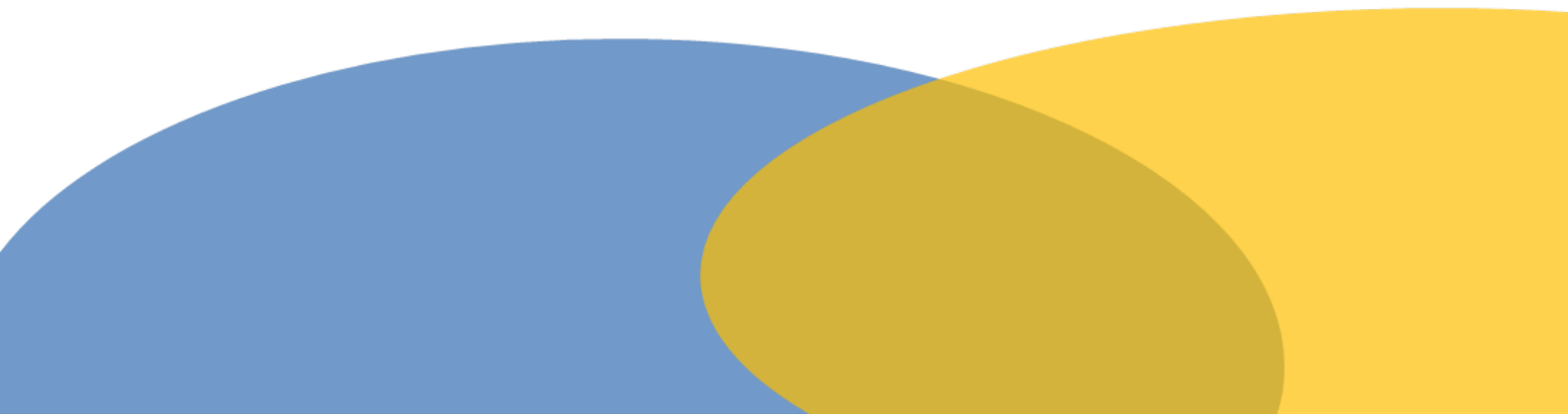
Introdução à Linguagem Python

Paradigmas de Linguagens de Programação

Rômulo Souza Fernandes
Ausberto S. Castro Vera

26 de setembro de 2022



UENF - UNIVERSIDADE ESTADUAL DO NORTE FLUMINENSE DARCY RIBEIRO

CCT - CENTRO DE CIÊNCIA E TECNOLOGIA
LCMAT - LABORATÓRIO DE MATEMÁTICAS
CC - CURSO DE CIÊNCIA DA COMPUTAÇÃO

Primeira edição, Setembro 2022

Sumário

1. Introdução

O Python é uma linguagem orientada a objetos de alto nível, que possui uma sintaxe simples e objetiva, assim colaborando para a fácil compreensão do código-fonte e permitindo que a linguagem seja produtiva. O Python contém várias estruturas de alto nível, como hora, data, dicionários, listas, complexos, entre outras estruturas. Contém um amplo conjunto de módulos disponíveis para utilização, frameworks que podem ser acrescentados, ferramentas de outras linguagens atuais, como persistência, unidades de teste, geradores, introspecção e metaclasse, além de ter disponíveis diversas bibliotecas, como IPython, Matplotlib, mIPy, NumPy, Pandas, SciPy, ScraPy, entre outras bibliotecas conhecidas.

O Python é uma linguagem multiparadigma, suportando a programação orientada a objetos, modular e funcional. A linguagem Python foi criada na Holanda, no ano de 1990, por Guido van Rossum, no Instituto Nacional de Pesquisa para Matemática e Ciência da Computação. [Bor14]

A linguagem Python é de código aberto, porém o criador Guido van Rossum possui a função central de decidir a evolução da linguagem. O Python se popularizou e se tornou a linguagem de desenvolvimento de aplicações mais indicada para iniciantes, assim sendo aconselhada como primeira linguagem de programação. [Per16]

1.1 História da linguagem Python

O intuito de Guido van Rossum era criar uma linguagem que pudesse suprir suas exigências, assim criando o Python, com base na linguagem ABC, mas solucionando os problemas encontrados por ele na linguagem. O Python tinha como usuários principais os engenheiros e físicos.

A seguir um pouco da história da linguagem Python, baseados em [Per16] e [Bor14] :

- O Holandês Guido van Rossum foi o autor principal da linguagem Python. O autor trabalhava no CWI (Centrum Wiskunde & Informatica), localizada em Amsterdã na Holanda.
- O nome Python não veio da espécie de serpente e sim do seriado de comédia preferido do autor da linguagem, chamado Monty Python's Flying Circus.
- A versão 0.9.0 do Python foi lançado em 1991, incluindo manipulação de exceções, classes, listas e strings. Incluía também alguns aspectos de programação funcional como lambda, maps, filter e reduce.

- No ano de 1995, o autor da linguagem continuou seu trabalho sobre Python na Corporation for National Research Initiatives (CNRI) em Reston, Virginia, USA.
- Em Maio de 2000, Guido van Rossum e o grupo de desenvolvimento do Python se mudaram para BeOpen.com, assim formando a equipe BeOpen PythonLabs.
- A versão 1.6 do Python foi lançada em 5 de setembro de 2000.
- A versão 2.0 do Python foi lançada em 16 de outubro de 2000.
- A versão 3.0 do Python foi lançada em 3 dezembro de 2008.

1.2 Áreas de Aplicação da Linguagem

O Python está entre as linguagens de programação mais utilizadas no mundo, é muito utilizado por usuários individuais, mas sua aplicação se estende para empresas reais. A natureza do Python é de propósito geral, assim tornando a linguagem aplicável em quase todas as áreas. Como a IBM, Seagate e Hewlett-Packard, que utilizam o Python para testes de hardware, o Yahoo! e Google usam a linguagem em serviços de Internet, já a empresa Industrial Light and Magic e outras empresas de filmes utilizam o Python na produção de animações. Entre todas as aplicações para o Python atualmente, o ponto em comum é que a linguagem é usada em todo o espectro, em questão de domínios de aplicação. [Lut07]

1.2.1 Big Data

Atualmente o Python é uma das melhores linguagens de programação para trabalhar com Big Data. Um dos motivos dessa preferência de uso é o suporte avançado de inúmeras bibliotecas e frameworks, muitas das bibliotecas são voltadas para lidar com Big Data, dando suporte e auxiliando na implementação de algoritmos de Machine Learning e Data Analytics. Abaixo algumas das bibliotecas de software livre:

- SciPy: Utilizada para computação técnica e computação científica, possibilita a interpolação, otimização, integração e modificação de dados utilizando funções especiais, álgebra linear, etc.
- NumPy: Utilizada para computação numérica para dados com formas de grandes matrizes multidimensionais e Arrays. A biblioteca também oferece diversas funções matemáticas de alto nível, para manipular os dados com transformada de Fourier, álgebra linear, processamento de números aleatórios, etc.
- Scikit-learn: Utilizada para Machine Learning, relacionada a vários algoritmos de regressão, clustering e classificação. Pode ser utilizada também em conjunto com outras bibliotecas, como a NumPy e SciPy.
- Pandas: Utilizada para análise e manipulação de dados, oferece diversas estruturas de dados e operações para manipulação de dados, no formato de séries temporais e tabelas numéricas. A biblioteca também dispõe de diferentes ferramentas para gravar e ler dados, entre estruturas de dados na memória e diferentes formatos de arquivo.

O Python possui uma sintaxe simples, possibilitando uma fácil leitura do código, assim tanto os iniciantes quanto os desenvolvedores experientes, podem se concentrar melhor no objetivo, ao invés de se desgastar se concentrando nas nuances técnicas da linguagem que está utilizando. Sendo assim, o Python é a linguagem preferida dos Cientistas de dados e Engenheiros de Big Data.

A linguagem Python é extremamente flexível, permitindo finalizar mais trabalhos com menor número de linhas de código. O Python também é escalável na manipulação de dados em grandes quantidades, sendo um ponto muito importante quando se trata de Big Data. Comparando o Python com outras linguagens de programação utilizadas em Big Data Analytics, como R e Java, elas não são tão escaláveis e flexíveis como o Python, onde havendo um aumento no volume de dados, o Python sem dificuldades pode aumentar a velocidade de processamento dos dados, sendo uma

tarefa complicada para fazer em R ou Java. [McK19]

1.2.2 Orientação a objetos

O Python é uma linguagem multiparadigma, suportando diferentes abordagens de programação, um jeito de solucionar problemas de programação é criar objetos, esse método é conhecido como Programação Orientada a Objetos(POO) e a orientação a objetos é um dos paradigmas da linguagem Python, com isso, a criação de objetos e classes é mais simples.

No Python os dados são guardados em objetos, em outras linguagens, determinados tipos são guardados na memória, não em entidades abstratas como objetos. A programação orientada a objetos é um importante método de organizar e desenvolver códigos, focando em criar códigos reutilizáveis. [Lut07]

No Python, a programação orientada a objetos possui alguns pontos importantes, baseados no mesmo autor citado no parágrafo anterior:

- Encapsulamento: Restrição ao acesso de métodos e atributos de uma classe, evitando que os dados sejam alterados diretamente, na linguagem Python existe apenas o private e public.
- Métodos: São funções definidas no corpo da classe. Utilizados para definir o comportamento do objeto.
- Objeto: É a instância de uma classe. Somente a definição do objeto é definida no momento em que a classe é estabelecida. Consequentemente nenhum espaço de memória é alocado.
- Classe: Juntam as funcionalidades de um certo objeto, servindo como um modelo.
- Polimorfismo: É a capacidade de utilizar uma interface para vários tipos de dados. Possibilitando que o objeto possua o poder de assumir diversas formas.
- Herança: Cria uma classe nova que será descendente e irá utilizar particularidades de outra classe já existente, sem realizar modificações na mesma.

1.2.3 Pentest

O Python está entre as melhores linguagens para pentest e segurança da informação, isso é devido a grande quantidade de bibliotecas, ferramentas, frameworks, por ter versatilidade e ser multiplataforma, entre outros pontos que tornam o Python uma das melhores linguagens para essa aplicação, como ter um código de fácil leitura e sintaxe simples. A linguagem é usada para diversas finalidades dentro do processo, possibilitando que as soluções sejam vinculadas e automatizadas sem dificuldades, como decodificar e enviar pacotes, varrer redes e portas, analisar malwares, acessar servidores, etc, com base em [Mor18] e [Sei15].

Como citado, a linguagem Python possui uma grande diversidade de ferramentas, de acordo com os mesmos autores citados no parágrafo anterior, essas são algumas das ferramentas disponíveis:

- Análise de malware:
  - Exefilter: Utilizada para filtrar tipos de arquivos para páginas web e e-mails, podendo detectar diversos tipos de arquivos e apagar o conteúdo ativo.
  - PyClamAV: Acrescenta a detecção de vírus maliciosos nas ferramentas do Python.
  - Pyew: Utilizada geralmente para analisar malwares, o pyew desmonta e edita hexadecimais de linha de comando.
- Utilitários de rede:
  - Dpkt: Utilizado para gerar e analisar pacotes de dados utilizando definições do protocolo TCP/IP.
  - Spoodle: Usado para verificar subdomínios e vulnerabilidade.
  - Knock Subdomain Scan: utilizado para retornar a lista de subdomínios do domínio de destino através da técnica de lista de palavras.
- Explorar bibliotecas:
  - Scapy: É uma biblioteca e ferramenta de processamento de pacotes, podendo decodificar

diversos protocolos, enviar pela rede, capturar e combinar respostas e solicitações. Oferece funções como as oferecidas pelo Tcpdump, Wireshark e Nmap, também oferece acesso programático.
  - Python Nmap: Usado para analisar os resultados da varredura do Nmap e lançar ataques particularizados contra hosts específicos.
  - Monda: O Monda é um depurador de imunidade, que auxilia no desenvolvimento de programas de exploração.
- Forense
  - Rekall: O Rekall é um framework criado pelo Google para realizar varreduras e análises de memória.
  - LibForensics: É uma biblioteca para desenvolvimento de aplicativos Forenses digitais
  - Aft: É um pacote de ferramentas forenses voltado para Android.

2. Conceitos básicos da Linguagem Python

Neste capítulo é apresentado alguns conceitos básicos da linguagem de programação Python, como tipos de dados básicos aceitos pela linguagem, que são inteiro, ponto flutuante, booleano, string e lista. Alguns dos livros indicados para iniciar o estudo sobre a linguagem de programação Python são: [Lut07], [Per16].

2.1 Variáveis, constantes e atribuições

Como já sabemos pela álgebra, é útil atribuir nomes a valores, e chamamos esses nomes de variáveis. Por exemplo, o valor 3 pode ser atribuído à variável x em um problema da álgebra da seguinte forma: x = 3. A variável x pode ser imaginada como um nome que nos permite apanhar o valor 3 mais adiante. Para recuperá-lo, só precisamos avaliar x em uma expressão. O mesmo pode ser feito em Python. Um valor pode ser atribuído a uma variável: »> x = 4 A instrução x = 4 é denominada instrução de atribuição. O formato geral de uma instrução de atribuição é: <variável> = <expressão> Uma expressão que nos referimos como <expressão> se encontra no lado direito do operador =; ela pode ser algébrica, Booleana ou outro tipo de expressão. No lado esquerdo está uma variável denominada <variável>. A instrução de atribuição atribui a <variável> o valor avaliado pela <expressão>. No último exemplo, x recebe o valor 4. Quando um valor tiver sido atribuído a uma variável, a variável pode ser usada em uma expressão Python: »> x 4 Quando o Python avalia uma exemplo contendo uma variável, ele avalia a variável para seu valor atribuído e depois realiza as operações na expressão: »> 4 * x 16 Uma expressão envolvendo variáveis pode aparecer no lado direito de uma instrução de atribuição: »> contador = 4 * x Na instrução contador = 4 * x, x é primeiro avaliada como 4, e depois a expressão 4 * 4 é avaliada como 16, e depois 16 é atribuído à variável contador: »> contador 16 Até aqui, definimos dois nomes de variável: x com o valor 4 e contador com o valor 16. E o que podemos dizer, por exemplo, do valor da variável z, que não foi atribuída ainda? Vejamos: »> z Traceback (most recent call last): z NameError: name 'z' is not defined Não temos certeza do que esperávamos... mas obtivemos aqui nossa primeira (e, infelizmente, não a última) mensagem de erro. Acontece que, se a variável — z, neste caso — não teve um valor atribuído, ela simplesmente não existe. Quando o Python tenta avaliar um nome não atribuído, ocorrerá um erro e uma mensagem (como name 'z' is not defined) é emitida.

Aprenderemos mais sobre erros (também chamados de exceções) no Capítulo 4.

2.2 Tipos de Dados Básicos

2.2.1 String

Um string é uma sequência de caracteres considerado como um item de dado simples. Para Python, um string é um array de caracteres ou qualquer grupo de caracteres escritos entre doble aspas ou aspas simples. Por exemplo,

```
>>> #usando aspas simples
>>> pyStr1 = 'Brasil'
>>> print (pyStr1)  Brasil
>>> #usando aspas duplas
>>> pyStr2 = "Oi, tudo bem?"
>>> print (pyStr2)
Oi, tudo bem?
```

- *Concatenação de strings*
  Strings podem ser concatenadas utilizando o operador +, e o seu comprimento pode ser calculado utilizado o operador `len(string)`

```
>>> # concatenando 2 strings
>>> pyStr = "Brasil" + " verde amarelo"
>>> print (pyStr)
Brasil verde amarelo
>>> print (len(pyStr))
20
```

- *Operador de indexação*
  Qualquer caracter de um string o sequência de caracteres pode ser obtido utilizando o operador de indexação []. Existem duas formas de indexar em Python, os caracteres de um string:

  Index com inteiros positivos indexando a partir da esquerda começando com 0 e onde 0 é o index do primeiro caracter da sequência

  Index com inteiros negativos indexando a partir da direita começando com -1, e onde -1 é o último elemento da sequência, -2 é o penúltimo elemento da sequência, e assim sucessivamente.

```
>>> # Indexando strings
>>> pyStr = "Programando"
>>> print (len(pyStr))
11
>>> print (pyStr)
Brasil verde amarelo
```

- *Operador de Fatias*
  O operador de acesso a itens (caracteres individuais) também pode ser utilizado como operador de fatias, para extrair uma fatia inteira (subsequência) de caracteres de um string. O operador de Fatias possui três sintaxes:
  `seq[ inicio ]`
  `seq[ inicio : fim ]`
  `seq[ início : fim : step ]`
  onde `início`, `fim` e `step` são números inteiros.

```
>>> # Indexando strings
>>> pyStr = "Programando Python"
>>> print (len(pyStr))
11
>>> print (pyStr)
Brasil verde amarelo
```

2.2.2 Lista

Em muitas situações, organizamos os dados em uma lista: uma lista de compras, uma lista de cursos, uma lista de contatos no seu telefone celular, uma lista de canções no seu player de áudio e assim por diante. Em Python, as listas normalmente são armazenadas em um tipo de objeto denominado lista. Uma lista é uma sequência de objetos. Os objetos podem ser de qualquer tipo: números, strings e até mesmo outras listas. Por exemplo, veja como atribuiríamos a variável animais à lista de strings que representa diversos animais:

```
>>> animais = ['peixe ', 'gato', 'cao']
```

A variável animais é avaliada como a lista:

```
>>> animais
['peixe', 'gato', 'cao']
```

Em Python, uma lista é representada como uma sequência de objetos separados por vírgulas, dentro de colchetes. Uma lista vazia é representada como []. As listas podem conter itens de diferentes tipos. Por exemplo, a lista chamada coisas em »> coisas = [’um’, 2, [3, 4]] tem três itens: o primeiro é a string ’um’, o segundo é o inteiro 2 e o terceiro item é a lista [3, 4]. Operadores de Lista A maioria dos operadores de string que vimos na seção anterior pode ser usada em listas de formas semelhantes. Por exemplo, os itens na lista podem ser acessados individualmente usando o operador de indexação, assim como os caracteres individuais podem ser acessados em uma string: »> animais[0] ’peixe’ »> animais[2] ’cão’ Figura 2.3 Uma lista de objetos de string. A lista animais é uma sequência de objetos. O primeiro objeto, no índice 0, é a string ’peixe’. Índices positivos e negativos podem ser usados, assim como para as strings. A Figura 2.3 ilustra a lista animais junto com a indexação dos itens da lista. Índices negativos também podem ser usados: »> animais[-1] ’cão’ O comprimento de uma lista (ou seja, o número de itens nela) é calculado usando a função len(): »> len(animais) 3 Assim como as strings, as listas podem ser “adicionadas”, significando que podem ser concatenadas. Elas também podem ser “multiplicadas” por um inteiro k, que significa que k cópias da lista são concatenadas: »> animais + animais [’peixe ’, ’gato’, ’cão’, ’peixe ’, ’gato’, ’cão’] »> animais * 2 [’peixe ’, ’gato’, ’cão’, ’peixe ’, ’gato’, ’cão’] Se você quiser verificar se a string ’coelho’ está na lista, pode usar o operador in em uma expressão Booleana que é avaliada como True se a string ’coelho’ aparecer na lista animais: »> ’coelho’ in animais False »> ’cão’ in animais True Na Tabela 2.2, resumimos o uso de alguns dos operadores de string. Incluímos na tabela as funções min(), max() e sum(), que podem apanhar uma lista como entrada e retornar, respectivamente, o menor item, o maior item, a soma dos itens da lista: »> lst = [23.99, 19.99, 34.50, 120.99] »> min(lst) 19.99 »> max(lst) 120.99 »> sum(lst) 199.46999999999997 Tabela 2.2 Operadores de lista e funções. Somente alguns dos operadores de lista comumente usados aparecem aqui. Para obter a lista completa no seu shell interativo, use a função de documentação help(): »>help(list) Uso Explicação x in lst Verdadeiro se o objeto x estiver na lista lst; caso contrário, falso x not in lst Falso se o objeto x estiver na lista lst; caso contrário, verdadeiro lstA + lstB Concatenação das listas lstA e lstB lst * n,n * lst Concatenação de n cópias da lista lst lst[i] Item no índice i da lista lst len(lst) Comprimento da lista lst min(lst) Menor item na lista lst max(lst) Maior item na lista lst sum(lst) Soma dos itens na lista lst

2.2.3 Classes e Objetos

Até aqui, vimos como usar diversos tipos de valores que o Python admite: int, float, bool, str e list. Nossa apresentação foi informal para enfatizar a técnica normalmente intuitiva que o Python utiliza para manipular valores. Porém, a intuição nos leva somente até aí. Nesse ponto, damos um passo atrás por um instante para entender mais formalmente o que significa um tipo e os operadores e métodos admitidos pelo tipo. Em Python, cada valor, seja um valor inteiro simples (como 3) ou um valor mais complexo (como a string 'Hello, World!' ou a lista ['hello', 4, 5]) é armazenado na memória como um objeto. É útil pensarmos em um objeto como um contêiner para o valor que fica dentro da memória do seu computador. A ideia de contêiner captura a motivação por trás dos objetos. A representação real e o processamento de, digamos, valores inteiros em um sistema de computação é bastante complicada. No entanto, a aritmética com valores inteiros é bem natural. Os objetos são contêineres para valores, inteiros ou não, que ocultam a complexidade do armazenamento e processamento de inteiros e oferece ao programador a única informação de que ele precisa: o valor do objeto e qual tipo de operações que podem ser aplicadas a ele. Tipo de Objeto Cada objeto tem, associado a ele, um tipo e um valor. Isso pode ser visto na Figura 2.4, que ilustra quatro objetos: um objeto inteiro com valor 3, um objeto de ponto flutuante com valor 3.0, um objeto de string com valor 'Hello World' e um objeto de lista com valor [1, 1, 2, 3, 5, 8]. Figura 2.4 Quatro objetos. Ilustração de quatro objetos com tipos diferentes. Cada objeto tem, associado a ele, um tipo e um valor. O tipo de um objeto indica que tipo de valores o objeto pode manter e que tipo de operações podem ser realizadas sobre esse objeto. Os tipos que vimos até aqui incluem o inteiro (int), ponto flutuante (float), Booleano (bool), string (str) e lista (list). A função type() do Python pode ser usada para determinar o tipo de um objeto: »> type(3) <class 'int'> »> type(3.0) <class 'float'> »> type('Hello World') <class 'str'> »> type([1, 1, 2, 3, 5, 8]) <class 'list'> Quando usada sobre uma variável, a função type() retornará o tipo do objeto ao qual a variável se refere: »> a = 3 »> type(a) <class 'int'> AVISO Variáveis Não Têm um Tipo É importante observar que uma variável não tem um tipo. Uma variável é apenas um nome. Somente o objeto ao qual ela se refere tem um tipo. Assim, quando vemos »> type(a) <class 'int'> Na realidade, isso significa que o objeto ao qual a variável a atualmente se refere é do tipo inteiro. Enfatizamos atualmente porque o tipo de um objeto ao qual a se refere pode mudar. Por exemplo, se atribuirmos 3.0 à variável a: a = 3.0 então a se referirá a um valor do tipo float: »> type(a) <class 'float'> A linguagem de programação Python é considerada orientada a objeto porque os valores são sempre armazenados em objetos. Em linguagens de programação diferentes de Python, os valores de certos tipos não são armazenados em entidades abstratas, como objetos, mas explicitamente na memória. O termo classe é usado para se referir aos tipos cujos valores são armazenados em objetos. Como cada valor em Python é armazenado em um objeto, cada tipo Python é uma classe. Neste livro, usaremos classe e tipo significando a mesma coisa. Anteriormente neste capítulo, apresentamos diversos tipos numéricos do Python informalmente. Para ilustrar o conceito do tipo do objeto, agora vamos discutir seus comportamentos com mais detalhes.

2.2.4 Biblioteca Padrão Python

A linguagem de programação básica do Python vem com funções como max() e sum() e classes como int, str e list. Embora estas não sejam, de forma alguma, todas as funções e classes embutidas da linguagem Python, o núcleo da linguagem Python é deliberadamente pequeno, para fins de eficiência e facilidade de uso. Além das funções e classes básicas, Python tem muitas outras funções e classes definidas na Biblioteca Padrão Python. A Biblioteca Padrão Python (Python Standard Library) consiste em milhares de funções e classes organizadas em componentes chamados módulos. Cada módulo contém um conjunto de funções e/ou classes relacionadas a determinado domínio de aplicação. Mais de 200 módulos embutidos formam juntos a Biblioteca Padrão Python. Cada módulo na Biblioteca Padrão contém funções e classes para dar suporte à programação de

aplicações em um certo domínio. A Biblioteca Padrão inclui módulos para dar suporte, dentre outros, a: •Programação em rede •Programação de aplicação Web •Desenvolvimento de interface gráfica com o usuário (GUI) •Programação de bancos de dados •Funções matemáticas •Geradores de números pseudoaleatórios

2.2.5 Objetos Turtle Graphics

Em nosso primeiro estudo de caso, usaremos uma ferramenta gráfica para ilustrar (visualmente) os conceitos abordados neste capítulo: objetos, classes e métodos de classe, programação orientada a objeto e módulos. A ferramenta, Turtle graphics, permite que um usuário desenhe linhas e formas de um modo semelhante ao uso de uma caneta sobre o papel. DESVIO Turtle Graphics Turtle graphics tem uma longa história, desde a época em que o ramo da ciência da computação estava sendo desenvolvido. Ele fez parte da linguagem de programação Logo, desenvolvida por Daniel Bobrow, Wally Feurzig e Seymour Papert em 1966. A linguagem de programação Logo e seu recurso mais popular, turtle graphics, foi desenvolvida para fins de ensino de programação. A tartaruga (turtle) era originalmente um robô, ou seja, um dispositivo mecânico controlado por um operador de computador. Uma caneta era presa ao robô e deixava um rastro na superfície enquanto o robô se movia de acordo com as funções inseridas pelo operador. Turtle graphics está disponível a desenvolvedores Python através do módulo turtle. No módulo, estão definidas 7 classes e mais de 80 métodos de classe e funções. Não vamos examinar todos os recursos do módulo turtle. Só apresentaremos um número suficiente para nos permitir realizar gráficos interessantes enquanto cimentamos nosso aprendizado de objetos, classes, métodos de classe, funções e módulos. Fique à vontade para explorar essa divertida ferramenta por conta própria. Vamos começar importando o módulo turtle e depois instanciando um objeto Screen. »> import turtle »> s = turtle.Screen() Você notará que uma nova janela aparece com um fundo branco depois da execução do segundo comando. O objeto Screen é a tela de desenho na qual iremos desenhar. A classe Screen é definida no módulo turtle. Mais adiante, vamos apresentar alguns métodos de Screen que mudam a cor do fundo ou fecham a janela. No momento, só queremos começar a desenhar. Para iniciar nossa caneta ou, usando a terminologia turtle graphics, nossa tartaruga, instanciamos um objeto Turtle que chamaremos de t:

2.3 Tipos de Dados de Coleção

2.3.1 Tipos Sequenciais

2.3.2 Tipos Conjunto

2.3.3 Tipos Mapeamento

2.4 Estrutura de Controle e Funções

2.4.1 O comando IF

2.4.2 Laço FOR

2.4.3 Laço WHILE

2.5 Módulos e pacotes

2.5.1 Módulos

2.5.2 Pacotes

Código fonte para a linguagem Python:

```
    number_1 = int(input('Ingresse o primeiro numero: '))
    number_2 = int(input('Ingresse o segundo numero: '))

    # Soma
```

```
    print('{} + {} = '.format(number_1, number_2))
    print(number_1 + number_2)

    # Substra\c{c}\~{a}o
    print('{} - {} = '.format(number_1, number_2))
    print(number_1 - number_2)

    # Multiplica\c{c}\~{a}o
    print('{} * {} = '.format(number_1, number_2))
    print(number_1 * number_2)

    # Divis\~{a}o
    print('{} / {} = '.format(number_1, number_2))
    print(number_1 / number_2)
```

Referências Bibliográficas

[Bor14] Luiz Eduardo Borges. *Python para desenvolvedores: aborda Python 3.3*. Novatec Editora, Sao Paulo, SP, 2014. Citado na página 5.

[Lut07] M. Lutz. *Aprendendo Python*. Bookman, Porto Alegre, RS, 2007. Citado 3 vezes nas páginas 6, 7 e 9.

[McK19] Wes McKinney. *Python para analise de dados*. Novatec Editora, Sao Paulo, SP, May 2019. Citado na página 7.

[Mor18] Daniel Moreno. *Python para pentest*. Novatec Editora, Sao Paulo, SP, 2018. Citado na página 7.

[Per16] Ljubomir Perkovic. *Introducao a computacao usando Python: um foco no desenvolvimento de aplicacoes*. Rio de Janeiro, RJ, 2016. Citado 2 vezes nas páginas 5 e 9.

[Sei15] Justin Seitz. *Black Hat Python: Python Programming for Hackers and Pentesters*. No Starch Press, Sao Paulo, SP, 2015. Citado na página 7.

Disciplina: *Paradigmas de Linguagens de Programação 1970*
Linguagem: *LinguagemXYZabcd*
Aluno: *Rômulo Souza Fernandes*

Ficha de avaliação:

Aspectos de avaliação (requisitos mínimos)	**Pontos**
Introdução (Máximo: 01 pontos) • Aspectos históricos • Áreas de Aplicação da linguagem	
Elementos básicos da linguagem (Máximo: 01 pontos) • Sintaxe (variáveis, constantes, comandos, operações, etc.) • Cada elemento com exemplos (código e execução)	
Aspectos Avançados da linguagem (Máximo: 2,0 pontos) • Sintaxe (variáveis, constantes, comandos, operações, etc.) • Cada elemento com exemplos (código e execução) • Exemplos com fonte diferenciada (listing)	
Mínimo 5 Aplicações completas - Aplicações (Máximo : 2,0 pontos) • Uso de rotinas-funções-procedimentos, E/S formatadas • Uma Calculadora • Gráficos • Algoritmo QuickSort • Outra aplicação • Outras aplicações ...	
Ferramentas (compiladores, interpretadores, etc.) (Máximo : 1,0 pontos) • Ferramentas utilizadas nos exemplos: pelo menos DUAS • Descrição de Ferramentas existentes: máximo 5 • Mostrar as telas dos exemplos junto ao compilador-interpretador • Mostrar as telas dos resultados com o uso das ferramentas • Descrição das ferramentas (autor, versão, homepage, tipo, etc.)	
Organização do trabalho (Máximo: 01 ponto) • Conteúdo, Historia, Seções, gráficos, exemplos, conclusões, bibliografia • Cada elemento com exemplos (código e execução, ferramenta, nome do aluno)	
Uso de Bibliografia (Máximo: 01 ponto) • Livros: pelo menos 3 • Artigos científicos: pelo menos 3 (IEEE Xplore, ACM Library) • Todas as Referências dentro do texto, tipo [ABC 04] • Evite Referências da Internet	
Conceito do Professor (Opcional: 01 ponto)	
Nota Final do trabalho:	

Observação: Requisitos mínimos significa a *metade* dos pontos