
Universidade Federal de Pernambuco
Centro de Informática

Graduação em Ciência da Computação

# Willow: Uma Ferramenta com Conceitos de Programação Interativa para Auxiliar no Ensino de Introdução a Programação e Algoritmos e Estruturas de Dados

Pedro Henrique Sousa de Moraes

Trabalho de Graduação

Recife
05 de Julho de 2018

Universidade Federal de Pernambuco
Centro de Informática

Pedro Henrique Sousa de Moraes

# Willow: Uma Ferramenta com Conceitos de Programação Interativa para Auxiliar no Ensino de Introdução a Programação e Algoritmos e Estruturas de Dados

*Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.*

Orientador: *Prof. Leopoldo Motta Teixeira*

Recife
05 de Julho de 2018

# Agradecimentos

Agradeço aos meu pais por me ajudarem desde o início, fazendo o que estivesse ao seu alcance para me dar o melhor, principalmente quando o assunto era educação. Agradeço também meu orientador Leopoldo Teixeira pela parceria no ultimo ano e por ser compreensivo com os meus problemas. Agradeço também a todos os amigos que fiz dentro e fora do Centro de Informática, pois graças a eles passei por muitas experiencias. A todos, meus sinceros agradecimentos.

*Education is not preparation for life; education is life itself.*
—JOHN DEWEY

# Resumo

Um dos grande problemas enfrentados pelas instituições de ensino superior é a evasão dos alunos nos períodos iniciais. Este cenário é muito frequente nos cursos da área de Tecnologia da Informação. Os motivos que causam a evasão podem ser oriundos de problemas sociais, pessoais ou institucionais. Dentre estes motivos, há o problema da dificuldade com os conteúdos requeridos pelas disciplinas de programação, causados principalmente pela deficiência na formação básica do aluno. Para ajudar no ensino destas disciplinas, algumas ferramentas para auxiliar a programação através da abstração de elementos foram propostas. Apesar disto, estas ferramentas não são capazes de auxiliar na explicação de diversos conceitos de programação.

Este trabalho propõe o desenvolvimento de uma ferramenta para criação de visualizações personalizáveis de programas que podem ser usadas como exemplos interativos por instrutores para explicar conceitos de programação e algoritmos a alunos das disciplinas iniciais dos cursos de Tecnologia da Informação. Uma avaliação preliminar da ferramenta, feita com monitores da disciplina de Introdução a Programação, indica que a ferramenta é percebida como útil ao ensino por eles.

**Palavras-chave:** Ensino, Programação, Algoritmos, Visualização, Programação Interativa

# Abstract

One of the major problems faced by higher education institutions is the high dropout rate of students in the initial stages of their courses. This is a very frequent scenario in Information Technology courses. The reasons that cause such high rates might come from social, personal or institutional problems. Among these reasons, there is the problem of the difficulty with concepts required by the programming courses, usually caused by deficiencies in the basic training of the student. To assist in the teaching of these courses, some tools to aid programming through the abstraction of elements have been proposed. In spite of this, these tools are not able to help in the explanation of several programming concepts.

This work proposes the development of a tool for creating customizable program visualizations that can be used as interactive examples by instructors. This tool has the potential to better explain programming concepts and algorithms to students of the initial subjects of Information Technology courses. A preliminary evaluation of the tool, performed with teaching assistants for introductory programming courses indicates that the tool is perceived as useful for them.

**Keywords:** Teaching, Programming, Algorithms, Visualization, Interactive Programming, Live Programming

# Sumário

# Lista de Figuras

CAPÍTULO 1

# Introdução

A grande taxa de evasão no Ensino Superior é frequentemente analisada por diversos estudos, nacionais e internacionais [SFMHL07]. Infelizmente, este é um cenário frequente na área das Ciências Exatas, o que traz problemas tanto para os docentes como para a sociedade em geral.

Muitos estudantes que ingressam nas Universidades mas desistem de seus cursos nos primeiros semestres promovem grandes desperdícios econômicos, acadêmicos e sociais. Os desperdícios afligem tanto instituições públicas, pois não haverá retorno do investimento, como instituições privadas, onde há grande perda de receita.

## 1.1 Motivação

A evasão dos alunos é muito frequente nos períodos iniciais dos cursos de Tecnologia da Informação (Ciência da Computação, Engenharia da Computação e Sistemas de Informação) [dCMG13], principalmente nas disciplinas de Introdução a Programação e Algoritmos e Estruturas de Dados.

Diversos estudos exploram o tema da aprendizagem de Programação e Algoritmos, muitos deles apontam para diversos motivos relacionados a problemas sociais, pessoais, institucionais e de ensino que contribuem para a desistência dos alunos nos períodos iniciais [dSRBB12].

Uma das razões para a acentuada reprovação e desistência dos alunos é a dificuldade com os conteúdos e habilidades necessárias nas disciplinas de Introdução a Programação e Algoritmos e Estruturas de Dados [GHM08], onde o foco é a resolução de problemas multidisciplinares de forma a incentivar o raciocínio lógico. Isso é provocado pela deficiência na formação básica do aluno, principalmente na área das ciências exatas [PJF10], que influencia na capacidade de abstração das informações, na lógica-matemática e no entendimento da sintaxe e estruturas das linguagens de programação.

## 1.2 Objetivo

Esse trabalho tem como objetivo propor uma ferramenta que poderá ser usada por professores e alunos para auxiliar no ensino de Introdução a Programação e Algoritmos e Estruturas de Dados.

Abaixo estão listados os objetivos específicos deste trabalho.

- Realizar um estudo de ferramentas que já foram usadas para auxiliar o ensino de programação e algoritmos;

- Propor uma ferramenta com conceitos de programação interativa capaz de demonstrar conceitos de programação e algoritmos que não são possíveis com outras ferramentas;
- Realizar uma pesquisa qualitativa com monitores da disciplina de Introdução a Programação responsáveis por ministrar aulas de revisão;
- Analisar os resultados obtidos da pesquisa para uma avaliação preliminar sobre o uso da ferramenta nas turmas de Introdução a Programação.

## 1.3 Estrutura do Documento

Além desse capítulo, que apresenta o contexto, motivação e objetivos do trabalho, existem mais cinco capítulos, são eles:

- Capitulo 2: Analisa os trabalhos relacionados, que servem de inspiração para o desenvolvimento dessa ferramenta;
- Capítulo 3: Apresenta um conjunto de conceitos e ferramentas necessários para entender como a ferramenta funciona;
- Capítulo 4: Descreve com detalhe a forma que a ferramenta foi construída, o que é possível fazer com ela e seu fluxo de uso;
- Capítulo 5: Apresenta a metodologia utilizada e analisa o nível de aceitação do Willow como ferramenta de auxílio no ensino de Introdução a Programação;
- Capítulo 6: Expõe as considerações finais sobre o trabalho.

CAPÍTULO 2

# Trabalhos Relacionados

Este capítulo apresenta trabalhos usados na prática no ensino de Programação. Os trabalhos desenvolvidos usam diferentes estratégias como incentivo ao aluno. Estas estratégias se baseiam em usar visualizações para ajudar os alunos no desenvolvimento de programas, discutiremos sobre quais problemas as ferramentas tentam minimizar.

## 2.1 Programação com Blocos

Ferramentas deste tipo oferecem uma forma simplificada de programação através da manipulação de blocos com o mouse. Blocos representam estruturas de controle de fluxo comuns nas linguagens de programação como *"if-else"*, *"while"* e *"for"*. Espera-se que isso ajude a diminuir a carga cognitiva dos alunos, já que não precisam se preocupar com a sintaxe das linguagens de programação tradicionais.

### 2.1.1 IVProg

O IVProg é uma dessas ferramentas, apresentado na Figura 2.1, permite ao usuário desenvolver um programa manipulando a estrutura de blocos de declaração de variáveis, controle de fluxo e impressão para controlar a aplicação [dSRBB12]. A ferramenta foi usada em experimentos com alunos e como resultado houve um aumento médio de 15% na frequência e 0,9 pontos na média final dos alunos .

O IVProg foi usado com turmas de licenciatura em Matemática, os alunos usaram a linguagem de programação C e o IVProg. Comparado a outras turmas, o uso da ferramenta provocou um aumento médio de 15% na frequência dos alunos e e de 1,2 pontos em suas médias finais

### 2.1.2 PyGoGoBlocos

O PyGoGoBlocos [BSV+09], apresentado na Figura 2.2 tem o foco na área de robótica. Além dos blocos de controle de fluxo, ele possui blocos de disposição, que executam instruções vetoriais e blocos de controle de ações de um robô como ligar, frear e girar. Estes blocos são usados para manipular robores LEGO.

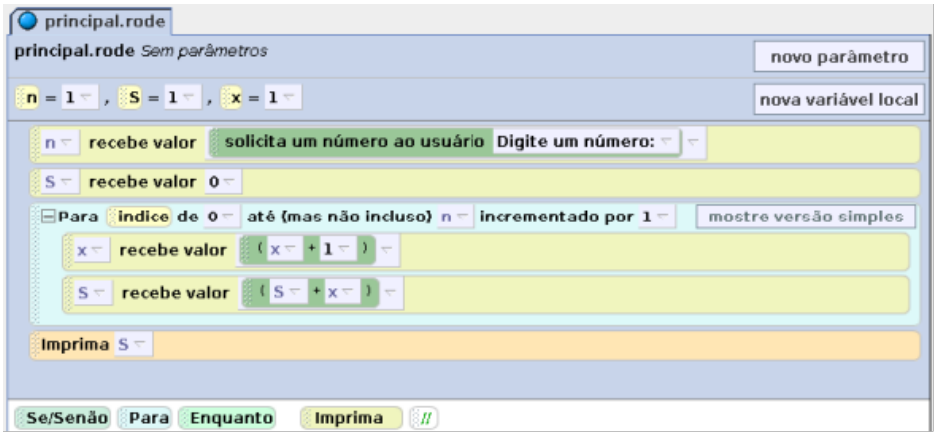


**Figura 2.1** Interface gráfica do IVProg

Este programa imprime a soma de todos os números do intervalo de zero a um número definido pelo usuário. Fonte: `http://www.matematica.br/ivprog/`.

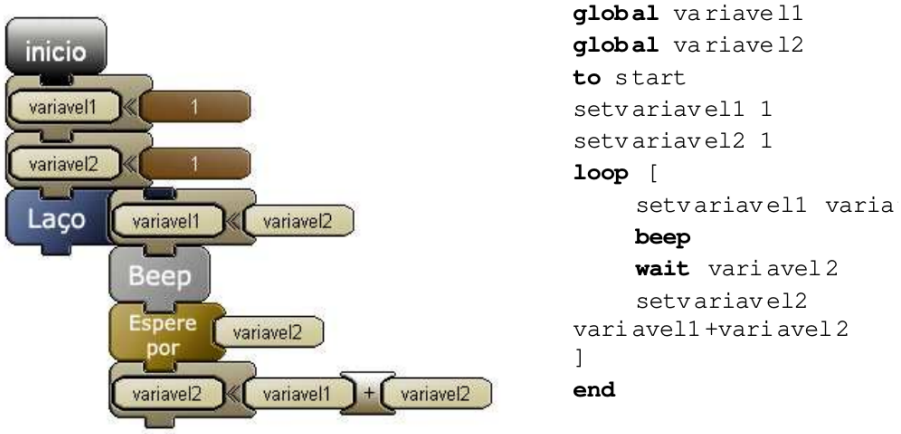


**Figura 2.2** Interface gráfica do PyGoGoBlocos

O programa implementado emite sons em intervalos da sequência de fibonacci. Fonte: Obtida no artigo [BSV+09].

### 2.1.3 Alice

Alice [Syk07] é uma ferramenta de programação com blocos que possui também um ambiente tridimensional, como pode ser visto na Figura 2.3, onde é possível arrastar e manipular objetos para criar animações. Alice foi desenvolvido com foco no ensino de programação, com o propósito de ensinar a teoria sem a complexidade das linguagens de produção como Java ou C++.

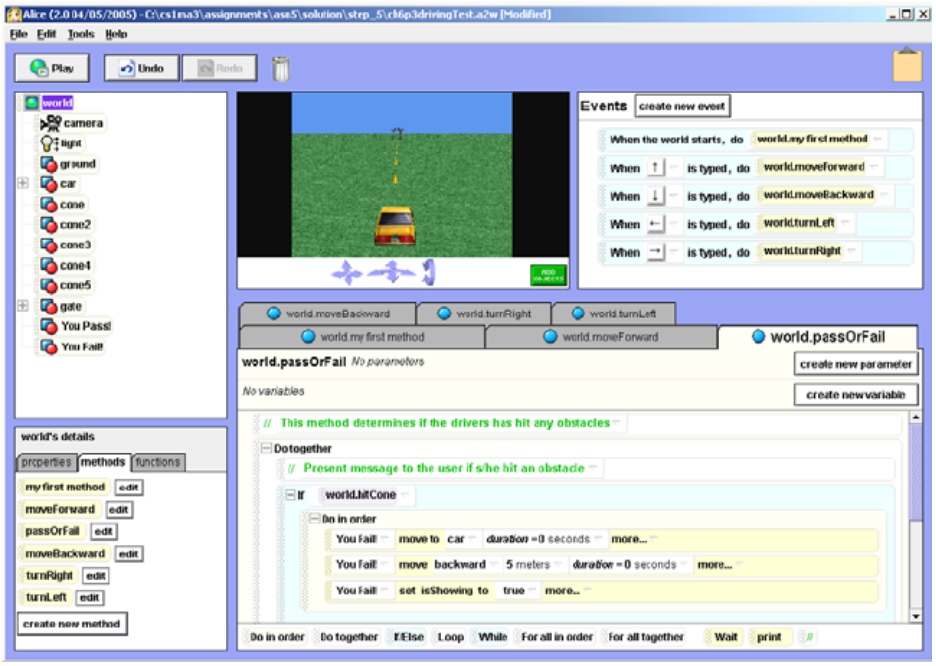


**Figura 2.3** Editor do Alice

Interface gráfica da ferramenta Alice. Fonte: Obtida no artigo [Syk07].

### 2.1.4 Scratch

O Scratch [MRR+10] é uma ferramenta que pode ser acessada online, há uma comunidade e muitos exemplos com os quais os usuários podem interagir. Scratch possui grande conjunto de blocos de instrução, é capaz de mover pequenas imagens com instruções de movimento semelhantes as do PyGoGoBlocos, reproduzir sons, responder a eventos de clique ou teclas, por estas funcionalidades, ele também é mais interativo, já que seus programas podem responder as ações do usuário enquanto executa.

Na Figura 2.4 podemos ver um programa em Scratch, ao ser executado, este programa faz com que a imagem do gato siga o cursor do mouse, e ao pressionar a tecla espaço, a imagem do gato se afasta do cursor.

### 2.1.5 Discussão

Essas ferramentas de programação visual cumprem bem o papel de abstrair a sintaxe das linguagens de programação, mas estão restritas a programas extremamente simples. Os blocos de controle de fluxo são limitados, isso não permite o uso de alguns conceitos comuns na programação, como funções.

Existem muitos outros conceitos da programação que não podem ser demonstrados com estas ferramentas. Tais conceitos estão relacionados ao que acontece durante a execução de um programa ou aos paradigmas das linguagens de programação, tais como recursão, vetores, referências e objetos.

Esses conceitos são fundamentais aos alunos das disciplinas de Programação e Algorit-

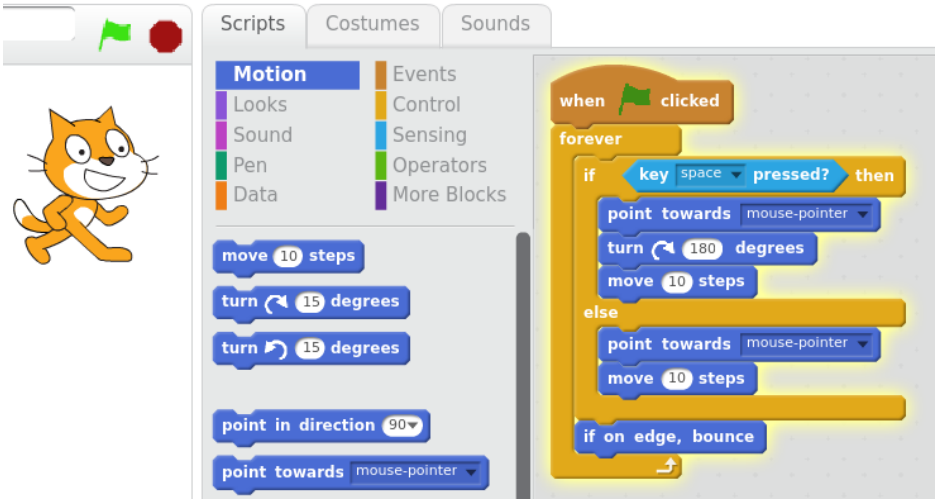


**Figura 2.4** Interface gráfica do Scratch

Ao executar o programa, a imagem do gato se aproxima ou afasta da posição do cursor. Fonte: Algoritmo implementado e capturado pelo autor em `https://scratch.mit.edu/`

mos, pois eles são necessários para expressar algoritmos mais complexos, mas requerem maior capacidade de abstração. Apesar da capacidade de abstração ser uma das maiores deficiências encontradas nos alunos [dSRBB12, PJF10, Guo13], a quantidade de estudos que propõem a solução desse problema é menor. Sem ferramentas de visualização, a demonstração dos conceitos deixa de ser interativa, o que pode dificultar o entendimento dos alunos.

## 2.2 Representação da Memória

Existem algumas opções de ferramentas que podem ser usadas para mostrar o que está acontecendo durante a execução dos programas. Depuradores são uma das opções mas não costumam ser bons para ensinar programação, pois seu objetivo é encontrar problemas em programas já escritos e definidos. Como muitas vezes a dificuldade dos alunos está justamente em expressar sua intenção por meio de um programa, isto torna esta estratégia difícil de ser usada, além de que suas representações dos dados não são suficientemente claras para exibir à iniciantes.

Instrutores tipicamente ilustram a execução dos programas através de apresentações de slides, o que requer um grande esforço para a preparação, ou podem ser desenhadas no quadro, mas isso é demorado e tedioso para os alunos e professores, além ser propicio à erros[OZS12].

### 2.2.1 Python Tutor

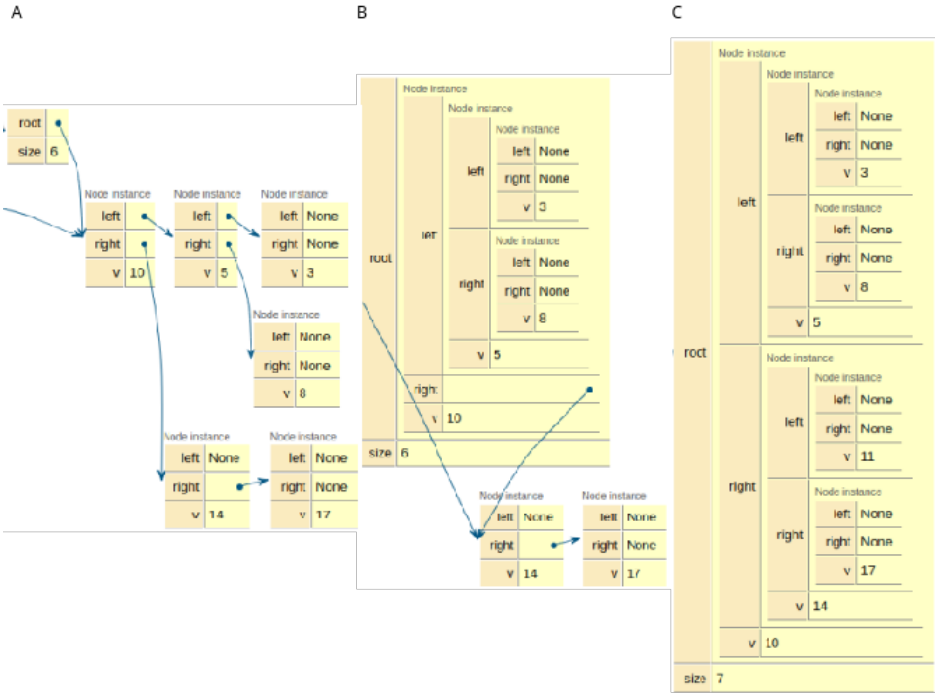


**Figura 2.5** Inserção em uma árvore de busca binária no Python Tutor

Em (A) uma árvore com seis elementos é representada, durante a inserção de um novo elemento em (B), a estrutura da árvore é aglutinada e todos os nós são mostrados dentro de seus parentes, exceto pelos nó que possui outra variável o referenciando, em (C) o nó recém criado e seu parente são também colocados juntos dos outros nós. Fonte: Implementado e capturado pelo autor em `http://www.pythontutor.com/`

Python Tutor é uma das poucas aplicações para visualização de programas em Python e em outras linguagens. Essa ferramenta é capaz de mostrar a estrutura dos objetos durante a execução de um programa, a ferramenta é usada por algumas universidades [Guo13] nos cursos de Introdução a Programação.

Por ser uma aplicação web, pode ser usada em qualquer lugar através de um navegador, sem a necessidade de instalação de softwares adicionais. Isso a torna acessível para alunos e professores, pois não há necessidade da configuração de um ambiente.

Apesar de ser capaz de mostrar qualquer objeto da aplicação em execução, as representações geradas pelo Python Tutor não podem ser manipuladas, ou seja, não é possível estilizar ou mover seus componentes, tornando-a inadequada para o uso na disciplina de Algoritmos e Estruturas de Dados.

Durante a execução dos algoritmos as visualizações geradas mudam drasticamente, o que dificulta o entendimento de algumas estruturas de dados mais complexas. pois a visualização de estruturas mais complexas como árvores como na Figura 2.5 e grafos é comprometida pelo posicionamento estático e aglutinação de elementos que a ferramenta usa.

### 2.2.2 Outras Ferramentas

Além do Python Tutor, existem ferramentas especializadas em análise da memória de um programa como em [CPP08] e [CR11], cujo objetivo é analisar algoritmos para verificar seu desempenho em relação ao uso de memória. Estas ferramentas não são adequadas para o ensino de algoritmos pois suas visualizações são especializadas para seu objetivo.

CAPÍTULO 3

# Conceitos Básicos

Este capítulo apresenta um conjunto de conceitos básicos, ferramentas e técnicas que foram usadas como inspiração, ou nescessárias para entender o desenvolvimento do Willow.

## 3.1 Programação Interativa

Programação envolve edição de código, em alguns momentos depuração e após isso a execução, a fim de obter o resultado de um programa. O conceito de Programação Interativa, também conhecido Programação *On-the-fly*, ou ainda *Live Programming*, se refere a habilidade de modificar um programa que está em execução.

É um modelo de programação onde edição, depuração e execução do código acontecem simultaneamente.Desta forma o desenvolvimento se torna mais fluido e o usuário tem resposta imediata do seu programa, o que torna possível diagnosticar problemas rapidamente, visualizar o resultado do código durante a edição e perceber o efeito das mudanças feitas. A resposta imediata promove um melhor entendimento do programa e das modificações feitas nele, por isso, o uso da Programação Interativa se torna interessante no que diz respeito ao ensino de programação e Algoritmos.

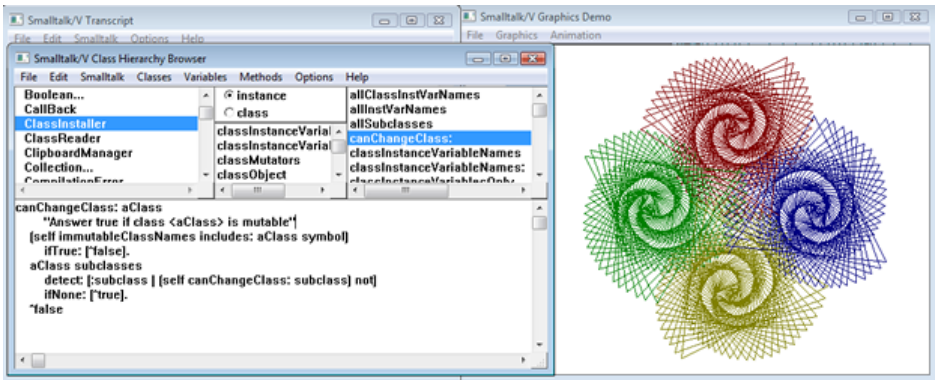


**Figura 3.1** Interface do ambiente de desenvolvimento de Smalltalk

O código do programa pode ser modificado enquanto executa, os resultados da modificação são refletidos imediatamente no resultado. Fonte:
`http://liveprogramming.github.io/liveblog/`

Alguns ambientes de desenvolvimento de linguagens de programação proveem a característica de Programação Interativa a uma linguagem. As linguagens mais conhecidas por esta

característica são Smalltalk 3.1 e Pharo. Estas linguagens possuem por padrão um ambiente de desenvolvimento construído para desenhar e manipular elementos gráficos, as modificações no código refletem diretamente nas visualizações geradas.

Outras linguagens possuem ambientes de Programação Interativa, são em sua maioria linguagens dinâmicas, pois possuem funcionalidades como avaliação dinâmica de código ou reflexão, que promovem suporte a esses tipos de ferramentas. Isso é aproveitado por desenvolvedores para ambientes de desenvolvimento ou outras ferramentas de resposta imediata. Alguns exemplos de linguagens populares que possuem características de Programação Interativa são:

- Javascript: Possibilita a execução de código mesmo após uma página HTML ser renderizada, o que possibilita a criação e manipulação de objetos da linguagem ou elementos da página;
- Python: Possui ferramentas e ambientes de desenvolvimento bem conhecidas para Programação Iterativa, as mais conhecidas são IPython (Interactive Python) e Jupyter Notebook, apresentado na Figura 3.2, com essas ferramentas é possível construir parcialmente um programa, já que é feita a execução parcial dos trechos de código providos pelo usuário;
- Lua: Possui o ZeroBrane Studio, que é um ambiente de desenvolvimento que se encarrega de executar o código à qualquer modificação;
- Scratch: Também possui a característica de Programação Interativa, seu editor online é capaz de executar as instruções nos blocos enquanto o usuário os manipula.

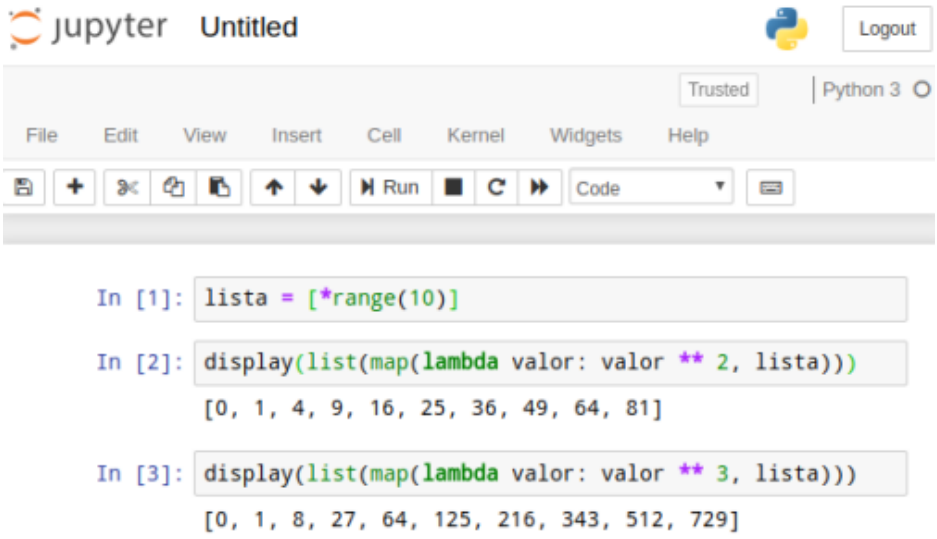


**Figura 3.2** Aplicação Python em execução no ambiente de desenvolvimento Jupyter

Cada célula pode ser executada independentemente das outras, e em qualquer ordem, caso o usuário deseje saber o conteúdo de qualquer variável ou objeto, ele pode criar uma nova célula e consultar o seu valor.

## 3.2 Depuração

Depuração é o processo para identificar e eliminar problemas em um programa, ou até mesmo em hardware. O processo de depuração de software geralmente depende das ferramentas que que a linguagem de programação a ser depurada possui, mas em muitos casos a depuração pode ser difícil e trabalhosa [Zel09]. Linguagens de alto nível são mais fáceis de depurar, pois possuem ferramentas que tornam a depuração mais fácil, além de prevenir erros comuns em linguagens de baixo nível como acessos ou escrita ilegais a memória. A depuração pode também ser usada mesmo que não hajam problemas com o programa, como por exemplo para entender o comportamento de algoritmos.

A maioria dos depuradores possuem conjuntos comuns de ações que ajudam o desenvolvedor a encontrar problemas. Por exemplo, podemos citar opções para indicar pontos de parada em um programa, executar linha a linha, inspecionar e modificar a memória e avaliar expressões no contexto atual da aplicação.

## 3.3 Heap e Stack

Grande parte das linguagens de programação adotam a o formato de divisão da memória dos programas em duas partes principais, são elas *heap* e *stack*. A *heap* é parte da memória onde a maioria objetos são criados, esses objetos são referenciados por variáveis em outros objetos ou na *stack*. A *stack* é a parte da memória que armazena a informação relacionada ao escopo das funções e variáveis declaradas. Além de *heap* e *stack*, é comum que existam outras partes da memória com diferentes responsabilidades, mas isso varia de acordo com a linguagem de programação.

## 3.4 Trace

*Trace* é a ação de analisar um ponto específico de um programa em execução e registrar todos os objetos no ponto de parada. Na maioria das linguagens, devido a forma com a qual a memória é estruturada em *heap* e *stack*, este processo funciona através da inspeção das variáveis na *stack*, pois são facilmente identificáveis. Através de uma busca em largura ou profundidade, cataloga todos os objetos alcançáveis, já que outros objetos existentes na *heap* não são mais acessíveis, não é necessário identifica-los.

A ação de *trace* está intimamente relacionada ao processo de Depuração, pois o *trace* é usado para encontrar as informações das variáveis e objetos que serão exibidas ao usuário.

## 3.5 JSON

JSON *(Javascript object notation)* é uma especificação usada para serializar dados em texto [JSO], é muito usada na comunicação entre sistemas web. O uso de JSON traz vantagens na comunicação entre sistemas pois possui uma sintaxe simples, que pode ser facilmente entendida

por humanos, mas também enxuta e altamente formatada, isso minimiza a quantidade de dados transferidos e torna fácil leitura e geração por programas. Por ser muito difundida, a maioria das linguagens de programação modernas já possuem bibliotecas para o processamento de JSON, isso promove ainda mais o uso dessa especificação.

## 3.6 REST

A Transferência de Estado Representacional (em inglês: Representational State Transfer, REST), é um conjunto de restrições e propriedades que moldam a forma de comunicação entre duas entidades [FT00]. REST é geralmente usado por serviços Web para definir uma interface de comunicação genérica e interoperável com outras aplicações. O uso de REST permite que aplicações desenvolvidas com tecnologias distintas, seja por usar arquiteturas, linguagens de programação ou hardware diferentes, possam se comunicar sem maiores problemas.

Os dados transferidos na comunicação entre duas aplicações que seguem o padrão REST devem sempre estar no formato JSON. REST pode ser usado com diferentes protocolos de comunicação, mas como é usualmente utilizado por serviços web, é muito comum que seja usado em conjunto com o protocolo HTTP (Hypertext transfer protocol), que é o protocolo de comunicação mais comum para estes sistemas.

## 3.7 Daemon

*Daemon* é um programa que é executado em um processo secundário, seja por um sistema operacional ou uma aplicação qualquer. Um usuário não pode se comunicar diretamente com um *daemon* através de periféricos de controle como mouse e teclado, mas outros processos podem trocar informações com *daemons*.

## 3.8 Pipe - Comunicação Inter-Processos

*Pipe* é uma das várias formas de comunicação entre processos, funcionando como um canal unidirecional que manda mensagens de um processo ao outro. Os dados escritos no *pipe* pelo processo produtor são armazenados pelo sistema operacional até que sejam lidos pelo processo consumidor. A comunicação bidirecional entre processos pode ser alcançada ao utilizar dois *pipes* para comunicação. *Pipes* também podem ser usadas para controlar a execução dos processos pois ao tentar consumir alguma informação de um *pipe* vazio, o processo pode interromper sua execução e esperar até que algo seja escrito.

CAPÍTULO 4

# Willow

Willow é uma ferramenta para geração de exemplos interativos, que pode ser usada tanto por professores como por alunos. Por ter sido implementada como uma aplicação Web, poderá ser acessada sem a necessidade de instalação de qualquer software.

Willow é capaz de modificar suas visualizações interativamente através de várias opções de personalização que podem ser inseridas e modificadas diretamente no código do programa executado, isso pode ser usado para tornar suas visualizações mais didáticas.

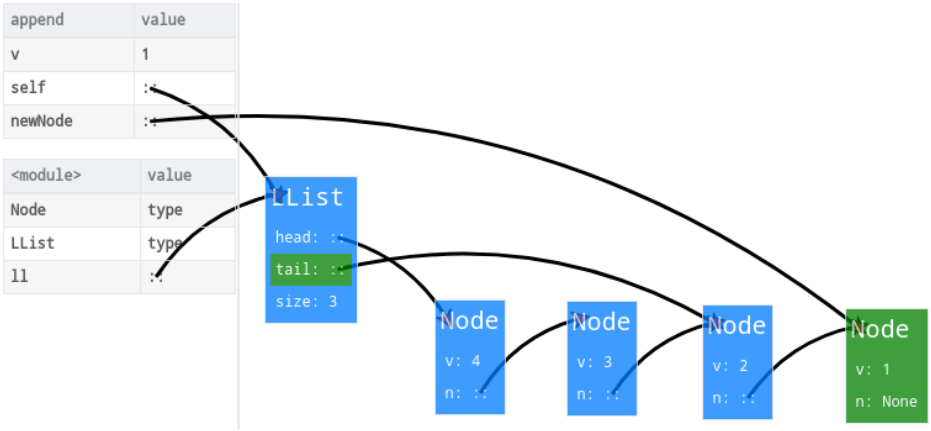


**Figura 4.1** Parte da interface do Willow durante execução de exemplo de lista encadeada

Apresentamos nesse capítulo o design da aplicação, os detalhes da a arquitetura do sistema, as funcionalidades da ferramenta e o seu fluxo de uso.

## 4.1 Arquitetura do Sistema

Essa sessão detalha a arquitetura do sistema e de seus módulos.

### 4.1.1 Visão Geral

Na Figura 4.2 temos uma visão geral dos do sistema, nela vemos os três módulos fundamentais do Willow.

O Tracer é responsável por analisar o código do usuário durante sua execução e gerar dados necessários para a visualização. Apesar de ser implementado em Python e para analisar código em Python, ele se comunica com o servidor através de JSON, isso torna possível o acoplamento

de Tracers para outras linguagens de programação. Efetivamente, permitindo a extensão de Willow para uso com outras linguagens.

O servidor atua como uma interface de comunicação entre o Tracer e o Cliente e possui outras poucas funcionalidades, foi implementado em Python e biblioteca Django.

O Cliente online é o módulo responsável por receber o código fonte e entradas do programa do usuário e gerar visualizações manipuláveis do programa em execução. Foi implementado com HTML, CSS, Javascript e a biblioteca React.

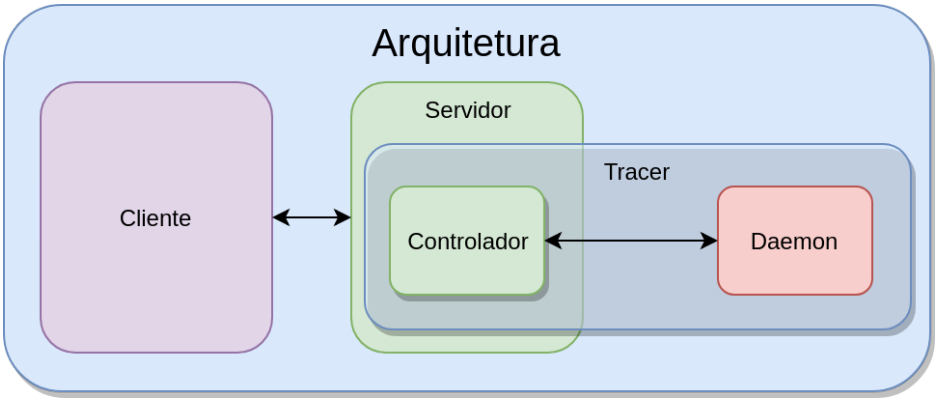


**Figura 4.2** Arquitetura do Willow

## 4.2 Tracer

Tracer é o módulo responsável pela execução e análise do código fornecido. O módulo é dividido em duas partes, o Daemon e o Controlador. A comunicação entre as partes é feita através de pipes. Embora existam Tracers para outras linguagens, e também para Python, este foi implementado especificamente para se comunicar com o servidor do Willow, que requer dados específicos obtidos do programa.

A Figura 4.3 mostra a estrutura básica deste módulo, que será detalhada a seguir.

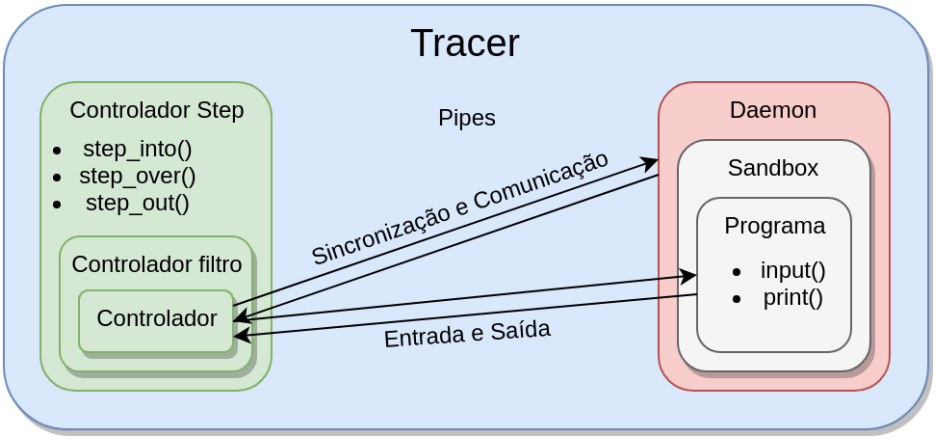


**Figura 4.3** Estrutura do Tracer

### 4.2.1 Comunicação

Daemons e Controladores se comunicam através de *pipes*, que são responsáveis por todas as trocas de informações entre essas entidades. São quatro *pipes* ao todo usadas na comunicação, que podem ser divididas em dois grupos, *pipes* de sincronização e comunicação e *pipes* de entrada e saída.

Os *pipes* de sincronização e comunicação conectam o Daemon e o Controlador e são usados para controlar e trocar de informações sobre a execução do programa.

- Pipe de ações: Usado pelo controlador para enviar ações que devem ser executadas no Daemon, como por exemplo uma requisição de encerramento do Daemon;
- Pipe de Respostas: Usada pelo Daemon para responder às ações enviadas pelo controlador, as mais comuns são respostas da análise da memória do programa.

Pipes de entrada e saída conectam o Controlador e a aplicação provida em execução dentro do Daemon e são usados para enviar e receber mensagens da aplicação.

- *Pipe* de *print*: Envia as mensagens impressas pelo programa do usuário em execução para o controlador, qualquer chamada da função `print` escreverá informações nesse *pipe*;
- *Pipe* de *input*: Usadas pelo controlador para enviar dados para o programa do usuário, ao chamar a função `input`, o programa do usuário lerá os dados desse *pipe*.

### 4.2.2 Daemon

Daemon é o componente principal do Tracer, responsável por configurar o ambiente e executar o programa recebido.

Como o Daemon é um processo separado do principal, ele atua como uma *sandbox*, pois não possui acesso aos outros módulos do Willow. O Daemon restringe a maioria dos módulos de Python, deixando disponíveis apenas `bisect`, `collections`, `copy`, `datetime`, `functools`, `hashlib`, `heapq`, `itertools`, `math`, `operator`, `random`, `re`, `string`, `time` e `typing`. Esses módulos não podem acessar recursos do sistema, o que é possível com alguns outros módulos como o `sys`. O Daemon também restringe o acesso às funções nativas `compile`, `exec` e `open`. Essas restrições impedem que o programa acesse elementos do servidor ou use recursos do sistema indevidamente.

As funções nativas `print` e `input` são redefinidas, agora elas leem e escrevem diretamente de *pipes*. Isso permite que o Daemon possa gerenciar o único meio de comunicação do programa, pois os módulos que possuem qualquer função de entrada ou saída não estão disponíveis, seja através das entradas e saídas padrão, de leitura de escrita em arquivos ou acesso as bibliotecas de sockets.

Após a configuração da *sandbox*, o Daemon executa o programa recebido instrução por instrução. Isso é possível devido a funções e módulos nativos de Python, especificamente, as funções nativas `compile` e `exec`, que servem, respectivamente, para compilar um programa Python em uma *string* e executar um programa compilado, e a função `set_trace` do módulo `sys` que pode ser usada para analisar o resultado de um programa após cada instrução.

O processo é sincronizado com o Controlador antes da execução do programa e a cada instrução executada. Isso impede que programas que consumem muitos recursos do sistema ou infinitos sejam executados de uma única vez, o que poderia causar uma demora muito grande na resposta ou travamento do servidor. Chamadas de métodos de outros módulos não são sincronizadas. Os resultados de cada instrução são ignorados nesses casos, diminuindo a quantidade de dados gerada pelo Daemon.

Após a primeira sincronização com o Controlador, o programa recebido é compilado. Se não houver nenhum problema, uma mensagem de sincronização é enviada ao Controlador e a primeira instrução do programa é executada. Caso haja algum erro durante o processo de compilação, como erros de sintaxe, estes são enviados para o Controlador e o Daemon é encerrado.

A cada instrução executada o Daemon espera uma mensagem do Controlador para analisar a memória do programa. Após a análise, o Daemon responde ao Controlador as informações do programa, executa a próxima instrução e então volta a esperar. O Daemon também pode receber uma mensagem de encerramento. Nesse caso, a execução do programa e do Daemon são encerradas. O Daemon também pode por si só encerrar a execução do programa se este for muito longo.

A analise de memória do Daemon é feita através da inspeção das variáveis globais, locais e da pilha de execução do programa. A análise identifica dados como a linha executada, seu número, as linhas onde cada novo escopo foi criado, a profundidade da pilha de execução, se a ação executada foi uma chamada ou retorno de uma função, se alguma exceção foi lançada, o que inclui o tipo da exceção, seu valor e o seu *traceback*, se é o fim da execução do programa e se o Daemon forçou o encerramento do programa.

Além desses dados, o Daemon também retorna uma representação das variáveis e objetos na *heap* do programa, essa representação e construída através de uma busca em profundidade desses objetos a partir das variáveis da pilha de execução. Variáveis que começam e terminam com "_" são ignoradas e variáveis que apenas começam com "_" não são recursivamente analisadas, mas são retornadas ao Controlador.

### 4.2.3 Controlador

O Controlador é a parte do Tracer responsável por criar e enviar instruções a um Daemon e receber e preprocessar os resultados obtidos.

O Controlador é organizado hierarquicamente em várias classes. Cada classe traz uma nova funcionalidade ou processa e aglutina os resultados recebidos do Daemon, como ignorar respostas que não possuem nenhum dado novo ou agrupar conjuntos de respostas para ações comuns de depuradores como *step over* ou *step out*.

Várias ações são oferecidas pelo controlador, são elas a inicialização de um Daemon com o programa que ele deverá executar, a captura de uma ou de um conjunto de respostas do Daemon, o envio de entradas para o programa que está em execução no Daemon e o seu encerramento. As ações do Controlador não uma interface para a comunicação entre os *pipes* que ligam o Controlador ao Daemon. Ao todo quatro tipos de mensagens são enviadas:

- Inicialização: É envidada para dar inicio a compilação e execução do código.

- Requisição de Resposta: Informa ao Daemon para analisar a memória após a ultima instrução executada, gerar os dados do estado do programa, retornar esses dados através do pipe e então executar a próxima instrução.
- Encerrar: Força a parada da execução do programa e encerra o Daemon.
- Enviar Entrada: Usada para enviar dados de entrada diretamente para o programa em execução dentro do Daemon.

A captura das respostas no controlador une as respostas provenientes do Daemon e da aplicação do usuário. Desta forma, quando são consultadas, as respostas do programa e do Daemon estão corretamente alinhadas. O Controlador também permite o pré-carregamento de entradas para o programa do usuário, dessa forma, as requisições de resposta não serão impedidas pela falta de dados de entrada para o programa em execução.

### 4.2.4 Estrutura das Respostas do Daemon

As respostas do Daemon para ações de *tracing* vindas do controlador precisam armazenar a estrutura dos objetos do programa. Tais estruturas podem ser extremamente simples, como um conjunto de referencias de variáveis para números ou *strings*, ou extremamente complexas, no caso de grafos direcionados com vários objetos. Como a comunicação entre o Controlador e o Daemon é feita através de JSON, que não possui forma de representar grafos, foi necessário criar uma.

A Figura 4.4 mostra a estrutura da primeira camada da resposta do Daemon, o formato dos dados da chave *locals*, que contem a informação dos objetos, exibe apenas os nomes das chaves internas, a estrutura destes elementos é mostrada na Figura 4.5.


```
{
    "event": "line",
    "line": 3,
    "text": "l = [False, 123, \"string\", {\"list\": 1234.5}]",
    "stack": [
        {
            "line": 0,
            "test": "<module>"
        }
    ],
    "depth": 1,
    "exception": null,
    "locals": { "classes": [], "stack": [], "objects": {} },
    "kill": null,
    "end": false
}
```


**Figura 4.4** Base da estrutura do JSON de resposta

Mostra as informações básicas obtidas no Daemon, como linha de código em execução, funções da pilha, se é o fim do programa e mais outros dados.

```
"locals": {
    "classes": [
        "LList",
        "Node"
    ],
    "stack": [
        {
            "name": "<module>",
            "variables": { "str": "some string", "ll": [ "108239740891" ] },
            "injects": { "_varHides": [ "str" ]
            }
        }
    ],
    "objects": {
        "108239740891": {
            "type": "list",
            "members": { "0": false, "1": 123, "2": "string", "3": [998570234897] },
            "injects": {}
        },
        "998570234897": {
            "type": "list",
            "members": { "\"list\"": 1234.5 },
            "injects": {}
        }
    }
}
```


**Figura 4.5** Estrutura do grafo de objetos representadas no JSON

A estrutura dos objetos no JSON é dividida em três elementos. O primeiro e mais simples é a lista de classes de objetos. Essa lista inclui apenas classes declaradas pelo programa e não classes de objetos nativos de Python, como listas, e dicionários.

O segundo elemento é a lista de escopos. Cada elemento dessa lista representa um dos escopos do programa no momento que o Daemon o analisou. Cada objeto possui o nome do escopo, que é normalmente o nome da função ou o nome do módulo raiz, o a lista de variáveis declaradas no escopo com os seus respectivos valores e a lista de variáveis injetadas, que serão explicadas mais adiante.

O terceiro elemento é o dicionário de objetos do programa. Para cada objeto é atribuído um identificador único, esse identificador é usado como valor nas variáveis na lista de escopos ou nas variáveis dentro de outros objetos. Pelo fato deste identificador ser um número, para que ele não seja entendido erroneamente como um número, mas sim como referência, ele é encapsulado dentro de uma lista. Os objetos, assim como escopos possuem três atributos, o seu tipo, que pode ser nativo de Python ou declarado pelo usuário, sua lista de variáveis declaradas e injetadas, que possuem o mesmo comportamento das variáveis nos escopos.

Com essa estrutura, sempre que mais de uma variável aponta para o mesmo objeto, é armazenada apenas a sua referência. Isso resolve problemas de duplicação e pendência cíclica do JSON.

## 4.3 Servidor

O Servidor é o módulo mais simples do Willow, ele é responsável por gerenciar Controladores para cada usuário. Isso é feito através da criação de sessões. A comunicação entre o Servidor

e os Clientes é feita através da interface REST e é extremamente simples. O servidor apenas oferece uma web API para acessar o conjunto de métodos disponíveis nos Controladores.

O Servidor também é responsável por fornecer exemplos de código para o cliente, esses exemplos implementam programas simples que podem ser usados e modificados pelos clientes.

## 4.4 Cliente

O Cliente é o módulo do Willow onde acontecem todas as interações dos usuários e onde respostas da aplicação são exibidas, sua estrutura pode ser vista na Figura 4.6. O Cliente é divido em vários submódulos responsáveis por diferentes partes do processamento de dados e da interação. Esses módulos estão divididos em dois grupos principais, os componentes e os gerenciadores de estado do Cliente (*reducers*).

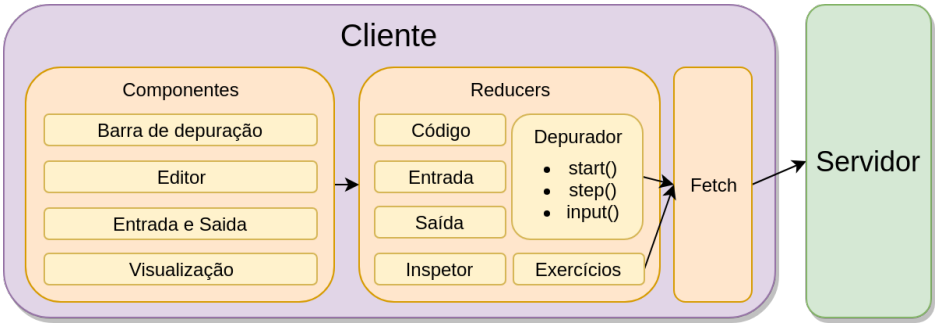


**Figura 4.6** Arquitetura do Cliente

### 4.4.1 Componentes

Os componentes do Cliente são as partes da estrutura que é visualizada na aplicação web. O componentes compartilham dados entre si através dos *reducers*, que armazenam o estado de maneira global e informam aos componentes quando esse estado muda. Eles são implementados com React e geram o HTML que será renderizado na tela.

Todos os componentes estão ligados ao processo de depuração, seja para prover dados, para exibir dados, ou os dois. Alguns dos componentes principais são:

- Barra de Depuração: É o elemento que controla a depuração no Cliente. Ele envia ações como iniciar uma depuração ou ir para o próximo passo para o Servidor, e recebe os resultados dos gerados pelo Daemon. Os resultados são então salvos e ficam disponíveis para outros componentes através dos *reducers*;

- Editor: Espaço onde o usuário escreve o seu código. Cada modificação no código é salva e pode ser usada por outros componentes. Durante o processo de depuração esse componente também marca a linha que será analisada e quando o programa está requer entrada de dados;

- Editor de Entrada: Esse editor armazena as informações de entrada para o programa do usuário. Durante o processo de depuração, ele também mostra que linhas já foram lidas;
- Editor de Saída: Esse componente não pode ser modificado pelo usuário, durante a depuração ele exibe todas as informações impressas pelo programa do usuário;
- Visualizador: Componente principal do Cliente, ele é responsável por exibir a pilha de execução e a *heap* do programa em execução, personalizar as visualizações de acordo com as variáveis injetadas e gerar as curvas que representam as referências entre os objetos.


**Figura 4.7** Interface completa do Willow sem nenhum programa em execução

### 4.4.2 Reducers

*Reducers* são os elementos que armazenam toda a informação do Cliente. No Willow especificamente, eles também são os únicos responsáveis por fazer requisições ao Servidor para enviar e receber os dados de Depuração. Esses dados são então armazenados e ficam disponíveis para que os componentes gerem as visualizações.

## 4.5 Visualizações e Personalizações

Willow pode ser usado para explicar tanto algoritmos que manipulem objetos na *head*, como listas encadeadas ou árvores, como para algoritmos que manipulam a pilha de execução como algoritmos recursivos.

### 4.5.1 Heap

A *Heap* é o componente da visualização que exibe o grafo de objetos e suas referências. Os objetos desenhados podem ser reposicionados manualmente, isso permite que o usuários os mova livremente para encontrar a melhor forma de visualizar o estado dos objetos.

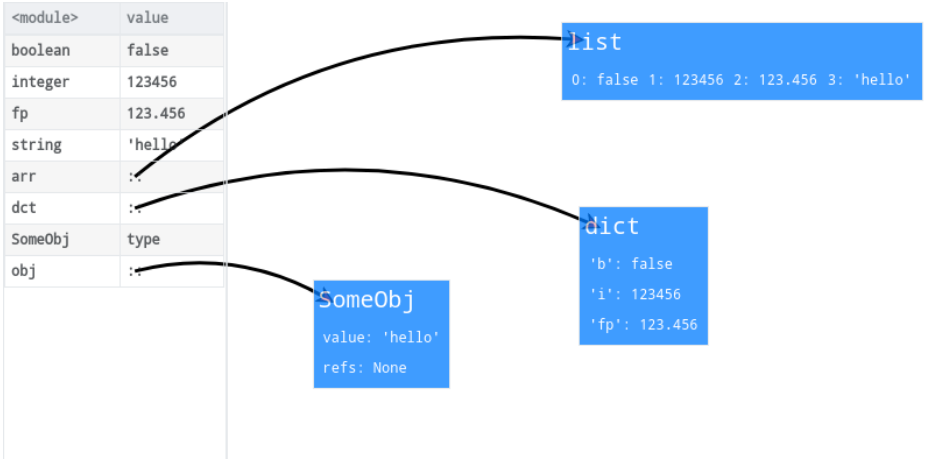


**Figura 4.8** Objetos desenhados pelo Willow sem nenhuma personalização

Os objetos podem ser movidos para qualquer posição na heap.

Os objetos da heap podem ser personalizados de várias maneiras, desde esconder suas variáveis a manipular completamente seus estilos e tamanho.

A manipulação é possível através das variáveis injetadas, essas variáveis são interceptadas pelo componente visualizador do Cliente, que as usa para modificar a forma com a qual um objeto é desenhado.

### _style

Ao declarar a variável `_style` como propriedade de qualquer objeto, essa variável poderá ser usada para manipular o estilo do elemento a ser desenhado. Qualquer propriedade CSS pode ser definida nela no formato de um dicionário, basta apenas uma pequena modificação na forma com a qual a propriedade é escrita. Deve ser usado o formato *camelCase* para os nomes das propriedades apenas, por exemplo a propriedade `background-color` deve ser escrita como `backgroundColor`.

A Figura 4.9 mostra a modificação do estilo da propriedade CSS acima em um dos objetos da Figura. 4.8. O objeto tem sua cor modificada para vermelho após a definição da sua variável de estilo.

### _varStyle e _varsStyle

As variáveis `_varStyle` e `_varsStyle` são usadas para modificar os estilos das variáveis de um objeto e não do objeto inteiro. Qualquer propriedade CSS também pode ser atribuída a esses objetos, para o `_varStyle` deve-se definir qual a variável que o estilo será aplicado, já em `_varsStyle` o estilo é aplicado em todas as variáveis.

A Figura 4.10 mostra a modificação do estilo das propriedades CSS em `_varStyle` e `_varsStyle` no objeto da Figura 4.9. O objeto tem uma de suas variáveis marcadas com a cor laranja e o estilo do texto passa a ser negrito e itálico.

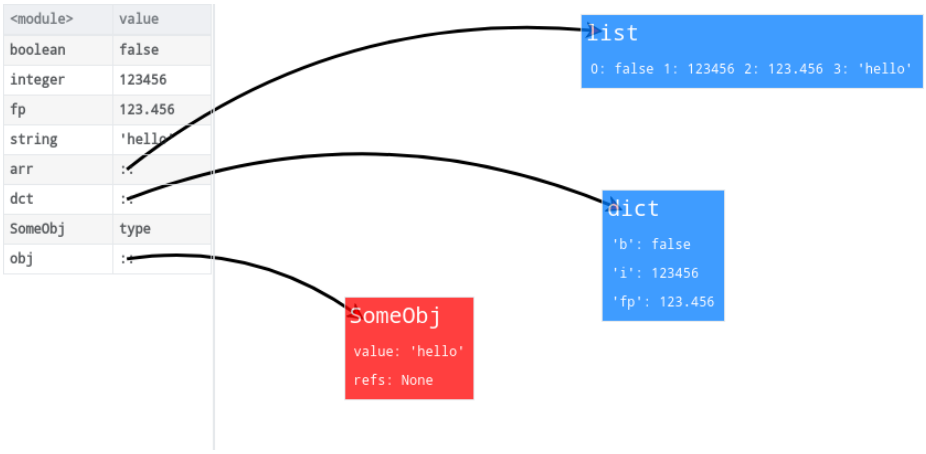


**Figura 4.9** Modificação do estilo de um objeto da *heap*

A propriedade `_style` do objeto foi setada com o valor `{'backgroundColor': 'red'}`.

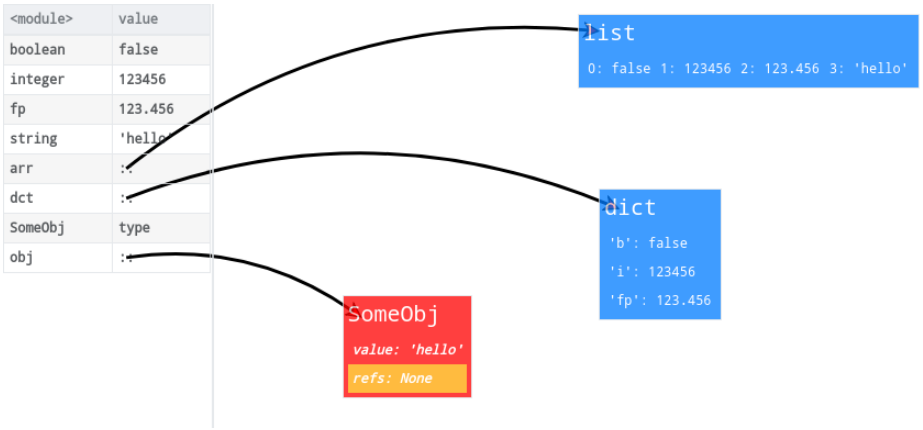


**Figura 4.10** Modificação do estilo das variáveis de um objeto na *heap*

A propriedade `_varsStyle` do objeto recebeu o valor `{'fontStyle': 'italic', 'fontWeight': 'bold'}` e a propriedade `_varStyle` recebeu o valor {'refs': {'backgroundColor': 'orange'}}.

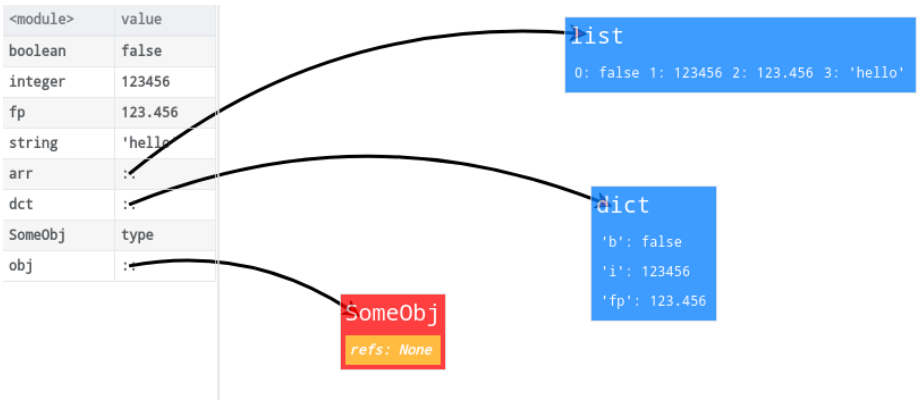


**Figura 4.11** Ocultação de uma variável de um objeto na `heap`

A propriedade `_varHides` do objeto recebeu o valor `['value']`.

### _varHides

A variável `_varHides` pode ser usada para esconder variáveis de um objeto, ao receber uma lista com nomes de variáveis, elas serão removidas da visualização enquanto seus nomes estiverem em `_varHides`.

Podemos ver na Figura 4.11 o efeito da remoção da variável `value` de um dos objetos da visualização na Figura 4.10.

### _varInside

A variável `_varInside` é usada para desenhar um objeto dentro de outro. Essa propriedade recebe uma lista com as variáveis que devem desenhar seus objetos dentro delas. Se a variável que desenhará o objeto dentro for a única que o referencie, então o objeto será desenhado apenas dentro da área da variável, caso contrário uma cópia será desenhada para as outras referências.

A Figura 4.12 mostra o efeito da operação de inserção de uma das variáveis do objeto na Figura 4.11. O objeto que a variável referencia é desenhado dentro do objeto que contem a variável.

## 4.5.2 Stack

Escopos da pilha de execução também podem ser modificados, mas com menos liberdade que os objetos da *heap*, apenas a propriedade `_style` pode ser usada. Seu efeito é aplicado no escopo em que foi declarada, dessa forma é possível definir diferentes estilos para diferentes escopos da pilha.

Podemos ver na Figura 4.13 a manipulação dos escopos da pilha.

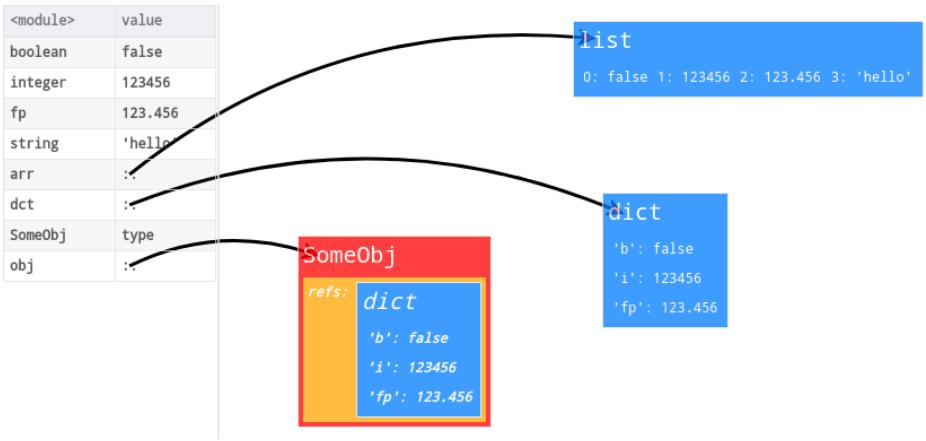


**Figura 4.12** Inserção de um objeto dentro de uma variável de outro objeto na heap

A propriedade `refs` recebeu a referência para o dicionário e então a propriedade `_varInside` do objeto recebeu o valor `['refs']`.

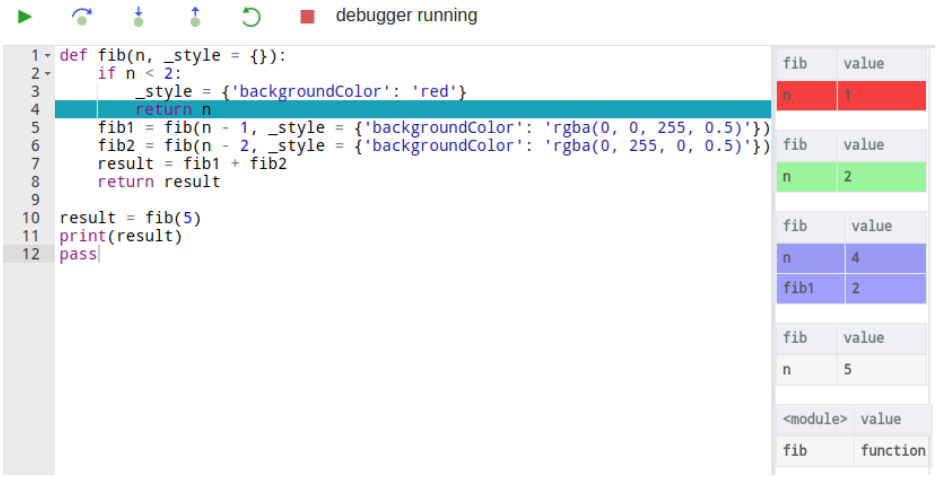


**Figura 4.13** Aplicação de estilo na pilha de escopos do Willow

## 4.6 Fluxo de Uso

O fluxo de uso do Willow é parecido com o da maioria dos depuradores, conforme detalhamos a seguir.

### 4.6.1 Inicialização

A primeira etapa é prover o código que será executado, o usuário pode usar um dos exemplos disponíveis se os achar adequados. Neste caso, basta ir no seletor de Exercícios no topo da interface e o código do exercício será carregado no editor. Para iniciar a execução do programa basta clicar no primeiro botão *(play)* da barra de depuração, o programa imediatamente co-

meçará a executar, a barra de depuração mostrará a mensagem *debugger running* e a primeira linha de código será marcada, assim como na Figura 4.13.

Caso haja algum erro de sintaxe no programa, ele será mostrado no Editor de Saída como na Figura 4.14, assim como em qualquer outro ambiente de desenvolvimento de Python.

```
Traceback (most recent call last):
  File "<script>", line 12
    pass
       ^
SyntaxError: invalid syntax
```

**Figura 4.14** Erro de sintaxe detectado pelo Willow

### 4.6.2 Passos da Execução do Código

Se a execução do programa iniciar corretamente, o usuário terá as opções de passos comuns em outros depuradores, são elas:

- *Step Over*: Avança para a próxima linha de código no mesmo escopo, se a próxima linha estiver em um escopo anterior então volta para o escopo anterior. Se houver uma linha que requer entrada mas não existe entrada disponível então avança para essa linha;
- *Step Into*: Avança para a próxima linha se não for uma chamada de função, caso seja uma chamada de função entra no novo escopo. Se a próxima linha estiver em um escopo anterior então volta para o escopo anterior;
- *Step Out*: Avança para a primeira linha de código onde o escopo é anterior ao atual, se houver uma linha que requer entrada mas não existe entrada disponível então avança para essa linha;
- *Restart*: Reinicia a execução da aplicação;
- *Stop*: Para a execução do código.

A Figura 4.15 mostra as ações disponíveis ao usuário durante a depuração.


**Figura 4.15** Barra de depuração do Willow durante a execução de programa

Durante a execução do programa, o usuário pode a qualquer momento mover os objetos do Visualizador.

### 4.6.3 Entradas e Saídas do Programa

Para programas que requerem entrada do usuário, se a aplicação chegar em algum ponto onde não há entrada disponível, será exibida uma barra amarela no editor como na Figura 4.16 indicando que a aplicação está parada naquele ponto e não prosseguirá até que a entrada seja fornecida.

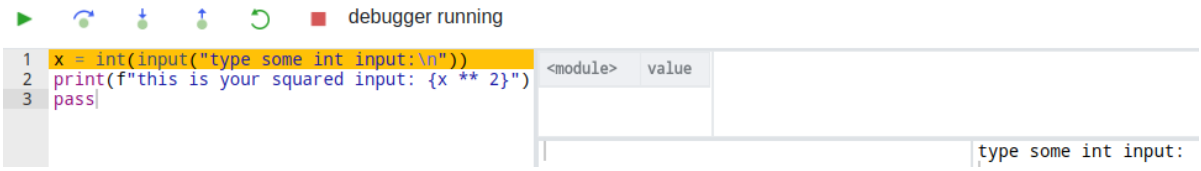


**Figura 4.16** Aplicação bloqueada devido a falta da entrada requerida

Após o fornecimento da entrada, Willow marcará a linha da entrada lida pelo programa como na Figura 4.17, essa linha não será mais modificável pelo usuário.

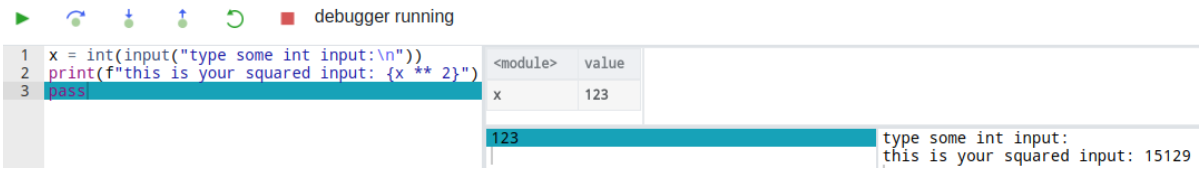


**Figura 4.17** Aplicação após ler entrada

As saídas do programa são exibidas no editor de saídas ao lado do editor de entradas, tanto as mensagens das funções de `print` como de `input` aparecem no editor de saídas.

### 4.6.4 Exceções

O Willow exibe as exceções com uma marcação vermelha na linha atual do programa, a propagação da exceção é exibida também na visualização, cada linha de chamada de função é passada até que a exceção seja tratada ou chegue ao escopo base e encerre a visualização, caso isso aconteça, o `traceback` da exceção é exibido no editor de saída do Willow, podemos ver na Figura 4.18 como o Willow exibe as exceções.

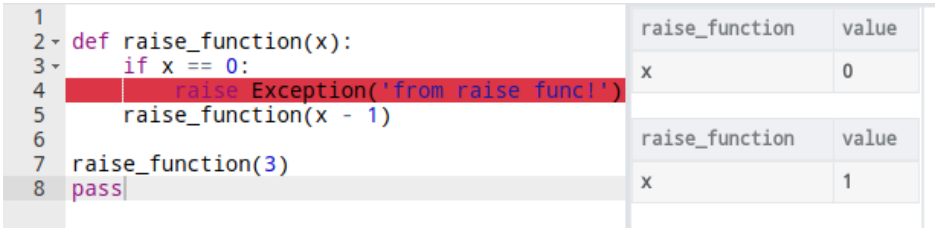


**Figura 4.18** Aplicação ao lançar uma exceção

CAPÍTULO 5

# Avaliação

Este capítulo apresenta a metodologia utilizada para analisar o nível de aceitação do Willow como ferramenta de auxílio no ensino de Introdução a Programação. Apresentamos o contexto da pesquisa, os participantes envolvidos, a ferramenta usada para coleta de dados, como foi aplicada a pesquisa e a análise dos resultados.

## 5.1 Contexto

Como já mencionado, Willow é uma ferramenta para criação de exemplos interativos de conceitos de programação e algoritmos. Este capítulo contempla uma pesquisa qualitativa para avaliar a usabilidade dessa ferramenta nos cursos de Introdução a Programação. A pesquisa foi realizada no Centro de Informática da Universidade Federal de Pernambuco (CIn - UFPE)

## 5.2 Participantes

Participaram da avaliação alunos monitores na disciplina de Introdução a Programação do Curso de Ciência da Computação. O corpo de monitores inclui alunos que estão em diferentes estágios do curso, desde o segundo período até alunos próximos a conclusão. Nesta disciplina existem muitas aulas de revisão onde os monitores são responsáveis por ministrá-las. Ao todo, sete monitores participaram da pesquisa.

## 5.3 Ferramentas de Coleta

### 5.3.1 Questionário

Elaboramos um questionário com dez questões, onde nove são de múltipla escolha e uma é aberta. O objetivo das questões é entender as percepções dos participantes sobre o que o Willow é capaz de fazer, o quão instrutivas são as visualizações geradas, se é possível demonstrar outros conceitos não relacionados diretamente à visualização do programa e também obter sugestões de funcionalidades ou indicação de problemas.

Foi feito um teste piloto do questionário com três monitores para avaliá-lo. Com esse teste foram corrigidos pequenos problemas em algumas questões.

A maioria das questões de múltipla escolha possuem quatro opções de resposta como por exemplo, a questão:

- A ferramenta pode ser usada para demonstrar como passagem de objetos por referência funciona?

Essa questão possui "não", "em alguns casos", "em muitos casos"e "sempre"como alternativas de resposta, outras questões seguem o mesmo padrão com quatro respostas possíveis, como "impossível usar", "difícil de usar", "fácil de usar"e "trivial".

As questões abaixo avaliam subjetivamente a capacidade do Willow para explicar alguns conceitos de programação e algoritmos simples.

- A ferramenta pode ser usada para explicar algoritmos recursivos?
- A ferramenta pode ser usada para demonstrar como passagem de objetos por referência funciona?
- É possível demonstrar o funcionamento de estruturas de dados simples como listas e árvores?

Algumas questões analisam a facilidade de manipular uma visualização e a qualidade das visualizações geradas, são elas:

- O quão fácil é usar as construções providas pela ferramenta para manipulação das visualizações?
- Qual o aspecto visual das visualizações geradas?

Uma das questões avalia a qualidade dos exemplos já providos pelo Willow para alguns algoritmos e estruturas de dados como listas encadeadas.

- Os exemplos básicos providos pela ferramenta são bons e podem ser usados sem modificação?

Duas das questões de múltipla escolha possuem apenas duas respostas possíveis, "sim"e "não", a primeira delas avalia a possibilidade do participante usar o Willow em sala de aula, e a segunda, a possibilidade de usar o Willow para explicar o funcionamento de depuradores, que é uma da ferramentas mais importantes para programadores, essas questões são:

- Você usaria a ferramenta em sala de aula?
- Você acha que é possível explicar o funcionamento de *debuggers* aos alunos com a ferramenta?

Uma das questões avalia se os participantes pensam que a ferramenta poderia ser usada para explicar conceitos de Programação à alunos que estão usando outra linguagem.

- Mesmo que a ferramenta use Python, você acha que a ferramenta pode ser usada com alunos aprendendo outra linguagem de programação?

A última questão era aberta e verificava se o participante possuía alguma opinião de funcionalidade ou crítica em relação ao Willow.

- Há alguma crítica ou sugestão para a ferramenta?

### 5.3.2 Vídeos

Além do questionário, criamos dois vídeos de programas em execução no Willow, estes vídeos são usados para demonstrar os conceitos da ferramenta e exemplos de algoritmos.

O primeiro vídeo[1] mostra como o Willow representa os tipos de dados mais simples como dados booleanos, números inteiros, números de ponto flutuante e strings. Em seguida são mostradas listas e dicionários, que são estruturas de dados da biblioteca padrão de Python. Após isso uma classe é declarada e instanciada para mostrar a forma com a qual esses elementos são representados. Com as variáveis declaradas, o vídeo mostra todas a opções de manipulação da exibição vistas na Seção 4.5.

O segundo vídeo[2] mostra a execução de dois algoritmos e usa algumas das opções de manipulação da Seção 4.5, o primeiro algoritmo é o número de *fibonacci*, onde o algoritmo faz marcações na pilha para indicar os casos base e se o escopo foi a primeira ou segunda chamada de função do escopo anterior. O segundo exemplo é a execução de múltiplas inserções numa lista encadeada, nele são mostrados passo a passo o processo de cada inserção com marcações nas variáveis a serem modificadas.

## 5.4 Aplicação da Pesquisa

A pesquisa foi feita online, inicialmente os participantes assistem aos vídeos explicados na Seção 5.3.2 para entender as funcionalidades do Willow. Com conhecimento sobre a ferramenta, podem então responder o questionário descrito no na Seção 5.3.1 através da do Google Formulários.

## 5.5 Análise dos Resultados

Como as alternativas das perguntas no questionário são autoexplicativas, não há margem para interpretações errôneas. Cada resposta possível, e não um conjunto delas, é considerada como uma das categorias de interpretação para a análise.

**A ferramenta pode ser usada para explicar algoritmos recursivos?**


**Figura 5.1** Resultado da primeira pergunta do questionário

[1] https://www.youtube.com/watch?v=zWWXWyhDSH4\&t=21s
[2] https://www.youtube.com/watch?v=ZZIi1yCc5aU\&t=76s

**A ferramenta pode ser usada para demonstrar como passagem de objetos por referência funciona?**

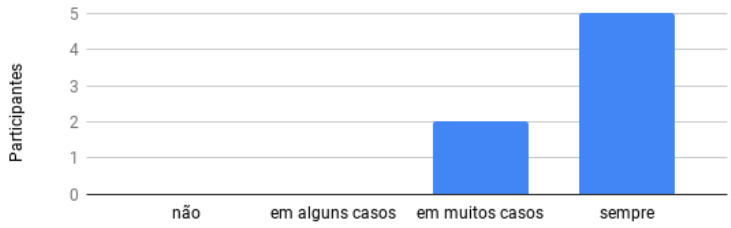


**Figura 5.2** Resultado da segunda pergunta do questionário

**É possível demonstrar o funcionamento de estruturas de dados simples como listas e árvores?**

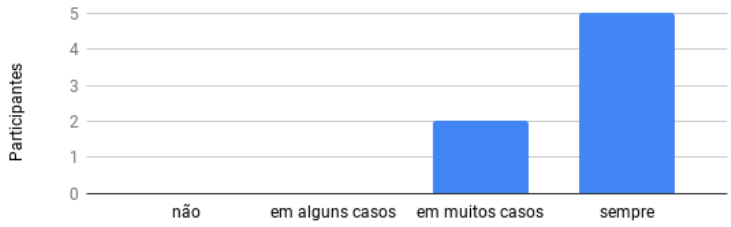


**Figura 5.3** Resultado da terceira pergunta do questionário

As primeiras três questões verificavam o que os participantes pensam que o Willow é capaz de representar com suas visualizações, a fim a ajudar com a explicação de conceitos de programação ou algoritmos. Observamos que houve grande aceitação da forma com a qual os conceitos são mostrados através da representação dos dados.

**Os exemplos básicos providos pela ferramenta são bons e podem ser usados sem modificação?**

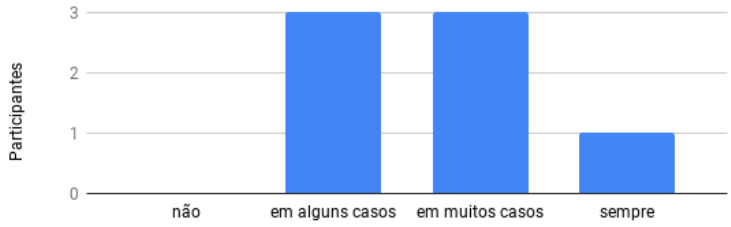


**Figura 5.4** Resultado da quarta pergunta do questionário

Neste momento, poucos exemplos pré-definidos são providos pelo Willow. Para a maioria dos problemas, os usuários tem que implementar todos os seus algoritmos e visualizações.

**Mesmo que a ferramenta use python, você acha que a ferramenta pode ser usada com alunos aprendendo outra linguagem de programação?**

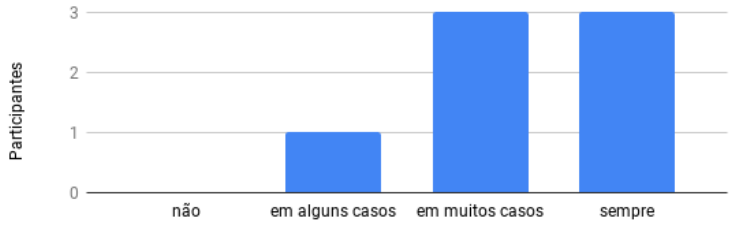


**Figura 5.5** Resultado da quinta pergunta do questionário

Um dos grande problemas para a utilização do Willow nas turmas de Introdução a Programação do curso de Ciência da Computação no CIn-UFPE é a linguagem de programação adotada. Atualmente o curso usa a linguagem Java. Apesar disto, os participantes acreditam que as visualizações geradas podem ser usadas para ensinar mesmo que a linguagem não seja Python.

**Qual o aspecto visual das visualizações geradas?**

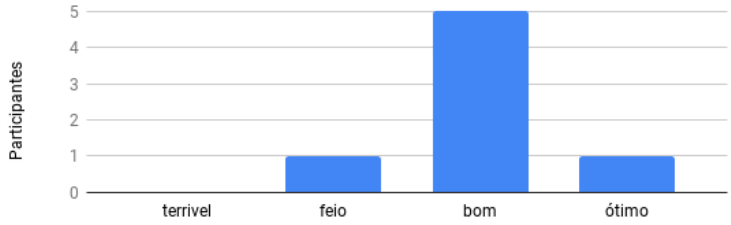


**Figura 5.6** Resultado da sexta pergunta do questionário

**O quão fácil é usar as construções providas pela ferramenta para manipulação das visualizações?**

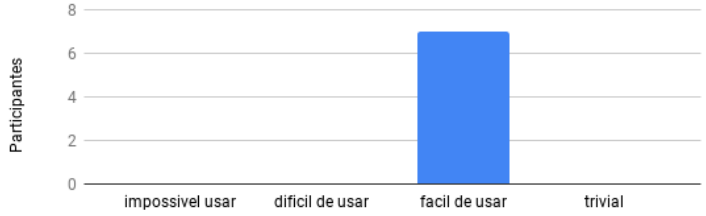


**Figura 5.7** Resultado da sétima pergunta do questionário

A qualidade da visualização é um aspecto importante, principalmente para chamar a atenção dos alunos. Apenas um dos participantes não gostou das visualizações. Apesar disto, estas podem ser personalizadas. Os participantes também concordaram que as visualizações são fáceis de manipular através das construções oferecidas pela ferramenta.

**Você usaria a ferramenta em sala de aula?**


**Figura 5.8** Resultado da oitava pergunta do questionário

Como os participantes ministram aulas de revisão, perguntamos se eles chegariam a usar a ferramenta durante as aulas. Todos concordaram que usariam a ferramenta. Isso mostra um alto nível de aceitação da ferramenta pelos participantes.

**Você acha que é possível explicar o funcionamento de *debuggers* aos alunos com a ferramenta?**

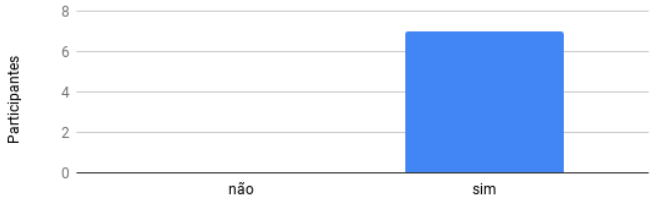


**Figura 5.9** Resultado da nona pergunta do questionário

Como o Willow tem uma estrutura parecida com a de depuradores, os participantes concordaram que há potencial para ensinar sobre as ferramentas de depuração.

**Há alguma crítica ou sugestão para a ferramenta?**

As maiores críticas foram em relação a linguagem de programação Java não ser suportada, pois é a linguagem ensinada na disciplina de Introdução a Programação atualmente. Também houveram críticas relacionadas a situações em que há referências entre objetos. Dois dos participantes da avaliação acharam a direção das referências difícil de identificar.

Outra reclamação foi em relação a restrição do posicionamento dos objetos na visualização, que não podem ser movidos para fora da região do visualizador.

As críticas também foram tomadas como sugestões, pois correções aos problemas citados pelos participantes podem ser implementadas na ferramenta.

CAPÍTULO 6

# Conclusão

Este trabalho teve como objetivo propor e avaliar uma nova ferramenta de auxílio no ensino de programação e algoritmos que possa ser efetivamente usada nas disciplinas iniciais dos cursos de Tecnologia da Informação. Ferramentas que auxiliam no ensino são de extrema relevância para a sociedade em geral, visto que a redução da quantidade de reprovados e desistentes das universidades podem diminuir a quantidade de desperdícios econômicos e sociais.

Após buscar e identificar os motivos que provocam a alta taxa de evasão de alunos nas disciplinas de Introdução a Programação e Algoritmos e Estruturas de Dados e avaliarmos ferramentas que são usadas na prática nos cursos Superiores da área de Tecnologia da Informação, desenvolvemos uma ferramenta com conceitos de programação interativa capaz de criar visualizações interativas e personalizáveis, provendo auxílio visual e interativo aos alunos. Com aulas menos tediosas, haverá maior interesse dos alunos, aumentando sua participação.

Esta ferramenta foi avaliada por meio de um estudo preliminar com monitores da disciplina de Introdução a Programação responsáveis por ministrar aulas de revisão. Os resultados obtidos indicam que houve boa aceitação da ferramenta, mas são necessários mais estudos para validar o seu uso nas salas de aula.

## 6.1 Trabalhos Futuros

- Implementar algumas das sugestões feitas pelos participantes, aprimoramento nas representações das referências dos objetos e maior flexibilidade no posicionamento dos objetos;
- Exibição personalizada de certos tipos de estruturas de dados como pilhas, filas, árvores e grafos;
- Habilitar modificação nas estruturas de dados padrão de Python, como listas e dicionários;
- Implementar o suporte a outras linguagens de programação como Java;
- Realizar um estudo em maior escala em uma disciplina do Centro de Informática, envolvendo não apenas monitores, mas também alunos e professores

# Referências Bibliográficas


[BSV+09] Marcelo RG Barbosa, Felipe A Silva, M de A Victor, Valéria D Feltrim, Luiz GB Mirisola, Paulo C Gonçalves, Josué JG Ramos, and Lucas T Alves. Implementação de compilador e ambiente de programação icônica para a linguagem logo em um ambiente de robótica pedagógica de baixo custo. In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, volume 1, 2009.

[CPP08] ANM Imroz Choudhury, Kristin C Potter, and Steven G Parker. Interactive visualization for memory reference traces. In *Computer Graphics Forum*, volume 27, pages 815–822. Wiley Online Library, 2008.

[CR11] ANM Imroz Choudhury and Paul Rosen. Abstract visualization of runtime memory behavior. In *Visualizing Software for Understanding and Analysis (VISSOFT), 2011 6th IEEE International Workshop on*, pages 1–8. IEEE, 2011.

[dCMG13] Michael da Costa Mora and Lucia Maria Martins Giraffa. Evasão na disciplina de algoritmo e programação: um estudo a partir dos fatores intervenientes na perspectiva do aluno. In *Tercera Conferencia sobre el Abandono en la Educación Superior (III CLABES, 2013, Espanha.*, 2013.

[dSRBB12] Romenig da Silva Ribeiro, Leônidas de O Brandão, and Anarosa AF Brandão. Uma visão do cenário nacional do ensino de algoritmos e programação: uma proposta baseada no paradigma de programção visual. In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, volume 23, 2012.

[FT00] Roy T Fielding and Richard N Taylor. *Architectural styles and the design of network-based software architectures*, volume 7. University of California, Irvine Doctoral dissertation, 2000.

[GHM08] Anabela Gomes, Joana Henriques, and António Mendes. Uma proposta para ajudar alunos com dificuldades na aprendizagem inicial de programação de computadores. *Educação, Formação & Tecnologias-ISSN 1646-933X*, 1(1):93–103, 2008.

[Guo13] Philip J Guo. Online python tutor: embeddable web-based program visualization for cs education. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 579–584. ACM, 2013.

[JSO] Introducing json. `http://www.json.org/`. Acessado em: 26/06/2018.

[MRR+10] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 10(4):16, 2010.

[OZS12] Michael C. Orsega, Bradley T. Vander Zanden, and Christopher H. Skinner. Experiments with algorithm visualization tool development. In *SIGCSE*, 2012.

[PJF10] Dilermando Piva Jr and Ricardo L Freitas. Estratégias para melhorar os processos de abstração na disciplina de algoritmos. In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, volume 1, 2010.

[SFMHL07] Roberto Leal Lobo Silva Filho, Paulo Roberto Motejunas, Oscar Hipólito, and Maria Beatriz Carvalho Melo Lobo. A evasão no ensino superior brasileiro. *Cadernos de pesquisa*, 37(132):641–659, 2007.

[Syk07] Edward Sykes. Determining the effectiveness of the 3d alice programming environment at the computer science i level. 36:223–244, 05 2007.

[Zel09] Andreas Zeller. *Why programs fail: a guide to systematic debugging*. Elsevier, 2009.