aplicação das FOS. Esses resultados indicam que o grupo 1, em comparação com os demais, apresentou maiores dificuldades em dominar esses conceitos.

Ao analisar as respostas à sexta questão, observamos que tanto o grupo 0 quanto o grupo 1 apresentaram dificuldades em relação ao conceito de transformação de listas. Um percentual considerável de participantes de ambos os grupos (8% do grupo 0 e 14% do grupo 1) acreditou erroneamente que alguns números positivos não foram transformados conforme as regras estabelecidas (alternativa A1). Além disso, 7% dos participantes do grupo 1 demonstraram um equívoco mais fundamental, supondo que a lista original seria completamente esvaziada após a aplicação da transformação (alternativa A3).

A análise da sétima questão revelou que os participantes apresentaram diferentes *misconceptions* em relação ao funcionamento do código. As alternativas A1, A4, A5 e A7, que representam interpretações incorretas do código, foram escolhidas por uma parcela significativa dos participantes.

O grupo 1, por exemplo, apresentou uma maior tendência a escolher as alternativas A1 (13%), A4 (6%) e A5 (13%), indicando uma possível dificuldade em compreender a lógica de filtragem e transformação da lista "numeros". O grupo 0, por sua vez, apresentou um perfil de respostas mais distribuído, com 5% dos participantes escolhendo as alternativas A1, A5 e A7. A alternativa A4 foi escolhida por 6% dos participantes do grupo 1 e 20% do grupo 2, sugerindo que uma parcela considerável dos participantes não compreendeu a ordem em que as operações são realizadas no código.

A análise da última questão revelou que os participantes apresentaram diferentes *misconceptions* em relação ao funcionamento do código. As alternativas A2, A4, A5 e A7, que representam interpretações incorretas do código, foram escolhidas por uma parcela significativa dos participantes.

O grupo 0, por exemplo, demonstrou maior dificuldade em compreender a lógica de filtragem e transformação da lista "sl", com 15% dos participantes escolhendo a alternativa A2, que representa uma interpretação incorreta do conteúdo da lista após a execução da linha 21. Já o grupo 1 apresentou uma maior tendência a escolher a alternativa A4 (13%), indicando uma possível dificuldade em calcular a média corretamente. O grupo 2, por sua vez, se destacou por escolher a alternativa A5 (25%), sugerindo uma incompreensão da lógica de inserção de elementos na lista. A alternativa A7, que indica a presença de um erro no código, foi escolhida por 6% dos participantes do grupo 1, evidenciando a necessidade de aprimorar a identificação de erros sintáticos e lógicos.

Nossa análise detalhada revelou que alguns participantes apresentaram equívocos comuns, corroborando os achados da literatura (Krishnamurthi e Fisler, 2021). Entre esses equívocos, destacamos: (i) a crença de que o valor inicial do acumulador em "reduce" não influencia o cálculo da média e (ii) a suposição de que a ordem de aplicação das Funções de Ordem Superior FOS não altera o resultado final.

As hipóteses levantadas em nosso estudo, referentes a possíveis equívocos na compreensão de funções de ordem superior e que não estão descritos na literatura, também foram confirmadas pelos resultados da pesquisa. Observamos que uma parcela dos participantes apresentaram dificuldades em relação a:

- A natureza não destrutiva de "filter": a crença de que "filter" altera a lista original.
- A definição de critérios de filtragem: a compreensão da natureza booleana dos critérios de filtragem se mostrou desafiadora.
- A preservação da ordem: a manutenção da ordem dos elementos após a aplicação de "map" não foi totalmente compreendida.

Capítulo 5

Discussão dos Resultados

Ao adaptar uma ferramenta de visualização e simulação para o ensino de funções de ordem superior (FOS) este estudo se diferencia da literatura existente buscando investigar o impacto das visualizações e simulações de programas na compreensão de *FOS*, como 'map', 'filter' e 'reduce', e da técnica de composição de funções. A pesquisa envolveu 21 participantes, divididos aleatoriamente em três grupos: controle, 'array intermediário' e 'pistas visuais'.

Devido a limitações inerentes à realização da pesquisa em um ambiente online, com uma amostra relativamente pequena e a autonomia dos participantes no uso dos materiais, não foi possível estabelecer uma conclusão definitiva sobre a influência das visualizações e simulações na compreensão das *FOS*. Os dados coletados indicam a necessidade de estudos futuros com amostras maiores e controles mais rigorosos para aprofundar a investigação nesse tema. A partir dessas considerações buscamos responder nossas questões de pesquisa.

5.1 QP1: Quais as potenciais características de design que podem ser utilizadas na simulação e nas representações visuais de Funções de Ordem-Superior tais como o Filter, Map e Reduce?

As animações desenvolvidas neste estudo exploraram diferentes abordagens para visualização de *FOS*. Uma das abordagens, denominada "array intermediário", destaca-se pelo uso de instrumentos visuais que atuam como auxiliares cognitivos externos. Ao exibir explicitamente cada passo da operação, a simulação torna compreensível como os dados são transformados, diminuindo potencialmente a carga cognitiva. Além disso, a utilização de um array auxiliar permite decompor o problema em subproblemas menores, facilitando acompanhar o fluxo de dados. A representação

visual de um array intermediário concretiza o conceito abstrato de uma estrutura de dados intermediária.

A outra abordagem, "pistas visuais", prioriza a ênfase nos resultados. Ao minimizar os passos intermediários e utilizar cores para destacar os resultados, a simulação direciona a atenção do usuário para o efeito final das operações. As cores funcionam como uma abstração visual, representando os diferentes estados dos dados ao longo do processo. Essa abordagem proporciona agilidade, permitindo visualizar mais exemplos em menor tempo.

A escolha da abordagem mais adequada depende dos objetivos da simulação. A abordagem do array intermediário é útil para um aprendizado mais detalhado e passo a passo, enquanto a abordagem das pistas visuais é mais adequada para uma visualização rápida e geral dos resultados.

Em suma, as características de design apresentadas neste estudo evidenciam o potencial das animações para tornar a aprendizagem de *FOS* mais eficaz. No entanto, é importante ressaltar que nosso estudo não incluiu uma análise aprofundada do impacto dessas características na aprendizagem dos usuários, por meio de questionários ou entrevistas. Essa lacuna representa uma oportunidade para futuras pesquisas, que poderiam quantificar e qualificar os benefícios das diferentes abordagens de visualização.

5.2 QP2: A partir destas características de design, quais melhoram a compreensão de programas utilizando Funções de Ordem-Superior escritos em Python?

Os resultados da análise de variância (ANOVA) não revelaram diferenças significativas entre os grupos de visualização, sugerindo que, para as condições experimentais deste estudo, as animações utilizadas não influenciaram significativamente o desempenho dos participantes. No entanto, essa ausência de efeito diferencial não diminui o potencial das características de design exploradas, como transparência, decomposição do problema e abstração visual. Ao exibir de forma detalhada cada etapa da transformação dos dados, a transparência permite que o programador visualize a semântica interna das funções e compreenda como cada *FOS* contribui para o resultado final. A decomposição do problema em subproblemas menores torna o código mais fácil de entender e raciocinar, enquanto a abstração visual, por meio do uso de cores e formas, torna a compreensão mais intuitiva. Pesquisas futuras podem investigar como essas características interagem com outros fatores, como a experiência prévia dos programadores e a complexidade das tarefas, e como podem ser otimizadas para diferentes tipos de aprendizado.

Concluímos que os resultados deste estudo evidenciam a necessidade de pesquisas

mais aprofundadas sobre o impacto da visualização e simulação de programas na compreensão de *FOS*. Estudos futuros com amostras maiores e controle mais rigoroso das variáveis experimentais são cruciais para estabelecer relações de causa e efeito mais robustas entre as características das visualizações e o aprendizado dos participantes.

Capítulo 6

Conclusões

Neste estudo, investigamos a efetividade de visualizações e simulações na compreensão de funções de ordem superior *FOS* em Python. Para isso, desenvolvemos animações personalizadas utilizando a biblioteca JSVEE, que representam de forma visual o funcionamento de funções como "filter", "map" e "reduce" e a composição de funções, além de elaboramos um instrumento avaliativo. Ao aplicar esses materiais a um grupo de participantes e analisar os dados coletados, observamos uma receptividade positiva às visualizações. No entanto, o tamanho limitado da amostra impediu a obtenção de resultados conclusivos sobre o impacto direto dessas ferramentas na aprendizagem. Apesar disso, este trabalho contribui para o campo da educação em computação ao oferecer um ponto de partida para futuras pesquisas, além de fornecer materiais didáticos inovadores e identificar desafios e oportunidades no ensino de *FOS*.

6.1 Limitações

O principal desafio enfrentado neste estudo foi o tamanho limitado da amostra. Essa limitação impediu que realizássemos inferências estatísticas mais robustas e generalizáveis sobre a efetividade das visualizações e simulações na aprendizagem de funções de ordem-superior. Além disso, a heterogeneidade do grupo de participantes em relação ao conhecimento prévio de programação pode ter influenciado os resultados. Devido à aplicação do estudo ter sido em ambiente on-line, outra limitação é a falta de controle ao tempo de exposição às animações.

6.2 Trabalhos Futuros

Para superar as limitações encontradas e aprofundar o conhecimento sobre o tema, sugerimos os seguintes trabalhos futuros:

- Aumento do tamanho da amostra. Realizar o estudo com um número maior de participantes permitirá obter resultados mais confiáveis e generalizáveis. Uma

das opções aventadas é a tradução dos materiais para outros idiomas, o que nos permitirá expandir a população de interesse.

- Controle de variáveis. Controlar variáveis como o tempo de exposição às animações pode ajudar a identificar com maior precisão o impacto das visualizações na aprendizagem. Pesquisas anteriores (Sorva et al., 2012) mostram que a interação com a visualização é uma variável fundamental na obtenção de resultados significativos. Objetiva-se implementar abordagens como quizes embutidos na ferramenta de visualização que permitam aumentar o engajamento dos estudantes com a ferramenta e mensurar tal engajamento.

- Comparação com outras metodologias. Comparar a efetividade das animações com outras metodologias de ensino, como explicações tradicionais ou exemplos práticos, pode fornecer insights valiosos sobre as melhores práticas para o ensino de *FOS*.

- Análise de métricas mais específicas. Os dois designs sugeridos têm o potencial de reduzir a carga cognitiva dos estudantes ao compreender programas que utilizam *FOS*. Entretanto, este aspecto não foi mensurado. Planeja-se que no futuro mensure-se a carga cognitiva ao usar cada um dos protótipos (usando instrumentos como o de Morrison et al. (2014), por exemplo). Uma métrica importante também não foi avaliada neste trabalho: eficiência. É possível que os designs apresentados tenham a mesma eficácia do que o não-uso de visualizações, mas que estas reduzam significativamente o tempo necessário para compreender programas.

- Integração das animações em plataformas de ensino. Integrar as animações em plataformas de ensino online ou em ambientes de programação interativos pode facilitar o acesso e a utilização das ferramentas por um público mais amplo.

- Análises qualitativas. Ainda não é claro como os estudantes utilizam estas visualizações ou como interpretam os materiais e que tipos de modelos mentais os mesmos desenvolvem ao utilizarem *FOS*. Investigações qualitativas podem ajudar a elucidar estes modelos mentais e as escolhas que os estudantes realizaram ao responder o questionário pós-teste.

Referências

Austin, T., Horstmann, C., e Vu, H. (2018). Explicit short program practice in a programming languages course. *J. Comput. Sci. Coll.*, 33(4):114–122.

Backstrom, L. e Kleinberg, J. (2011). Network bucket testing. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, página 615–624, New York, NY, USA. Association for Computing Machinery.

Beckmann, J. F. (2010). Taming a beast of burden–on some issues with the conceptualisation and operationalisation of cognitive load. *Learning and instruction*, 20(3):250–264.

Berger-Wolf, T., Igic, B., Taylor, C., Sloan, R., e Poretsky, R. (2018). A biology-themed introductory cs course at a large, diverse public university. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, páginas 233–238.

Boulay, B. D. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research*, 2(1):57–73.

Brodley, C. E., Hescott, B. J., Biron, J., Ressing, A., Peiken, M., Maravetz, S., e Mislove, A. (2022). Broadening participation in computing via ubiquitous combined majors (cs+ x). In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1*, páginas 544–550.

Cao, L. (2017). Data science: A comprehensive overview. *ACM Comput. Surv.*, 50(3).

Cowan, N. (2001). The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and brain sciences*, 24(1):87–114.

de Koning, B. B., Tabbers, H. K., Rikers, R. M., e Paas, F. (2010). Attention guidance in learning from a complex animation: Seeing is understanding? *Learning and Instruction*, 20(2):111–122. Eye tracking as a tool to study and enhance multimedia learning.

Dorn, B. e Guzdial, M. (2006). Graphic designers who program as informal computer science learners. In *Proceedings of the Second International Workshop on*

Computing Education Research, ICER '06, página 127–134, New York, NY, USA. Association for Computing Machinery.

Dümmel, N., Westfechtel, B., e Ehmann, M. (2023). A multi-paradigm programming language for education. In *Proceedings of the 5th European Conference on Software Engineering Education*, ECSEE '23, página 236–245, New York, NY, USA. Association for Computing Machinery.

Duran, R., Rybicki, J.-M., Sorva, J., e Hellas, A. (2019). Exploring the value of student self-evaluation in introductory programming. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*, ICER '19, página 121–130, New York, NY, USA. Association for Computing Machinery.

Duran, R., Sorva, J., e Leite, S. (2018). Towards an analysis of program complexity from a cognitive perspective. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*, ICER '18, página 21–30, New York, NY, USA. Association for Computing Machinery.

Duran, R., Sorva, J., e Seppälä, O. (2021). Rules of program behavior. *ACM Trans. Comput. Educ.*, 21(4).

Duran, R., Zavgorodniaia, A., e Sorva, J. (2022). Cognitive load theory in computing education research: A review. *ACM Trans. Comput. Educ.*, 22(4).

Eliot, Sutinen, E., Tarhio, J., Lahtinen, S.-p., Tuovinen, A.-P., Rautama, E., e Meisalo, V. (1997). Eliot - an algorithm animation environment.

Force, A. D. S. T. (2021). *Computing Competencies for Undergraduate Data Science Curricula*. Association for Computing Machinery, New York, NY, USA.

Goldenberg, P., Mark, J., Harvey, B., Cuoco, A., e Fries, M. (2020). Design principles behind beauty and joy of computing. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, SIGCSE '20, página 220–226, New York, NY, USA. Association for Computing Machinery.

Guo, P. J. (2013). Online python tutor: Embeddable web-based program visualization for cs education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, página 579–584, New York, NY, USA. Association for Computing Machinery.

Guzdial, M. (2019). *Computing for Other Disciplines*, página 584–605. Cambridge Handbooks in Psychology. Cambridge University Press.

Guzdial, M. e Shreiner, T. (2021). Integrating computing through task-specific programming for disciplinary relevance: Considerations and examples. In *Computational Thinking in Education*, páginas 172–190. Routledge.

Haajanen, J., Pesonius, M., Sutinen, E., Tarhio, J., Terasvirta, T., e Vanninen, P. (1997). Animation of user algorithms on the web. páginas 356 – 363.

Izu, C., Schulte, C., Aggarwal, A., Cutts, Q., Duran, R., Gutica, M., Heinemann, B., Kraemer, E., Lonati, V., Mirolo, C., e Weeda, R. (2019). Fostering program comprehension in novice programmers - learning activities and learning trajectories. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*, ITiCSE-WGR '19, página 27–52, New York, NY, USA. Association for Computing Machinery.

Kaczmarczyk, L. C., Petrick, E. R., East, J. P., e Herman, G. L. (2010). Identifying student misconceptions of programming. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, SIGCSE '10, página 107–111, New York, NY, USA. Association for Computing Machinery.

Kather, P., Duran, R., e Vahrenhold, J. (2021). Through (tracking) their eyes: Abstraction and complexity in program comprehension. *ACM Trans. Comput. Educ.*, 22(2).

Ko, A. J., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M., Scaffidi, C., Lawrance, J., Lieberman, H., Myers, B., Rosson, M. B., Rothermel, G., Shaw, M., e Wiedenbeck, S. (2011). The state of the art in end-user software engineering. *ACM Comput. Surv.*, 43(3).

Kong, H.-K., Liu, Z., e Karahalios, K. (2017). Internal and external visual cue preferences for visualizations in presentations. *Comput. Graph. Forum*, 36(3):515–525.

Krishnamurthi, S. e Fisler, K. (2020). Data-centricity: a challenge and opportunity for computing education. *Communications of the ACM*, 63(8):24–26.

Krishnamurthi, S. e Fisler, K. (2021). Developing behavioral concepts of higher-order functions. In *Proceedings of the 17th ACM Conference on International Computing Education Research*, ICER 2021, página 306–318, New York, NY, USA. Association for Computing Machinery.

Lattu, M., Meisalo, V., e Tarhio, J. (2003). A visualisation tool as a demonstration aid. *Comput. Educ.*, 41(2):133–148.

Lattu, M., Tarhio, J., e Meisalo, V. (2000a). How a visualization tool can be used - evaluating a tool in a research and development project.

Lattu, M., Tarhio, J., e Meisalo, V. (2000b). How a visualization tool can be used - evaluating a tool in a research and development project.

Leidig, P. M. e Cassel, L. (2020). Acm taskforce efforts on computing competencies for undergraduate data science curricula. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, páginas 519–520.

Levy, R., Ben-Ari, M., e Uronen, P. (2003). The jeliot 2000 program animation system. *Computers & Education*, 40:1–15.

Lewis, C., Clancy, M., e Vahrenhold, J. (2019). *Student Knowledge and Misconceptions*, páginas 773–800. Cambridge University Press, United Kingdom. Publisher Copyright: © Cambridge University Press 2019.

Luxton-Reilly, A. (2016). Learning to program is easy. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '16, página 284–289, New York, NY, USA. Association for Computing Machinery.

Luxton-Reilly, A., Becker, B. A., Cao, Y., McDermott, R., Mirolo, C., Mühling, A., Petersen, A., Sanders, K., Simon, e Whalley, J. (2018). Developing assessments to determine mastery of programming fundamentals. In *Proceedings of the 2017 ITiCSE Conference on Working Group Reports*, ITiCSE-WGR '17, página 47–69, New York, NY, USA. Association for Computing Machinery.

Martins, M. R. e Duran, R. (2023). Ferramentas de visualização de programas na compreensão de funções de alta-ordem. In *Anais Estendidos do III Simpósio Brasileiro de Educação em Computação*, páginas 18–19. SBC.

Mayer, R. E. (1999). Multimedia aids to problem-solving transfer. *International Journal of Educational Research*, 31(7):611–623.

Mayer, R. E. (2002). Multimedia learning. In *Psychology of learning and motivation*, volume 41, páginas 85–139. Elsevier.

Moreno, A., Myller, N., Sutinen, E., e Ben-Ari, M. (2004). Visualizing programs with jeliot 3. páginas 373–376.

Morrison, B. B., Dorn, B., e Guzdial, M. (2014). Measuring cognitive load in introductory cs: adaptation of an instrument. In *Proceedings of the Tenth Annual Conference on International Computing Education Research*, ICER '14, página 131–138, New York, NY, USA. Association for Computing Machinery.

Muldner, K., Jennings, J., e Chiarelli, V. (2022). A review of worked examples in programming activities. *ACM Trans. Comput. Educ.*, 23(1).

Mutlu-Bayraktar, D., Cosgun, V., e Altan, T. (2019). Cognitive load in multimedia learning environments: A systematic review. *Computers Education*, 141:103618.

Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S., e Velázquez-Iturbide, J. A. (2002). Exploring the role of visualization and engagement in computer science education. *SIGCSE Bull.*, 35(2):131–152.

Nielsen, J. e Molich, R. (1990). Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, páginas 249–256.

Plaue, C. e Cook, L. R. (2015). Data journalism: Lessons learned while designing an interdisciplinary service course. In *Proceedings of the 46th ACM technical symposium on computer science education*, páginas 126–131.

Renkl, A. e Scheiter, K. (2017). Studying visual displays: How to instructionally support learning. *Educational Psychology Review*, 29.

Rivera, E. e Krishnamurthi, S. (2022). Structural versus pipeline composition of higher-order functions (experience report). *Proc. ACM Program. Lang.*, 6(ICFP).

Rivera, E., Krishnamurthi, S., e Goldstone, R. (2022). Plan composition using higher-order functions. In *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1*, ICER '22, página 84–104, New York, NY, USA. Association for Computing Machinery.

Scaffidi, C., Shaw, M., e Myers, B. (2005). An approach for categorizing end user programmers to guide software engineering research. In *Proceedings of the First Workshop on End-User Software Engineering*, WEUSE I, página 1–5, New York, NY, USA. Association for Computing Machinery.

Schroeder, N. e Cenkci, A. (2018). Spatial contiguity and spatial split-attention effects in multimedia learning environments: a meta-analysis. *Educational Psychology Review*, 30:1–23.

Shreiner, T. L. e Guzdial, M. (2022). The information won't just sink in: Helping teachers provide technology-assisted data literacy instruction in social studies. *British Journal of Educational Technology*, 53(5):1134–1158.

Sirkiä, T. (2013). A javascript library for visualizing program execution. In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research*, Koli Calling '13, página 189–190, New York, NY, USA. Association for Computing Machinery.

Sirkiä, T. et al. (2017). Creating, tailoring, and distributing program animations-supporting the production process of interactive learning content.

Sirkiä, T. e Sorva, J. (2015). Tailoring animations of example programs. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research*, páginas 147–151.

Sirkiä, T. (2016). Jsvee & kelmu: Creating and tailoring program animations for computing education. In *2016 IEEE Working Conference on Software Visualization (VISSOFT)*, páginas 36–45.

Sloan, R. H., Taylor, C., e Warner, R. (2017). Initial experiences with a cs+ law introduction to computer science (cs 1). In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, páginas 40–45.

Soloway, E. (1986). Learning to program = learning to construct mechanisms and explanations. *Commun. ACM*, 29(9):850–858.

Sorva, J. (2013). Notional machines and introductory programming education. *ACM Trans. Comput. Educ.*, 13(2).

Sorva, J. et al. (2012). *Visual program simulation in introductory programming education*. Aalto University.

Sorva, J., Karavirta, V., e Malmi, L. (2013). A review of generic program visualization systems for introductory programming education. *ACM Trans. Comput. Educ.*, 13(4).

Sorva, J. e Sirkiä, T. (2010). Uuhistle: a software tool for visual program simulation. In *Koli Calling*.

Stevens, J. P. (2013). *Intermediate statistics: A modern approach*. Routledge.

Surjono, H. D. e Muhtadi, A. (2017). The effects of various animation-based multimedia learning in e-learning course. In *Proceedings of the 9th International Conference on Education Technology and Computers*, ICETC '17, página 117–121, New York, NY, USA. Association for Computing Machinery.

Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cogn. Sci.*, 12:257–285.

Sweller, J. (1999). *Instructional design / John Sweller*. ACER Press.

Swidan, A., Hermans, F., e Smit, M. (2018). Programming misconceptions for school students. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*, ICER '18, página 151–159, New York, NY, USA. Association for Computing Machinery.

van Merrienboer, J. (1997). *Training complex cognitive skills: A four-component instructional design model for technical training*. Educational Technology Publications.

Wang, X., Lin, L., Han, M., e Spector, J. M. (2020). Impacts of cues on learning: Using eye-tracking technologies to examine the functions and designs of added cues in short instructional videos. *Computers in Human Behavior*, 107:106279.

Wiedenbeck, S. (2005). Facilitators and inhibitors of end-user development by teachers in a school. In *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*, páginas 215–222.

Apêndice A

APÊNDICES

A.1 Comitê de ética

Projeto CAAE: _ _, aprovado pelo Sistema CEP/CONEP, em _ _ de _ de 20_. Prezado(a) _, Você está sendo convidado(a) a participar da pesquisa intitulada: ferramentas de visualização de programas na compreensão de Funções de Ordem Superior (Higher-Order Functions). Este convite se deve ao fato de você ter/ser Estudante regularmente matriculado no curso de Engenharia da Computação da Universidade Estadual de Feira de Santana — UEFS ou do curso de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Mato Grosso do Sul — IFMS Campus Nova Andradina, o que seria muito útil para o andamento da pesquisa precisa ter concluído a disciplina de programação introdutória ou equivalente, possuindo assim conhecimentos básicos de programação tais como: variáveis, condicionais, loops, funções e listas, além de familiaridade com a linguagem Python e não possuir conhecimento de Funções de Ordem-Superior (Higher-Order Functions) O(A) pesquisador(a) responsável pela pesquisa é Marcos Rogério Martins, RG 07108280 80, discente do curso de Pós-graduação em Ciências da Computação da Universidade Estadual de Feira de Santana - UEFS. A pesquisa refere-se à aplicação de um estudo empírico visando investigar o impacto da visualização de programas na compreensão das Funções de Ordem-Superior, ferramentas amplamente utilizadas para tratamento e manipulação de dados, uma vez que ainda não existe nenhum sistema de visualização e simulação descrito atualmente na literatura com suporte explícito às Funções de Ordem-Superior. Após estudos realizados acerca das características de design de visualização de programas, protótipos foram desenvolvidos assim como adaptações de visualização para a execução de funções de ordem superior utilizando uma biblioteca JavaScript denominada (JSVEE), cujo objetivo é fornecer recursos necessários para criação de animações de programas. Essas animações integram um material instrucional abordando conceitos, fundamentos e exemplos de Funções de Ordem-Superior desenvolvidos pelo pesquisador, assim como um instrumento avaliativo que será aplicado ao final da pesquisa. Todos os materiais utilizados estão em

formato digital. Em sequência, incluir:

A pesquisa "Ferramentas de Visualização e Simulação de Programas na Compreensão de Funções de Ordem Superior" não apresenta riscos identificados para os participantes. A coleta de dados será realizada mediante um questionário online seguro e confidencial, e a identidade dos participantes será mantida em sigilo absoluto.

Riscos Potenciais da Pesquisa e Riscos Adicionais do Ambiente Online

Acessibilidade e Inclusão:

Risco: Dificuldade de acesso aos materiais e exame por parte de alguns participantes devido à falta de infraestrutura digital adequada (computador, internet) ou conhecimento técnico.

Medidas de Mitigação: A pesquisa será aplicada de forma assíncrona, em um ambiente remoto devidamente configurado para realização de todo o processo da pesquisa, onde o participante caso não disponha de equipamentos necessários, pode utilizar a infraestrutura das instituições UEFS e IFMS Campus Nova Andradina, buscando garantir o acesso ao material e durante a aplicação, o autor e o orientador se colocam à disposição para qualquer suporte. Utilizamos linguagem clara e acessível nos materiais e instruções. Foram realizados testes prévios de funcionamento dos materiais e do exame em diferentes dispositivos e navegadores.

Segurança e Privacidade de Dados:

Risco: Vulnerabilidade dos dados dos participantes a ataques cibernéticos, perda acidental ou uso indevido.

Medidas de Mitigação: Os dados serão armazenados em local seguro e protegido através da Plataforma JOTFORM, com medidas de criptografia e controle de acesso adequadas. Obter o consentimento livre e esclarecido dos participantes para coleta, armazenamento e uso de seus dados. A coleta de dados está limitada ao mínimo necessário para a pesquisa.

Validade e Confiabilidade dos Resultados:

Risco: Dificuldade em garantir a identidade dos participantes e evitar fraudes, como a participação de terceiros ou a realização múltipla do exame.

Medidas de Mitigação: A aplicação da pesquisa ocorrerá de forma totalmente remota, onde os participantes serão convidados durante as aulas da disciplina ministrada pelo Orientador no IFMS no referido curso de Tecnologia em Análise e Desenvolvimento de Sistemas e em aulas cedidas pelo Prof. Regente da disciplina de Jogos Digitais do curso de Engenharia da Computação na UEFS, a qual a supervisão ficará sob responsabilidade do autor da pesquisa.

Comunicação e Interação:

Risco: Dificuldade na comunicação com os participantes e na resolução de dúvidas, pela ausência de contato presencial.

Medidas de Mitigação: A aplicação da pesquisa ocorrerá de forma totalmente remota, onde os participantes serão convidados durante as aulas da disciplina ministrada pelo Orientador no IFMS no referido curso de Tecnologia em Análise e Desenvolvimento de Sistemas e em aulas cedidas pelo Prof. Regente da disciplina de Jogos Digitais do curso de Engenharia da Computação na UEFS e a supervisão da aplicação ficará sob responsabilidade do autor da pesquisa.

Bem-Estar dos Participantes:

Risco: Possíveis impactos psicológicos negativos nos participantes, como frustração, ansiedade ou estresse, devido à natureza da pesquisa ou à forma como ela é realizada.

Medidas de Mitigação: A aplicação da pesquisa ocorrerá de forma totalmente remota, onde os participantes serão convidados durante as aulas da disciplina ministrada pelo Orientador no IFMS no referido curso de Tecnologia em Análise e Desenvolvimento de Sistemas e em aulas cedidas pelo Prof. Regente da disciplina de Jogos Digitais do curso de Engenharia da Computação na UEFS e a supervisão da aplicação ficará sob responsabilidade do autor da pesquisa, de forma que facilita o monitoramento do bem-estar dos participantes durante a pesquisa para tomar medidas cabíveis em caso de necessidade.

Benefícios: • Aquisição de conhecimento: os participantes terão a oportunidade de aprender sobre um tema relevante e atual na área de Ciência da Computação: as Funções de Ordem Superior e seu papel na programação para análise de dados. • Aprimoramento de habilidades: A participação na pesquisa pode contribuir para o desenvolvimento de habilidades de raciocínio lógico, resolução de problemas, análise crítica e pensamento computacional. • Contribuição para o avanço do conhecimento: ao participar da pesquisa, os participantes estarão contribuindo diretamente para a geração de novos conhecimentos sobre como as pessoas aprendem Funções de Ordem Superior e como ferramentas de visualização podem facilitar esse processo.

Anonimato, Privacidade e Sigilo

Anonimato: A identidade dos participantes será mantida em sigilo absoluto durante toda a pesquisa. Os dados coletados não serão associados a nomes ou qualquer outra informação pessoal. Privacidade: A privacidade dos participantes será protegida durante toda a pesquisa. As informações coletadas serão armazenadas em local seguro e confidencial, acessível apenas aos pesquisadores responsáveis pelo projeto. A plataforma online utilizada para a coleta de dados possui diversas medidas de segurança para garantir a confidencialidade e integridade dos dados. Sigilo: O sigilo das informações coletadas será mantido durante toda a pesquisa. Os dados serão utilizados exclusivamente para fins da pesquisa e não serão compartilhados com terceiros sem a autorização expressa dos participantes. Os pesquisadores responsáveis pelo projeto se comprometem a manter a confidencialidade dos dados coletados e a utilizá-los apenas para fins de pesquisa.

Fases da Participação: Os participantes da pesquisa “Ferramentas de Visualização e Simulação de Programas na Compreensão de Funções de Ordem Superior” passa-

rão pelas seguintes fases: Leitura do Registro de Consentimento Livre e Esclarecido (TCLE): O participante lerá atentamente o TCLE, que apresenta todas as informações sobre a pesquisa, incluindo os objetivos, metodologia, riscos e benefícios, direitos dos participantes e forma de armazenamento dos dados. Assinatura do TCLE: Se o participante concordar com as condições descritas no TCLE, ele deverá assiná-lo eletronicamente. Preenchimento do Questionário: O participante responderá a um questionário online com duração estimada de cinco minutos. O questionário conterá perguntas sobre seus dados demográficos e conhecimentos de programação. Avaliação de Conhecimentos de Programação: Em seguida, o participante será direcionado para o material didático on-line elaborado pelo autor juntamente com o Orientador o qual aborda o ensino de Funções de Ordem-Superior utilizando uma ferramenta de Visualização de Programas para auxiliar o estudante na compreensão do assunto e ao final uma breve avaliação de conhecimentos de programação, visando avaliar seu nível de conhecimento adquirido sobre o tema da pesquisa. Agradecimento e Contato: Ao final da pesquisa, o participante receberá um agradecimento por sua participação e terá acesso aos contatos do pesquisador responsável para tirar dúvidas e obter mais informações sobre a pesquisa.

A pesquisa emprega o questionário on-line, portanto, recorre ao ambiente virtual para a coleta de dados. O instrumento de pesquisa utiliza a plataforma JOTFORM, da empresa JOTFORM. A plataforma e a empresa têm uma boa reputação, mas o(a) pesquisador(a) responsável não tem controle de como a empresa JOTFORM utiliza os dados que colhe dos participantes que respondem ao questionário. A política de privacidade da empresa está disponível em Dúvidas Política de Privacidade (jotform.com). Se você não se sentir seguro quanto às garantias da empresa JOTFORM quanto à proteção da sua privacidade, você deve cessar a sua participação, sem nenhum prejuízo. Caso concorde em participar, será considerado anuência quando responder ao questionário. Como medidas complementares decorrentes da utilização de ambiente virtual para coleta de dados, o(a) pesquisador(a) responsável assegura que: O TCLE depositado no Comitê de Ética tem a mesma formatação utilizada para visualização dos participantes da pesquisa. Não são utilizadas listas ou outro meio que permitam a identificação e/ou a visualização de seus dados pelos demais convidados ou por outras pessoas. O TCLE é apresentado anteriormente ao acesso às questões no caso de aplicação de questionários, mas contendo uma descrição do seu conteúdo tópicos que serão abordados que lhe permita avaliar e dar, ou não, o seu consentimento para participação na pesquisa. Você tem o direito de não responder qualquer questão, sem necessidade de explicação ou justificativa. Você tem o direito de se retirar da pesquisa, bem como retirar seu consentimento para a utilização de seus dados a qualquer momento, sem nenhum prejuízo. Para isso, basta declarar a retirada do consentimento através do martinsmrogerio@gmail.com. Nesse caso, o(a) pesquisador(a) responsável afiança que dará a ciência do seu interesse de retirar o consentimento de utilização de seus dados em resposta ao e-mail. O(A) pesquisador(a) responsável fará o download dos dados coletados para um dispositivo eletrônico pessoal assim que a coleta de dados for finalizada; e apagará

todo e qualquer registro do instrumento questionário e suas respostas na plataforma JOTFORM. Caso você aceite participar, é muito importante que guarde em seus arquivos uma cópia deste TCLE. Se for de seu interesse, o TCLE poderá ser obtido também na sua forma física, bastando uma simples solicitação através do endereço de e-mail: martinsmrogerio@gmail.com. Nesse caso, se perder a sua via física, poderá ainda solicitar uma cópia do documento ao(à) pesquisador(a) responsável. Qualquer dúvida ou necessidade – neste momento, no decorrer da sua participação ou após o encerramento, ou eventual interrupção da pesquisa – pode ser dirigida ao(à) pesquisador(a), por e-mail: martinsmrogerio@gmail.com, telefone (75)98869.4404, pessoalmente ou via postal para Centro Territorial de Educação Profissional do Recôncavo Jonival Lucas situado na Rua Firmino Rosalvo Lopes, n.º 71,Parque das Mangueiras, Sapeaçu- Bahia, CEP- 44530-000. Se preferir, ou em caso de reclamação ou denúncia de descumprimento de qualquer aspecto ético relacionado à pesquisa, você poderá recorrer ao Comitê de Ética em Pesquisa (CEP) da Universidade Estadual de Feira de Santana (UEFS), vinculado à CONEP (Comissão Nacional de Ética em Pesquisa), comissões colegiadas, que têm a atribuição legal de defender os direitos e interesses dos participantes de pesquisa em sua integridade e dignidade, e para contribuir com o desenvolvimento das pesquisas dentro dos padrões éticos. Você poderá acessar a página do CEP, disponível em:< INÍCIO | cepuefs > ou contatá-lo pelo endereço: Avenida Transnordestina,s/n, módulo I MA 17,Bairro Novo Horizonte, Feira de Santana–BA, CEP: 44036-900; e-mail: cep@uefs.br; telefone: +55 (75) 3161-8124. Se optar por participar da pesquisa, peço-lhe que escolha a opção Concordo ao final deste Termo de Consentimento Livre e Esclarecido.

Assinatura do pesquisador: Feira de Santana, de _ de 20

Se desejar receber os resultados da pesquisa e/ou o TCLE físico, assinale abaixo a sua opção e indique seu e-mail ou, se preferir, seu endereço postal, no espaço a seguir: _ _. [ ] RESULTADO DA PESQUISA [ ] TCLE IMPRESSO E RUBRICADO.

A.2 Material Didático

Errata: O Exemplo 1 sobre Composição de Funções correto está anexado na página 121.

Pesquisa: Funções de Ordem-Superior (Higher-Order Functions) em Python | Material de Apoio a Aprendizagem |

Autores: Prof. Dr Rodrigo Duran e Marcos Martins

Higher Order-Functions

"Funções de Ordem-Superior" (HoF - Higher-Order Functions) são descritas como funções que operam em outras funções, seja tomando-as como argumentos ou retornando-as como resultado. HoF também permitem composição de funções, ou seja, possuir pequenas funções que compõem funcionalidades que realizam tarefas mais extensas e complexas. As Funções de Ordem-Superior são componentes que estão cada vez mais presentes nas linguagens de programação modernas tais como Java, Javascript e R.

Funções de Ordem-Superior em Python

As funções de Ordem-Superior, são uma ferramenta poderosa em Python e podem melhorar significativamente como você constroi seus programas, pois tornam o código mais genérico, flexivel e reutilizável. Elas são parte do paradigma de programação funcional, de forma separada e organizada e apenas compor sub-resultados. As HoF são consideradas **"cidadãos de primeira classe"** e, portanto, podem ser usadas como qualquer outro tipo de valor.

```
# Higher-Order Functions
#Função que calcula o dobro de um valor
def  dobro (valor):
  return  2 * valor

#Função que calcula o quadrado de um valor
def quadrado (valor):
  return  valor * valor

#Função que calcula o negativo de um valor
def negativo (valor):
  return -1 * valor

#Função que aplica a função func ao valor
def mat_func (func, valor):
  return func(valor)

```

```
#Exemplos de Uso
print(mat_func(dobro,3))
6
print(mat_func(quadrado,3))
9
print(mat_func(negativo,3))
-3
```

Propriedades de funções de Ordem-Superior.

Algumas das propriedades mais importantes das funções de Ordem-Superior que são aplicáveis em Python são:

- Na função de Ordem-Superior, podemos armazenar uma função dentro de uma variável.
- Uma função pode atuar como uma instância de um tipo de objeto.
- Na função de Ordem-Superior, podemos retornar uma função como resultado de outra função.
- Em funções de Ordem-Superior, podemos passar uma função como parâmetro ou argumento dentro de outra função.
- Podemos armazenar funções de Ordem-Superior do Python em formato de estruturas de dados,como listas, tabelas de hash, etc.

Funções de Ordem-Superior aplicadas a listas vazias.

As funções de ordem superior (HoFs) em Python são ferramentas poderosas que podem ser usadas para processar listas de forma eficiente e concisa. No entanto, é importante considerar o comportamento dessas funções quando aplicadas a listas vazias. Devido à sua natureza recursiva, as HoFs geralmente definem um caso base para lidar com listas vazias, garantindo que a função funcione de maneira consistente, mesmo na ausência de elementos.

Map, Filter e Reduce: as HoF mais utilizadas

Obviamente, você pode construir qualquer tipo de função de Ordem-Superior e acoplá-las em suas funcionalidades. Entretanto, três funções ganharam enorme destaque dentro das abordagens funcionais, tanto que foram incluidas na maioria das linguagens de programação com suporte a comportamento funcional: **map, filter e reduce**. Vamos investigar cada uma delas em mais detalhes abaixo!

MAP

A HoF MAP possui dois parâmetros: a função de transformação, que é aplicada a cada elemento de uma lista, e a lista a ser mapeada. No exemplo abaixo, a função de mapeamento fornecida é aplicada a cada elemento da lista, um por vês, e o elemento transformado é inserido em uma nova lista, que será o resultado da HoF MAP.

Observe que a HoF MAP é imutável: a lista fornecida como parâmetro não é alterada, e uma nova lista é gerada como resultado do mapeamento. Perceba que a cardinalidade da lista resultado também não se altera: se N elementos estão presentes na lista argumento, a lista resultado também possuirá N elementos.Quando aplicada a uma **lista vazia**, a HoF MAP geralmente retorna uma lista vazia. Isso ocorre porque não há elementos na lista original para aplicar a função de transformação.

Exemplo 1 Map:

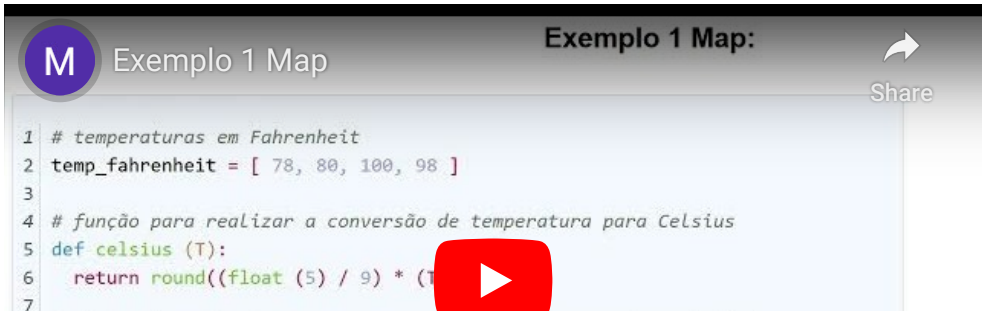
```
1  # temperaturas em Fahrenheit
2  temp_fahrenheit = [ 78, 80, 100, 98 ]
3
4  # função para realizar a conversão de temperatura para Celsius
5  def celsius (T):
6    return round((float (5) / 9) * (T-32),2)
7
8  #utilizando a função map para converter cada temperatura da lista
9  valores  =  list(map(celsius,temp_fahrenheit))
10
11 #Saída
12 print(valores)
13 [25.56, 26.67, 37.78, 36.67]
```

Quiz:

Qual é o primeiro argumento passado para a função map?

Selecione uma opção

Responder



Exemplo 2 Map:

```
# Listas contendo Siglas de alguns Estados Brasileiros
EstadosBR = [ 'am','ba','ms','rr','to' ]

#utilizando a função map para converter os itens da lista letras maiusculas
EstadosBR = list(map(str.upper,EstadosBR ))

#Saída
print(ESTADOSBR)
['AM','BA','MS','RR','TO']
```

Quiz:

Qual é o objetivo da função map no código?

Selecione uma opção

Responder

Exemplo 3 Map:

```
def obter_nome(email):
    return email.split("@")[0]

emails = [ "joaosilva@email.com", "mariagomes@email.com.br",
 "pedrodutra@email.com" ]

nomes  = list(map(obter_nome,emails))

print(nomes)

['joaosilva','mariagomes','pedrodutra']
```

Quiz:

Qual é o valor da variável nomes após a execução do código?

Selecione uma opção

Responder

FILTER

A HoF FILTER testa se cada elemento da lista argumento satisfaz a condição da função parâmetro. Para cada elemento cujo o resultado da função é **TRUE**, esse elemento é inserido na nova lista de resultado. Portanto, filter retorna um nova lista contendo apenas os elementos que satisfazem a função parâmetro.Quando a lista fornecida como argumento está **vazia**, a HoF FILTER retorna uma lista vazia.

A HoF FILTER também é imutável: a lista argumento permanece inalterada. Entretanto, a cardinalidade da lista resultado pode variar de 0 a N: se nenhum elemento satisfazer a função de condição, a lista resultado será vazia. Se todos os elementos atendem a função de condição, a lista resultado terá N elementos. Perceba que a função de condição deve avaliar cada elemento da lista, um a um, para um resultado **TRUE ou FALSE**. Logo, o resultado da função de condição só pode ser Booleana.

Exemplo 1 Filter:

```
# Criando a função de critério
def maior_que_zero (numeros):
    return numeros > 0

#Gerando uma lista com valores definidos aleatoriamente
valores = [ 10, 4, -1 , 3, 5, -9 ,-11 ]

#Aplicando filter na lista com valores
valores = list(filter(maior_que_zero,valores))

#Saída
print(valores)
[ 10, 4, 3, 5 ]
```

Quiz:

Qual é o tipo de dado da variável valores antes e depois da função filter?

Selecione uma opção Responder

Exemplo 2 Filter:

```
# Criando a função de critério
def retornaEstado (x):
   return  x  startswith ('M')

# Listas contendo Siglas de alguns Estados Brasileiros
ESTADOSBR  = [ 'MS','AM','BA','MT','TO','MG' ]

# utilizando a função filter para encontrar os estados que iniciam com a letra M
ESTADOSBR  = list(filter(retornaEstado,EstadosBR))

#Saída
print(ESTADOSBR)
['MS','MT','MG']
```

Quiz:

Qual o valor da variável ESTADOSBR após a execução da função filter?

Selecione uma opção Responder

Exemplo 3 Filter:

```
def eh_gmail(email):
    return email.split("@")[1] == "gmail.com"
```

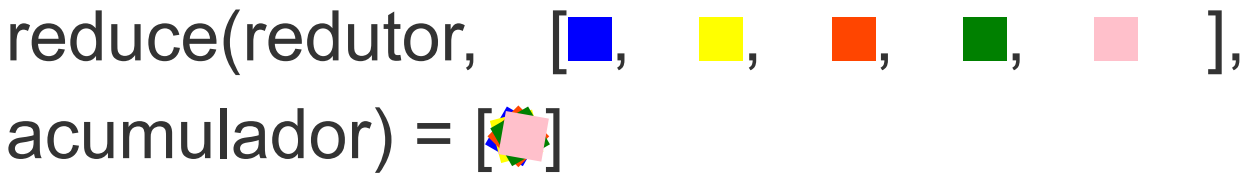
```
3
4 emails = [ "joaosilva@hotmail.com", "mariagomes@gmail.com",
   "pedrodutra@outlook.com", "marcos@gmail.com" ]
5
6 usuarios_gmail = filter(eh_gmail,emails)
7
8 print(list(usuarios_gmail))
9 ['mariagomes@gmail.com','marcos@gmail.com']
```

Quiz:

Qual a função do argumento eh_gmail dentro da chamada da função filter?

Selecione uma opção | Responder

REDUCE



A HoF REDUCE, diferentemente das funções MAP e FILTER, não é nativa no Python, e precisa ser importada de alguma biblioteca (functools). O REDUCE também é diferente pois não retorna (necessariamente) uma lista como resultado: ela pode retornar apenas um valor resultante da aplicação da função de redução a todos os elementos da lista.

Para tanto, REDUCE introduz um novo parâmetro que serve como acumulador para o resultado parcial a cada iteração com os elementos da lista.

A HoF REDUCE também é imutável: a lista argumento permanece inalterada. Para simplificar, podemos assumir que a cardinalidade do REDUCE é 1, pois a lista argumento será reduzida a um único elemento resultante. REDUCE combina elementos de uma lista usando uma função binária. Quando a lista fornecida como argumento está **vazia**, o comportamento do reduce depende do valor do acumulador. Se nenhum valor for fornecido para o acumulador, uma exceção IndexError é lançada. Caso um valor para o acumulador seja fornecido, este será o resultado da HoF REDUCE.

Exemplo 1 Reduce:

```
1 #Importando a função reduce
2 from functools import reduce
```

```
#Gerando uma lista com valores definidos aleatoriamente
numeros = [ 47, 11, 42, 13 ]

#criando a função soma
def soma (acc,elem):
    x = acc + elem
    return x

# Aplicando a função reduce na lista.
# 0 é o valor de inicialização do parâmetro acumulador.
valor = reduce(soma,numeros,0)

#Saída
print(valor)
113
```

Quiz:

Qual o valor da variável acc ao final da terceira iteração da função reduce?

Selecione uma opção | Responder

Exemplo 2 Reduce:

```
#Importando a função reduce
from functools import reduce
#criando função para verificar o comprimento
def comparar_comprimento (nome_a, nome_b):
    if len(nome_a) > len(nome_b):
        return nome_a
```

```
    else:
        return nome_b

#nomes de pessoas
nomes = [ "João", "Maria", "Pedro", "Rodrigo Duran","Maria Clara"]

resultado = reduce (comparar_comprimento,nomes,"")

#Saída
print(resultado)
Rodrigo Duran
```

Quiz:

Qual a ordem em que os nomes serão comparados pela função reduce?

Selecione uma opção

Responder

Exemplo 3 Reduce:

```
#Importando a função reduce
from functools import reduce
def contar_emails_dominio(acc,emails):
    if email.split("@")[1] == dominio:
        return acc + 1
    else:
        return acc

dominio = "gmail.com"

emails = [ "joaosilva@gmail.com", "mariagomes@yahoo.com", "marcos@gmail.com", "pedrodutra@hotmail.com"]

numero_emails_dominio = reduce(contar_emails_dominio,emails,dominio,0)

#Saída
print("Número de emails do dominio:",dominio)
print(numero_emails_dominio)
Número de emails do dominio gmail.com:
2
```

Quiz:

O que acontece na primeira iteração da função reduce?

Selecione uma opção | Responder

O PODER DA COMPOSIÇÃO

Encadeando Funções de Ordem-Superior e tornando seu código muito mais enxuto (e legível)!

Compondo Funções de Ordem-Superior

Uma grande vantagem oferecida por Funções de Ordem-Superior é a composição. Como **MAP** e **FILTER** retornam um iteravel, podemos imediatamente encadear uma outra função de Ordem-Superior a partir do resultado da anterior.
Dessa forma, podemos criar funções menores que gerenciam um (ou poucos) objetivos, compondo funções mais complexas utilizando várias funções menores. Esta técnica pode reduzir o número de bugs e fazer com que o código seja mais fácil de ser compreendido.
Assim, pode-se compor quantas funções achar necessário.

Exemplo 1 Composição de Funções:

```
produto = [ 'lapis', 'borracha', 'caderno', 'lapiseira', 'caderno' ]
# função que retorna se determinada string contem a substring la
def encontra (lista):
   for s  in lista:
      if "la" in lista:
   return s

# função que retorna se uma determinada string tem tamanho maior que 5
def tamanho (lista):
   return len(lista) > 5

resultado = list(filter(encontra,filter(tamanho,produto)))

#Saída
print(resultado)
['lapiseira']
```

Quiz:

Qual a ordem de execução das funções no código?

Selecione uma opção | Responder

Exemplo 2 Composição de Funções:

```
# temperaturas em Fahrenheit
temp_fahrenheit = [ 78, 80, 100, 98 ]
# função para realizar a conversão de temperatura para Celsius
def celsius (T):
   return round((float (5) / 9) * (T-32),2)

# função para encontrar altas temperaturas
def temperaturas_altas(x):
   return x > 30

resultado = list(filter(temperaturas_altas,map(celsius,temp_fahrenheit)))

#Saída
print(resultado)
[37.78, 36.67]
```

Quiz:

Qual o valor da variável resultado após a segunda iteração da função filter?

Selecione uma opção | Responder

Exemplo 3 Composição de Funções:

```
def obter_nome(email):
    return email.split("@")[0]

def eh_gmail(email):
    return email.split("@")[1] == "gmail.com"

emails = [ "joaosilva@gmail.com", "mariagomes@yahoo.com.br",
     "marcos@gmail.com","pedrodutra@hotmail.com" ]
```

```
8
9  primeiro_nome_gmail = list(map(obter_nome,filter(eh_gmail,emails)))
10
11 print(primeiro_nome_gmail)
12 ['joaosilva','marcos']
```

Exemplo 4 Composição de Funções:

```
1  def eh_hotmail(email):
2      return email.split("@")[1] == "hotmail.com"
3
4  def converter_para_outlook(email):
5      return email.replace("hotmail", "outlook")
6
7  emails = [ "joaosilva@gmail.com", "mariagomes@yahoo.com.br",
       "marcos@gmail.com","pedrodutra@hotmail.com" ]
8
9  emails_alterados = list(map(converter_para_outlook,filter(eh_hotmail,emails))
10
11 print(emails_alterados)
12 ['pedrodutra@outlook.com']
```

A.3 Questionário socioeconômico e Instrumento de Autoavaliação (SEI)

UEFS
UNIVERSIDADE ESTADUAL
DE FEIRA DE SANTANA

Consentimento

Estamos solicitando sua participação em um estudo que investiga as visualizações de programa no ensino de Funções de Ordem Superior .Sua participação é fundamental para o nosso estudo sobre o impacto das visualizações na compreensão de programas Python com Funções de Ordem Superior (Filter, Map, Reduce) entre estudantes de programação.
Para participar, basta preencher os formulários. Você pode pular qualquer pergunta que não se sinta confortável em responder.
Pedimos também o seu consentimento explícito para coletar e analisar os dados desta pesquisa para o estudo. Todos estes dados serão anonimizados e serão sempre armazenados de forma encriptada, pelo que não poderá ser identificado durante a análise dos dados. Quando for gerada uma publicação a partir destes dados, será anunciada através da coordenação de curso e o artigo será disponibilizado para você visualizar.
Se você tiver alguma dúvida sobre este estudo, pode entrar em contato com Marcos Rogério Martins discente do curso de Pós-Graduação em Ciência da Computação da Universidade Estadual de Feira de Santana (martinsmrogerio@gmail.com). Para obter uma opinião independente sobre a pesquisa e os direitos dos participantes da pesquisa, você pode entrar em contato com o comitê de ética da UEFS (cep@uefs.br).
Obs.: "Para uma experiência completa e eficiente, recomendamos a realização dessa pesquisa em um computador ou laptop. A tela maior e os recursos adicionais proporcionam uma navegação mais confortável e acesso a todas as ferramentas disponíveis"

Por favor, indique se consente que os seus dados sejam recolhidos e analisados: *

- Concordo que os dados que forneço sejam analisados como parte deste estudo
- Não concordo que os dados que forneço sejam analisados como parte deste estudo

Perfil Demográfico

Qual é o seu gênero, autoidentificado? Se você preferir não responder, deixe a questão em branco.

Qual a sua idade? Por favor, insira um número ou deixe a questão em branco se preferir não responder.

Como você se identifica étnico/racialmente? Você pode selecionar todas as categorias que se aplicam. Você pode deixar esta questão em branco se assim desejar. As categorias foram extraídas dos documentos oficiais do IBGE.

- Branco
- Pardo
- Preto
- Amarelo
- Indígena
- Outros

Qual instituição você estuda?

- UEFS - Universidade Estadual de Feira de Santana
- IFMS - Instituto Federal do Mato Grosso do Sul
- Outros

Qual curso você está matriculado atualmente?

- Engenharia da Computação
- Ciência da Computação
- Sistemas de Informação
- Licenciatura em Computação
- Engenharia de Software
- Tecnologia em Análise e Desenvolvimento de Sistemas
- Redes de Computadores
- Outros

Você é a primeira pessoa a frequentar o ensino superior (Universidade, Faculdade, etc.) na sua família?

- Sim
- Não
- Não sei ou não quero informar

O que melhor descreve o mais alto nível de escolaridade do seu tutor legal mais instruído (ou ex-tutor legal)?

- Não finalizou o ensino fundamental
- Ensino fundamental completo
- Não finalizou o ensino médio
- Ensino médio completo
- Ensino superior incompleto
- Ensino superior completo
- Pós-graduação (Mestrado, Doutorado)
- Não sei dizer ao certo

FLUÊNCIA EM LINGUAS

Em quais idiomas você se sente mais confortável para se comunicar? (Selecione uma ou mais respostas.)

- Inglês
- Espanhol
- Francês
- Alemão
- Italiano
- Outros

Você tem dificuldade em ler documentos formais escritos em Português (ou seja, devido a uma barreira linguística)?

- Sempre
- Muito Frequente
- Algumas vezes
- Raramente
- Nunca

EXPERIÊNCIA PRÉVIA EM PROGRAMAÇÃO

Indique o seu nível de concordância com as afirmações abaixo.

	Discordo Totalmente	Nem Concordo e nem discordo	Concordo Completamente
Aprendi programação antes de terminar o ensino médio em um ambiente informal (clube, autodidata, etc.).			
Fiz um curso de Computação durante o ensino médio (por exemplo, técnico em informática, bootcamp, etc.)			

Como você avaliaria sua proficiência em programação em comparação com seus colegas do curso?

- Muito Alta
- Acima da Média
- Na Média
- Abaixo da Média
- Muito Baixa

Sócio Econômico

As perguntas a seguir referem-se ao seu status socioeconômico. Este pode ser um tema delicado, por isso queremos reiterar que todos os dados que coletamos serão anonimizados. No entanto, como nas perguntas anteriores desta pesquisa, você também pode optar por não responder a uma ou mais dessas perguntas.

Qual é a renda média per capita anual da sua família (estimativa)

> R$ 300.000
R$ 150.000 a R$ 300.000
R$ 100.000 a R$ 150.000
R$ 50.000 a R$ 100.000
R$ 16.000 a R$50.000
< R$ 16.000
Não sei ao certo
Prefiro não responder

Quantas vezes você viajou ao exterior em férias ano passado?

Sua família possui uma máquina de lavar pratos?

Sim
Não
Prefiro não responder

Sua família possui uma máquina de lavar roupas?

Sim
Não
Prefiro não responder

Sua família possui uma máquina de secar roupas?

Sim
Não
Prefiro não responder

Você possui acesso à Internet em casa?

Sim
Não
Prefiro não responder

Sua família possui uma diarista, empregada doméstica?

Sim

Não

Prefiro não responder

Você recebe mesada ou algo similar de sua família?

Sim

Não

Prefiro não responder

Quantos banheiros você possui em sua casa?

Instrumento de Autoavaliação (SEI)

Os dados submetidos através deste formulário serão usados única e exclusivamente como diagnóstico do seu conhecimento prévio das habilidades de compreensão de programas escritos em Python. Leia com atenção a descrição de cada um dos níveis e assinale a qual dos níveis você identifica com a sua habilidade.

Legenda das opções de respostas:

A0 - Eu não tenho familiaridade com esse conceito.
A1 - Em geral, eu sei o que esse conceito significa.
A2 - Em geral, eu consigo reconhecer a sintaxe usada em Python para representar esse conceito.
B1 - Eu posso ler e realizar um tracing (teste de mesa) do código que usa esse conceito com poucos elementos. Eu consigo predizer o resultado de um código que usa esse conceito quando o código usa valores concretos e convenções de nomenclatura descritivas.
B2 - Eu posso ler e realizar um tracing (teste de mesa) do código que usa esse conceito com vários elementos diferentes. Eu consigo predizer o resultado de um código que usa esse conceito quando o código usa valores concretos e convenções de nomenclatura descritivas.
C1 - Na maioria das vezes, eu consigo reconhecer padrões usando esse conceito em diferentes tipos de código, mesmo quando as convenções de nomenclatura não são sempre descritivas. Eu sou capaz de compreender o propósito de comportamento de códigos que usam esse conceito usando uma grande quantidade de diferentes tipos de entradas.
C2 - Eu posso facilmente reconhecer padrões e explicar o comportamento de códigos que usam esse conceito, mesmo quando as convenções de nomenclatura não são descritivas. Eu posso generalizar o propósito e comportamento de código usando este conceito, mesmo quando este recebe um grande número de entradas diferentes. Eu sou capaz de resumir o propósito do código usando esse conceito em frases curtas.

VOCÊ DEVE ASSINALAR APENAS UM NÍVEL (COLUNA) A CADA UM DOS TÓPICOS (LINHAS)

Instrumento de Autoavaliação (SEI)

	A0	A1	A2	B1	B2	C1	C2
Varáveis							
Atribuição							
Entrada							

Saída

Operadores Aritméticos

Operadores Lógicos

Operadores Relacionais

Estruturas Condicionais

Loops ou Repetição

Funções ou Métodos

Classes ou Objetos

Funções de Ordem Superior (Filter, Map e Reduce)

Marque o campo abaixo quando concluir a leitura do material acima e clique em próximo quando estiver pronto para responder o questionário de avaliação.

*

Prosseguir

Marque o campo abaixo quando concluir a leitura do material acima e clique em próximo quando estiver pronto para responder o questionário de avaliação.

Prosseguir

Agradecimentos

Nesta pesquisa estamos realizando o sorteio dos livros "Ada Lovelace: A condessa curiosa" e "Alan Turing: Suas máquinas e seus segredos" de autoria da professora Silvia Bim. Caso deseje participar do sorteio, insira seu email abaixo e marque a opção "Dejeto participar do sorteio". Caso concorde em ser contatado posteriormente para uma entrevista sobre como você compreendeu os programas em Python utilizando funções de Ordem-Superior, marque a opção "Concordo em ser contatado posteriormente" e seu email.

Opções

- Desejo participar do sorteio dos livros
- Concordo em ser contatado posteriormente para uma entrevista

exemplo@exemplo.com

Gostaríamos de agradecer a todos que se dispuseram a participar de nossa pesquisa. Compreendemos que nem todos se sentem confortáveis em participar de pesquisas científicas, e respeitamos sua decisão. Agradecemos o seu tempo e consideração. Sua participação é importante para o avanço do conhecimento em Educação em Computação. Mesmo que você não tenha podido participar desta pesquisa específica, sua contribuição para outras pesquisas no passado ou no futuro é valiosa.

Atenciosamente,
Marcos Rogério Martins
Prof. Dr Rodrigo Duran
Pesquisa: Ferramentas de Visualização de Programas na compreensão de Funções de Ordem-Superior (Higher Order-Functions)
Universidade Estadual de Feira de Santana
Contato: martinsmrogerio@gmail.com

Quiz:

O que acontece na primeira iteração da função reduce?

Selecione uma opção | Responder

O PODER DA COMPOSIÇÃO

Encadeando Funções de Ordem-Superior e tornando seu código muito mais enxuto (e legível)!

Compondo Funções de Ordem-Superior

Uma grande vantagem oferecida por Funções de Ordem-Superior é a composição. Como **MAP** e **FILTER** retornam um iteravel, podemos imediatamente encadear uma outra função de Ordem-Superior a partir do resultado da anterior.
Dessa forma, podemos criar funções menores que gerenciam um (ou poucos) objetivos, compondo funções mais complexas utilizando várias funções menores. Esta técnica pode reduzir o número de bugs e fazer com que o código seja mais fácil de ser compreendido.
Assim, pode-se compor quantas funções achar necessário.

Exemplo 1 Composição de Funções:

```
produto = [ 'lapis', 'borracha', 'caderno', 'lapiseira', 'caderno' ]
# função que retorna se determinada string contem a substring la
def encontra (lista):
   return startswith("la")

# função que retorna se uma determinada string tem tamanho maior que 5
def tamanho (lista):
   return len(lista) > 5

resultado = list(filter(encontra,filter(tamanho,produto)))

#Saída
print(resultado)
['lapiseira']
```