Fundamentos em Python

Desenvolvedor Python

Erick Faria

2021

Fundamentos em Python

Bootcamp: Desenvolvedor Python

Erick Faria

Sumário

Capítulo 1. Introdução ao Python

Caro estudante, parabéns por ter escolhido esse Bootcamp! A linguagem Python é uma das mais conhecidas e utilizadas por várias pessoas ao redor do mundo. Além de cientistas, desenvolvedores de jogos, de *softwares* e inúmeros outros profissionais usam Python devido a sua flexibilidade e facilidade.

Nesta apostila, serão abordados os temas introdutórios fundamentais do módulo de Fundamentos em Python. O conteúdo aqui não é exaustivo, você precisará pesquisar mais se quiser ser um desenvolvedor Python. Entretanto, não se preocupe, em nossas aulas e com o conteúdo dessa apostila, vou te ajudar nos primeiros passos da sua caminhada.

O que é Python?

A definição mais direta é chamarmos o Python de uma linguagem de programação, assim como várias outras: Java, C++, Julia, etc. Python foi criada por Guido van Rossum e lançada em 1991. Desde então, a comunidade – grupo de pessoas que contribuem e usam a linguagem – fez adaptações e melhorias na linguagem Python.

Python é uma linguagem de programação de alto nível (uma linguagem feita para ser simples para humanos escreverem e lerem), que também é orientada a objetos. Comparada a outras linguagens de programação, Python é muito simples e fácil de aprender, porque requer uma sintaxe única que enfatiza a legibilidade.

Atualmente, Python é utilizada para diversas finalidades, sendo algumas das principais:

- Desenvolvimento Web
- Desenvolvimento de software
- Equações matemáticas, estatística
- Análise de bancos de dados

Capítulo 2. Instalando o Python

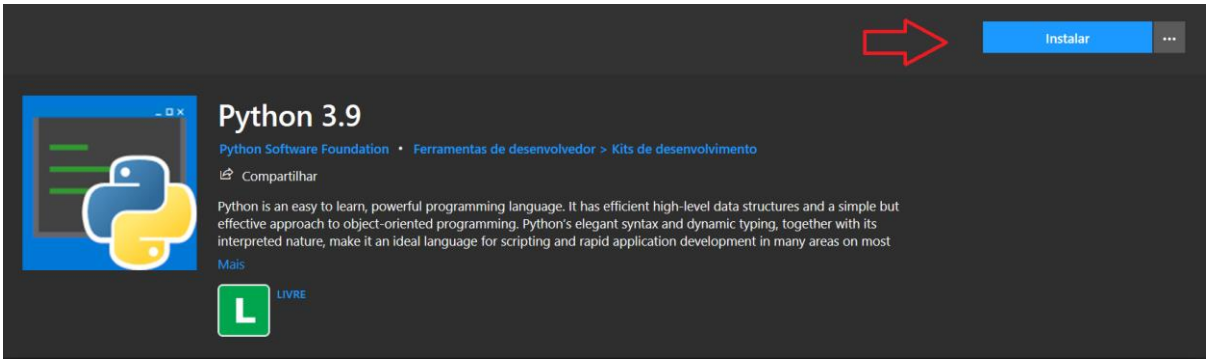
O Python pode ser instalado em múltiplas plataformas, seja o Windows, Mac ou Linux. Em todas elas você pode baixar, instalar e utilizar o Python de forma gratuita.

Windows

Para instalar o Python no Windows, você pode optar por duas formas: A primeira é pela loja da Microsoft, instalando via *Microsoft Store* do seu sistema operacional, ou fazendo o download direto do site da *Python Foundation*.

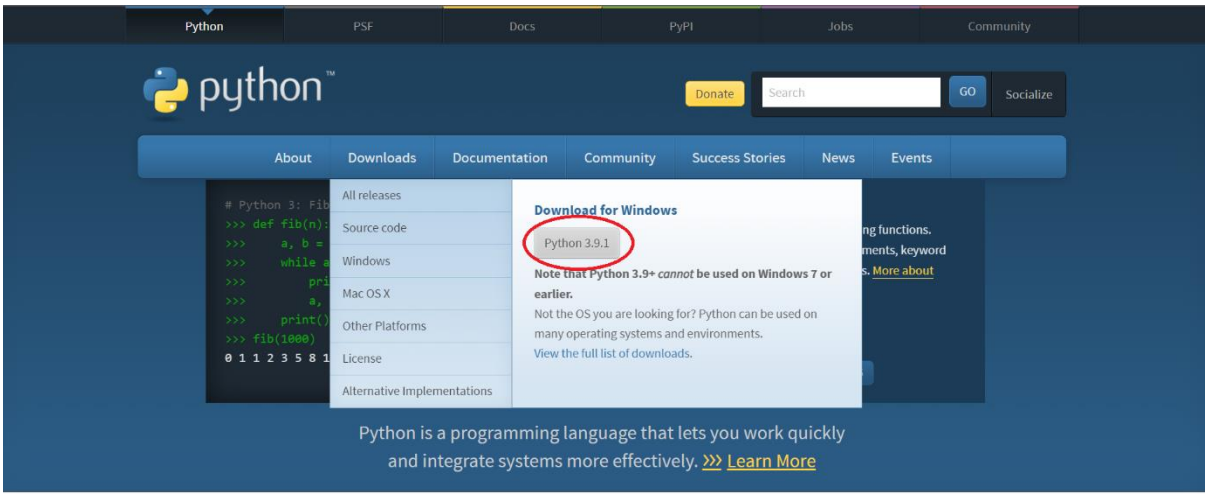
Por meio da *Microsoft Store*, basta abrir a loja, digitar Python no campo de busca, selecionar a versão mais nova do Python disponível e clicar em Instalar. Veja, na imagem abaixo, onde você deve clicar, indicado com uma seta vermelha.

Figura 1 – Instalando Python pela *Microsoft Store*



Outra forma é acessar o site: https://www.python.org, clicar em download, e depois fazer o download do arquivo como mostra a imagem abaixo.

Figura 2 – Página da *Python Foundation*



Linux

A instalação no ambiente Linux vai depender da distribuição que você está usando. Algumas distribuições já vêm, inclusive, com o Python instalado. Aconselho verificar antes de instalar, pois pode não haver a necessidade.

Em distribuições baseadas na distribuição Debian, você pode instalar o Python digitando as seguintes linhas de comando no seu terminal.

```
$ sudo apt-get install Python3
```

Mac

Para instalar o Python em seu Mac via terminal, siga os seguintes passos:

1. Para instalar o command line tools, digite em um terminal:

```
$ xcode-select --install
```

2. Para instalar o pip (gerenciador de pacotes Python), digite em um terminal:

```
$ sudo easy_install pip
```

3. Agora instale o Python 3:

```
$ brew install Python3
```

Google Colab

Se você preferir não instalar nada, ou está com dificuldades, existe a possibilidade de usar o Python gratuitamente de forma remota. Basta acessar o Google Colab, uma plataforma de desenvolvimento Python fácil e interativa. Para isso, basta fazer o *login* com sua conta google e acessar o seguinte endereço: https://colab.research.google.com/.

Nas aulas de conteúdo e nas aulas interativas desse módulo, será utilizado o Google Colab. O intuito é facilitar para todos(as) os(as) estudantes do curso. Se você tiver preferência por algum outro ambiente, não há problema; pode estudar como você se sentir melhor.

Capítulo 3. Primeiros Passos em Python

Chegou a hora de começarmos a programar em Python.

Abra seu IDLE Python e comece a praticar. Caso prefira, você pode usar o Google Colab (Plataforma do Google que permite o usuário a programar em Python sem necessidade de instalação). Caso você já tenha familiaridade com alguma outra IDE, fique à vontade para usar. O importante aqui é praticar.

Entretanto, antes de continuarmos em nossa jornada, gostaria de apresentar alguns dos termos mais utilizados em Python. São termos fundamentais e que serão utilizados em todo o *Bootcamp*. A tabela 1 é um glossário para você consultar em caso de dúvida. Por ser a língua inglesa o idioma oficial utilizado em programação, os termos foram mantidos em inglês, e a descrição traduzida para o português para facilitar a compreensão.

Tabela 1 – Glossário de termos mais utilizados no ambiente Python

Termos	Descrição
Instruction sets	Estes são os conjuntos de todas as instruções contidas no código da máquina que uma unidade de processamento central pode reconhecer e executar.
Binary	Binário é um conjunto de arquivos criados uma vez que você compila o código de objeto que é executado em máquinas.
Run time	Também chamado de tempo de execução, refere-se ao tempo em que um programa está sendo executado.
Shell	Shell é a camada de programação que entende e executa comandos que você digita como usuário.
Pip	Pip refere-se ao sistema de gerenciamento de pacotes usado para a instalação e o gerenciamento de pacotes de software escritos em Python.
IDLE (Integrated Development Environment)	O IDLE refere-se ao ambiente de desenvolvimento integrado do Python empacotado com a implementação padrão do idioma.

Class	Classe é uma espécie de mini-programa distinto que tem seu próprio contexto peculiar — ou seja, as propriedades ou variáveis e funções ou métodos.
Delimeter	Esta é a sequência de um único ou mais caracteres usados para especificar o limite entre regiões independentes e separadas em fluxos de dados, como texto simples.
Arguments	Argumentos são variáveis ou itens independentes que contêm códigos ou dados.
Assignment	Na programação, uma atribuição é uma instrução usada para definir em um nome variável, um valor.
Iteration	Iteração refere-se a um processo em que instruções ou estruturas são recursadas sequencialmente em um determinado número de vezes, ou até que alguma condição seja atendida.
Hashing	Isso se refere à mudança de uma sequência ou caracteres em um valor tipicamente mais curto com um comprimento fixo ou chave representando a sequência inicial.
Immutable object	Este é um objeto que tem um estado que se torna impossível de modificar uma vez que tenha sido criado.
Objects	Objetos são aquelas coisas que você pensa pela primeira vez quando está projetando um programa; eles são as unidades de código, em última análise, derivados do processo.
Null value	Refere-se a um não-valor; ele age como uma espécie de espaço reservado para um valor de dados desconhecido ou não especificado.
Modular design	Um design modular é uma abordagem de design que divide um sistema em pequenas seções conhecidas como skids ou módulos; você pode criar módulos de forma independente e, em seguida, usá-los em vários sistemas.
Docstring	Esta é uma sequência literal ocorrendo como a primeira instrução de função, módulo, definição de método ou classe.

Fonte: Adaptado de Menezes, (2019)

Além dos termos citados acima, existem também algumas palavras que são **reservadas** pelo Python. Essas palavras não podem ser utilizadas para nomear variáveis ou funções, por exemplo – devem ser usadas somente em momentos específicos, quando você desejar inseri-las em uma condição ou situação em que ela deve estar respeitando a sintaxe do Python. Veja, a seguir, quais são elas:

```
and       as       assert   break     class    continue
def       del      elif     else      except   exec
finally   for      from     global    if       import
in        is       lambda   nonlocal  not      or
pass      raise    return   try       while    with
yield     True     False    None
```

Uma dica para saber se você está utilizando uma palavra reservada é reparar a cor do seu IDE, pois, geralmente, as palavras reservadas têm cores específicas e serão facilmente notadas por você.

Agora que sabemos os termos mais utilizados em Python e as palavras reservadas, vamos começar a digitar nosso primeiro programa.

Execute Python Syntax

Ao abrir o prompt de comando do Python, você vai se deparar com três setas, ou sinal de maior/menor, se preferir. Esse sinal >>> quer dizer que o Python está pronto e aguardando seu comando.

O primeiro comando que vamos dar ao Python é:

```
print ('Olá Mundo')
```

O comando `print` é uma função que diz ao Python para imprimir aquilo que escrevermos dentro dos parênteses. Por se tratar de um texto, você deve sempre colocar entre aspas simples ‘ ou duplas “. Veja, a seguir, alguns exemplos:

```
print ('Estou adorando o bootcamp de Python')
print ('Vou beber um copo d\'água e voltar a estudar Python')
print ("Vou compartilhar com meus amigos sobre o bootcamp")
```

Repare que utilizei aspas simples e duplas e obtive o mesmo resultado. No segundo exemplo, você deve ter observado um item diferente. Foi adicionado um \ antes das aspas simples. Esse sinal deve ser inserido para não interromper seu texto.

O Python interpreta como textos tudo aquilo que está entre aspas. Assim, se o seu texto tiver aspas no interior, você deve incluir esse sinal \ antes.

A função `print` permite que você peça ao Python para imprimir números ou operações matemáticas. Porém, nesses casos não serão utilizadas as aspas por não se tratar de textos. Veja alguns exemplos:

```
print(5)
print(5+5)
print(2*8)
print(50/2)
```

Ao executar, você vai obter os seguintes resultados: 5; 10; 16 e 25. Isso ocorre porque no primeiro comando você pediu ao Python para imprimir o número 5, e nos demais, você pediu que ele realizasse contas para você.

Apesar de ser um exercício simples, conseguimos identificar que uma linguagem de programação parte do princípio de que você vai inserir um comando e o Python vai te dar uma resposta. Esse é a lógica da programação e com o Python não é diferente. Vamos agora pedir ao Python para executar comandos mais sofisticados.

Indentação em Python

Uma das maiores qualidades do Python é sua legibilidade e facilidade de leitura. É uma linguagem conhecida por executar comandos complexos em poucas linhas de código. Essa legibilidade se dá, entre outros fatores, pela indentação da linguagem. A indentação pode ser compreendida por alguns de vocês como “recuo” ou espaço. Vamos aos exemplos:

```
if 10 > 5:
    print ('10 é maior do que 5')
```

No comando acima eu usei o `if` (“se”, em português) seguido por uma expressão matemática 10 > 5 e dois pontos : . É o equivalente a estarmos conversando com o Python, seria algo como: “Python, se o número 10 for maior do que 5, imprima a frase ’10 é maior do que 5’).

Para executar comandos como esse, é preciso prestar atenção à indentação, aquele espaço (ou recuo) que foi dado entre a primeira linha do `if` e a linha abaixo do print. Isso quer dizer que tudo que estiver dentro daquele recuo faz parte de um bloco de código.

Vamos voltar a abordar a indentação logo à frente, quando começarmos a falar da criação de *loops* e criação de funções. Pratique alguns exemplos como esse fornecido mudando alguns parâmetros para que você fixe esse conceito. Será muito útil no futuro.

Variáveis em Python

Em Python, as variáveis são criadas a partir do sinal de igual (=) após um determinado nome que designará a variável. Para a criação de variável, existem algumas regras importantes a seguir:

- Uma variável não pode conter espaços
- Uma variável não pode começar com números
- Evitar usar acentos no nome de uma variável
- É aconselhável padronizar o nome das variáveis

Uma variável comporta vários tipos de dados, podem ser textos, números, expressões matemáticas, equações, etc. Independente do caminho que você seguir em Python, você irá lidar muito com elas, o que torna esse tópico muito importante. Vamos aos exemplos da criação de algumas variáveis:

```
numero = 5

Python = 'Melhor linguagem de Programação'

num_pi = 3.14

soma = 5+5
```

A primeira variável é apenas um número que se chama *numero* (sem acento propositalmente); a segunda chama-se Python com um texto atribuído; a terceira eu coloquei o valor de *pi* um número com casas decimais; e por último, no quarto exemplo, uma soma. Ao executar esses comandos, o Python não vai retornar nenhum valor imediatamente, ele vai apenas armazenar na sua memória *ram* os valores. Se você deseja imprimir os resultados, você deve executar o comando **`print`**.

```
print(numero)

print(Python)

print(num_pi)

print(soma)
```

Repare que na variável *num_pi* eu utilizei a abreviação *num* para me referir à palavra número. Eu poderia ter escrito a palavra *numero_pi*, porém, em meus códigos, eu utilizo uma padronização em minhas variáveis. Isso é muito útil para poupar tempo de digitação e facilitar a leitura de nossos códigos. Dessa forma, aconselho você a definir alguns padrões para suas variáveis, como por exemplo: *cod* para variáveis de códigos; o já citado *num* para variáveis de números; *som* ou *sum* para variáveis de soma etc. Assim, seu código ficará mais fácil de se ler e outras pessoas poderão contribuir mais facilmente.

Essa dica faz parte de uma série de recomendações de boas práticas em Python, que são normas que a comunidade Python segue para facilitar a interação e o compartilhamento da linguagem. Além de padronizar algumas coisas, existe também a possibilidade de deixar comentários em nosso código, o que é fortemente recomendado. Na seção a seguir, vamos aprender como fazê-los.

Comentários em Python

Os comentários são anotações que fazemos em nosso código que não são executados. Comentar seu código faz parte das boas práticas em Python, pois facilita outros usuários a lerem e a entenderem melhor o seu código. Comentar um código é muito útil, sobretudo quando estamos começando, pois a partir das aulas e dos exercícios que você vai desenvolver, é possível ir descrevendo o que está sendo feito.

Para comentar um código em Python, basta você colocar o sinal de cerquilha, conhecida por muitos atualmente como *hashtag* (#). Veja, a seguir, alguns exemplos de como comentar seu código:

```
# A função print imprime um resultado
print('Estou adorando aprender Python')

# A variável a seguir define o valor de pi
num_pi = 3.14
```

Observe que facilita muito a compreensão do que estamos fazendo. Ao descrever os códigos, fica mais fácil para você e outro leitor(a) do código interpretar o que você fez. Existem outras formas de se comentar um código, podendo fazê-lo na frente da linha digitada, ou usando três aspas simples (''') ou três aspas duplas (""") para comentários que ultrapassam mais de uma linha. Veja a seguir:

```
soma = 5+5 # Criando uma variável de soma

'''
A partir do momento que coloquei três aspas simples, tudo o que
eu escrever aqui dentro será considerado comentário.
É muito útil quando queremos explicar alguma coisa, ou deixar
instruções dentro ou antes do código.
'''

"""
O mesmo ocorre com as aspas duplas.
Você pode escolher o que você quer usar.
A comunidade Python recomenda usar as aspas simples, mas fica a
seu critério escolher.
"""
```

Capítulo 4. Tipos de dados em Python

Até o momento eu utilizei termos como *variável de número* ou *inserir um texto* para facilitar o entendimento daqueles que estão chegando agora no Python. Entretanto, esses nomes, apesar de não estarem errados, não são utilizados para se referir aos tipos de dados em Python. Você que já utilizou algum *software* estatístico ou já programou alguma vez, pode estar familiarizado com as palavras como: *string, integral, float*, *dictionary* e *boolean.*

Essas palavras referem-se ao tipo de dado que estamos inserindo ou lidando em Python. Nesse capítulo, vamos abordar cada um desses tipos e mostrar como eles se comportam em Python.

Strings

Strings são os textos ou as letras que inserimos o Python com o intuito de ser lido como tal. Para que o Python entenda que uma entrada é uma *string,* ela deve estar entre aspas duplas ou aspas simples, conforme mencionado anteriormente.

Alguns dos exemplos mais comuns de variáveis do tipo *string* são: endereços, nomes de lugares ou pessoas, nome de cidades, etc. Lembre-se do que foi dito no capítulo precedente: caso você vá utilizar uma palavra que comporte o sinal de aspas, você precisa utilizar o sinal de contrabarra para o Python não interromper sua string. Veja alguns exemplos de variáveis do tipo *string*:

```
cidade = 'Belo Horizonte'
lugar = 'IGTI'
curso = 'Desenvolvedor Python'
prof = 'Erick Faria'
aluno = 'Zé das couves'
```

Integer

As variáveis do tipo *integer* armazenam os números inteiros, ou seja, aqueles que não têm casas decimais. Os dados do tipo *integer* são representados pela

abreviação `int` em Python. Esse tipo de dado pode ser armazenado na forma positiva ou negativa representado com um símbolo – na frente.

São inúmeras as variáveis que são representadas pelo tipo `int`, sendo alguns exemplos: número de um endereço, número de alunos, quantidade de grãos colhidos em uma safra, etc.

Quando queremos criar uma variável do tipo `int` em Python, devemos inserir os números entre parênteses. Veja alguns exemplos:

```
num_alunos = (1580)
num_habitantes = (3000)
```

Observe que eu não coloquei o ponto (.) para separar os milhares dos números, pois caso eu faça isso, a variável será interpretada como `float`, o próximo tipo de dado que vamos estudar.

Em alguns casos, pode ser que você precise ter certeza de que a variável que você está criando seja do tipo `int`. Nesse caso, você deve dizer ao Python utilizando o seguinte comando:

```
num_alunos = int(1580)
num_habitantes = int(3000)
```

Repare que escrevi `int` antes de inserir os valores dentro dos parênteses. Isso é útil em alguns casos em que o usuário deve inserir um valor e evita que o Python se confunda com dados do tipo `float`.

Float

Os números que comportam casas decimais são chamados de dados do tipo `float`[1]. Em Python, os números podem ter essa variação. Na seção anterior, vimos que os números inteiros são do tipo `int` e os números com casas decimais, apesar de serem também números, são categorizados de forma diferente. É importante guardar essa informação para evitar confusões no futuro.

Diferentemente dos exemplos anteriores, para a criação dos dados do tipo `float` não existe uma abreviação. Alguns exemplos de variáveis do tipo `float` são: peso das pessoas ou objetos, altura de uma pessoa ou objeto, distâncias, etc. Veja, a seguir alguns exemplos:

```
distancia_km = (10.25)

altura = (1.75)

peso = (71.2)
```

E se quisermos ter certeza que a variável será reconhecida como `float`, basta adicionar a palavra float antes dos parênteses.

```
distancia_km = float(10.25)

altura = float(1.75)

peso = float(71.2)
```

É importante que suas casas decimais sejam separadas utilizando pontos (.) como nos exemplos a seguir, jamais vírgulas. Isso se deve à forma com que o Python lê os dados. Conforme veremos mais à frente, os elementos de uma variável que compõem uma *list* ou *tuple* são separados por vírgulas. Veja um exemplo:

```
# Exemplo de uma list
distancia_km = [85.45,98.45, 55.21]

# Exemplo de uma tuple
altura_alunos = (1.55,1.75,1.84)
```

[1] Na comunidade brasileira, algumas pessoas chamam de dados flutuantes. Aconselho a utilizar sempre os termos em inglês.

Dictionary

Os *dictionary* são dados que compõem um par de valores, sendo que um faz referência ao outro. Podemos fazer uma analogia com o dicionário que usamos na escola. O dicionário contém uma palavra seguida por uma acepção que traz o significado da palavra; em Python, as variáveis do tipo dictionary são semelhantes.

Variáveis do tipo dicionários são muito utilizadas quando vamos usar códigos que representam valores, geralmente do tipo string.

Veja, a seguir, alguns exemplos de como podemos usar *dictionary* em Python:

```
# Definindo código das UF's do sudeste brasileiro
uf_sud = {'MG':31, 'SP':35, 'RJ':33, 'ES':32}

# Definindo código da capital do Amazonas
cod_manaus = {'Manaus':1302603}

# Definindo o gentílico de quem nasce em Manaus
gentílico = {'Manaus':'manauara'}
```

Observe que em cada variável foram definidos dois valores, podendo ser a combinação: *string-string; int-int; int-string, string-int* ou mesmo *float.* O importante a guardar nesse momento é que os dicionários são compostos por dois valores, separados por um sinal de dois pontos (:).

Os dicionários são escritos entre chaves {} e separados por vírgulas. No próximo capítulo, vamos aprofundar melhor sobre como escrever esse tipo de dado em Python.

Boolean

Os dados do tipo *boolean* são informações que se limitam a dizer se algo é verdadeiro ou falso. Variáveis do tipo *boolean* têm como função dizer se um argumento é verdadeiro ou falso. Existem inúmeras aplicações para variáveis do tipo *boolean,* e elas serão muito úteis nos scripts em Python. Vamos ver alguns exemplos:

Eu vou “perguntar” ao Python se pela idade de uma determinada pessoa, ela pode ou não dirigir. Para isso, vou usar o comando `if` e os princípios de indentação que vimos no capítulo anterior.

```
maioridade = 18

idade_pessoa = 18

if idade pessoa == maioridade:
    print('Pode dirigir')
```

Observe que criei duas variáveis, uma com o valor da maioridade e outra com a idade da pessoa. Posteriormente, eu disse ao Python que se a idade da pessoa fosse igual ao valor da variável de maioridade, ela poderia dirigir. Você deve ter observado que utilizei dois sinais de igual juntos (==), isso é feito para consultar o Python se um argumento é `True` ou `False`.

Vamos ver outro exemplo:

```
nome = 'Zé das Couves'

if nome == 'Zé das Couves':
    print('Vai ter couve no almoço hoje')
```

No exemplo acima, eu disse ao Python que se o nome da pessoa for o ‘Zé das couves’, poderia imprimir uma mensagem. Isso será muito útil no futuro quando você for trabalhar com formulários, nos quais um usuário deve entrar um valor e você pode criar argumentos que retornem uma mensagem ou gerem algum outro resultado.

Você pode tentar alguns exemplos como:

```
# Consultar se um valor é igual ao outro
igual = 5==5

# Consultar se um valor é maior ou menor do que outro
consulta = 5 >= 6

# Testando strings
arroz == arroz
Python == java
```

Em todos esses casos, o Python vai te retornar um valor do tipo *boolean,* sendo `True` ou `False`.

Capítulo 5. List, Tuples, Dictionary e Sets

As *List, Tuple, Dictionary e Set* (na forma inglesa) são formatos de armazenamento de dados, geralmente em variáveis. São muito utilizados em Python, e cada um tem uma utilidade e particularidade em Python. Vamos ver, a seguir, um pouco mais de cada um deles.

Listas

As listas são um conjunto de dados que geralmente compõem uma variável que pode ser do tipo *float, int ou string.* **As listas são mutáveis**, ou seja, você pode alterar a ordem dos valores dentro de uma lista. Para criar uma lista em Python, você deve acomodar os valores entre colchetes [ ].

```
veiculo = ['carro', 'moto', 'bicicleta', 'velotrol']
lista = [152, 'ovo', 5865, 'mamão', 125, 2563, 1258, 'abacate']
num_mega_sena = [05, 25, 36, 45, 50, 57]
```

Observe, no exemplo acima, que uma lista pode ter valores do tipo *int, string* ou ambos, podendo ainda incluir valores do tipo *float*. Cada valor deve ser separado por vírgulas.

Tuples

As *tuples* são semelhantes às listas; acomodam valores geralmente em uma variável, podendo ser de qualquer um dos tipos de dados em Python. Entretanto, nas *tuples,* **os valores não são mutáveis**, ou seja, não há a possibilidade de mudar a

```
veiculo = ('carro', 'moto', 'bicicleta', 'velotrol')
lista = (152, 'ovo', 5865, 'mamão', 125, 2563, 1258, 'abacate')
num_mega_sena = (05, 25, 36, 45, 50, 57)
```

ordem dos valores de uma *tuple*. Para criar uma *tuple,* basta acomodar os valores entre parênteses ( ).

O exemplo acima é idêntico ao anterior, com somente uma alteração, pois no lugar dos colchetes eu coloquei os parênteses.

Dicionários

Os dicionários, conforme já vimos no capítulo anterior, são aqueles dados que comportam informações duplas e servem geralmente para atribuir um código de um dado, ou um valor de correspondência. Para criar um dicionário, você deve acomodar os valores entre chaves { } e atribuir os dois pontos: para fazer correspondência entre os valores no dicionário.

Veja, abaixo, um exemplo de utilização de dicionário:

```
cod_uf.replace({
11 : 'Rondônia',
12 : 'Acre',
13 : 'Amazonas',
14 : 'Roraima',
15 : 'Pará',
16 : 'Amapá',
17 : 'Tocantins',
21 : 'Maranhão',
22 : 'Piauí',
23 : 'Ceará',
24 : 'Rio Grande do Norte',
25 : 'Paraíba',
26 : 'Pernambuco',
27 : 'Alagoas',
28 : 'Sergipe',
29 : 'Bahia',
31 : 'Minas Gerais',
32 : 'Espírito Santo',
33 : 'Rio de Janeiro',
35 : 'São Paulo',
41 : 'Paraná',
42 : 'Santa Catarina',
43 : 'Rio Grande do Sul',
50 : 'Mato Grosso do Sul',
51 : 'Mato Grosso',
52 : 'Goiás',
53 : 'Distrito Federal'})
```

Observe que a utilização do dicionário foi muito útil para substituir os códigos das Unidades Federativas pelos respectivos nomes dos estados. A função `.replace` foi responsável pela substituição dos valores no exemplo.

Sets

Os *sets* são utilizados para armazenar múltiplos valores, assim como os anteriores, porém com algumas particularidades:

- Os valores não podem ser duplicados
- O *set* não pode ser modificado
- Os valores não são indexados
- Os valores são fora de ordem

Observe que o *Set* é um formato peculiar entre as 4 maneiras de armazenar dados em Python, e que mistura característica de alguns dos seus antecessores. Vejamos alguns exemplos de criação dos *sets*:

```
frutas = {'maçã', 'mamão', 'uva', 'abacate', 'melão'}

series = {'Friends', 'Big Bang Theory', 'Lost', 'House'}

num vacinas = {25, 21, 215, 214, 5648, 5486}
```

Observe que as *tuples* devem acomodar os valores entre chaves { }, assim como nos dicionários. Entretanto, no caso das *tuples,* não há o segundo valor separado pelos dois pontos.

Agora que aprendemos como armazenar os dados em Python, fica mais fácil para nós podermos continuar o nosso curso. Lembre-se que cada forma de armazenar tem um uso específico destinado. Quando for armazenar dados em Python, o primeiro passo é saber qual a melhor forma de armazenar.

Capítulo 6. Funções, While e For – Rotinas em Python

As funções estão diretamente relacionadas à rotina que você deve ou precisa criar em seus códigos. São mecanismos que vão facilitar a execução de seu programa e que precisa ser executada várias vezes.

Nós já fizemos o uso de funções várias vezes até o momento. Quando damos o comando `print`, estamos executando uma função nativa do Python. Contudo, existem alguns casos em que as funções nativas do Python não vão atender suas necessidades, por isso criamos funções que adequem e atendam suas necessidades de forma personalizada.

Nesse capítulo, vou apresentar os conceitos básicos de como criar e como funcionam as repetições em Python. Entretanto, é fundamental que você pratique, para que o conteúdo seja amplamente compreendido por você.

If - Elif

Nós já utilizamos a estrutura `if` em alguns momentos nos capítulos anteriores. Você já deve ter entendido que o `if` é uma estrutura condicional, ou seja, quando desejamos que uma condição seja satisfeita. Agora, imagine que você tenha várias condições a serem observadas no seu programa. Em Python, você não pode usar `if` várias vezes dentro do mesmo argumento; nesse caso, você deve usar o `elif`.

O `elif` é a mistura do `else` + `if`. Assim, podemos criar quantos argumentos desejarmos em nosso programa. Ao acabar de definir as condições, o último argumento deve ser sempre else, que é para dizer ao Python “todo o resto será isso, ou faça isso”. Vamos ver alguns exemplos:

```
idade = 18
if idade < 12:
    print('crianca')
elif idade < 18:
    print('adolescente')
elif idade < 60:
    print('adulto')
else:
    print('idoso')
```

Poderíamos continuar o exemplo acima com outros argumentos, não há um número máximo, mas lembre-se, criar demasiados argumentos em uma função pode deixar a execução comprometida.

For

A instrução `for - in` começa com a instrução for e, em seguida, uma variável criada pelo usuário (área) que representa cada item da lista. A instrução *for loop* atribui automaticamente um valor a essa variável, portanto, ela não precisa ser atribuída de antemão. A variável criada pelo usuário é seguida pela instrução in e a lista pretendida (AreaList).

A instrução `for` sempre termina com dois pontos, o que indica que você deseja executar uma ação para cada item da lista. A linha de código recuada indica que essa ação percorrerá a lista e será executada para cada item individual. As linhas de código que não são recuadas não serão executadas em loop por cada item.

Vamos aos exemplos:

```
for i in range(1,100):
    print(i)
```

No exemplo acima, eu pedi ao Python que criasse uma contagem a partir do número 1 até o valor na posição 100, utilizando o comando `range`. No fim, pedi ao Python que imprimisse o resultado. O exemplo pode ser lido da seguinte forma: “Python, do i que está dentro do `range` (que vai de 1 até 99), imprime o resultado”.

Vamos a outro exemplo, utilizando uma lista de nomes:

```
nomes = ['João', 'Maria', 'José', 'Geraldo', 'Sebastião']

for i in nomes:
    print(i)
```

A variável i após `for` é a lista inicializada dos valores, definida na variável nome. Assim, a instrução for percorreu todos os elementos e imprimiu o resultado no final.

While

O comando `while` serve para repetirmos alguma rotina em Python enquanto um argumento for verdadeiro ou satisfizer os argumentos que definimos. É recomendado utilizar o `while` quando não conhecemos o tamanho do arquivo, variável ou dado que vamos lidar. Um exemplo de utilização é contagem de número. Você deseja que o Python conte de 1 até 100. Para tal, o `while` será útil para nós. Veja o exemplo:

```
numero = 0

while numero <=100:
    numero += 1
    print(numero)
```

Execute o código em seu computador; você vai obter uma conta de 0 até 100. No código acima, foi dito ao Python “Enquanto a variável 0 for menor do que 100, você vai imprimir a variável número”, contudo, repare que na segunda linha eu tive de adicionar += 1. Isso foi necessário para que o Python somasse 1 a cada vez que ele lesse o código. Caso eu não fizesse isso, o número 0 ficaria se repetindo infinitamente. Por fim, eu pedi ao Python que imprimisse os resultados com a função `print`.

Funções

Para criar uma função em Python, você deve usar a palavra `def` antes de escrever os argumentos. Observe a função a seguir:

```
def area(point1, point2):
    x1, y1 = point1
    x2, y2 = point2
    return (x1*y2 - y1*x2)/2
```

Essa função foi criada para calcular a área de um polígono, pois é uma rotina que preciso executar muitas vezes em um programa que criei. Entretanto, não existe uma função nativa do Python para isso, e é pouco prático escrever a fórmula toda vez.

Agora que entendemos para que servem as funções, vamos entender o que cada item quer dizer. Como dito anteriormente, o `def` diz ao Python que estou criando uma função. Posteriormente, eu vou dar um nome a minha função; nesse caso, eu nomeei como *area*, mas poderia ser outro nome, como *calc_area*. Entre os parênteses, eu coloquei os argumentos que vão ser inseridos na minha função, que no exemplo mostrado são duas coordenadas.

Após esse primeiro passo, eu vou à linha posterior e escrevo os argumentos que quero que a função desempenhe, sempre **prestando atenção à indentação**, é muito importante se lembrar disso. No interior da função, você é livre para escrever os argumentos. Ao final, você deve escrever o comando `return`, para dizer ao Python que ao final da função você quer que ele retorne um resultado. No caso do exemplo, eu pedi para retornar o cálculo de área.

Agora que escrevi minha função, eu posso fazer como fizemos com o print ao longo de todo o processo. Dessa forma, eu escrevo *area(point1, point2)*, substituindo *point1* e *point2* por valores ou variáveis que comportem esses valores, e o Python vai retornar o cálculo de área automaticamente.

Funções Anônimas (Lambda)

As funções anônimas, mais conhecidas simplesmente como `lambda`, são semelhantes às funções que aprendemos na seção anterior. A `lambda` tem como objetivo automatizar uma rotina. Entretanto, você não precisa designar um nome para a função, por isso o nome **função anônima**. Ela é indicada para situações que não requerem a criação de uma função complexa e que não será uma função muito utilizada.

Vamos, primeiramente, criar uma função que soma dois valores, utilizando a metodologia que aprendemos anteriormente:

```
def soma (x, y):
    return x + y
```

Utilizando o método `lambda`, essa função ficaria da seguinte forma:

```
soma = lambda x, y: x + y
```

Note que eu não dei um nome para a função, e sim criei uma variável que se chama soma. Repare, também, que a sintaxe mudou: logo após a palavra `lambda`, eu escrevi os dois valores que vão compor a função e, depois dos dois pontos, os argumentos.

Capítulo 7. Instalando e importando Módulos e Pacotes

Os pacotes em Python são um conjunto de funções pré-existentes e criadas por inúmeras pessoas, que têm como objetivo auxiliar o trabalho do desenvolvedor em Python. A linguagem Python, apesar de ser muito completa, em alguns casos não tem algumas funções nativas, sendo necessário linhas e linhas de códigos para executar uma tarefa. Quando essa tarefa se torna uma rotina e várias pessoas fazem a mesma coisa, cria-se um pacote para auxiliar nessa tarefa.

Vamos fazer uma analogia com o sistema operacional Windows. Pense que o Python é o Windows. Somente com o Windows você consegue fazer algumas coisas básicas, navegar na internet, utilizar o *paint* para fazer alguns desenhos, escutar uma música. Porém, se você desejar ir além, existem os *softwares*, ou como chamamos no Brasil, os programas. A partir dessa necessidade, você instala um programa para trocar mensagens, instala um programa de edição de imagens, outro para trabalho. Esses programas que você instala no Windows, e que complementam sua experiência no Windows, podem ser entendidos como os pacotes que utilizamos em Python. Por fim, os módulos são como pequenas funções dentro dos pacotes.

Pip

Para instalar pacotes em Python, você deve utilizar o pip. O pip é um gerenciador de pacotes em Python, responsável por instalar e remover os pacotes. O processo é simples. Abra seu terminal ou prompt de comando e digite: pip install ‘nome do pacote’. Basta substituir ‘nome do pacote’ pelo nome do pacote que deseja instalar.

No exemplo, a seguir, vou instalar o pacote pandas, um dos mais utilizados pelos cientistas de dados em Python. Para isso, basta digitar pip install pandas, veja:

```
C:\Users\user> pip install pandas
```

Para instalar outros pacotes, basta repetir o mesmo padrão, mudando o nome dos pacotes.

Importando módulos e pacotes em Python

Usamos o termo importar quando desejamos utilizar um pacote. Importar um pacote ou um módulo em Python é equivalente a abrir um programa no Windows, para continuarmos a nossa analogia utilizada na introdução. Para importar um pacote, basta utilizar o comando import “nome do pacote”. Este comando deve ser utilizado antes de você começar a utilizar seu script; seria o equivalente a dizer que você precisa abrir o *Microsoft Paint* antes de começar a desenhar.

Veja, a seguir, um exemplo de importação de pacotes:

```
import pandas
import matplotlib
import numpy
import seaborn
import geopy
```

Por convenção e praticidade, utilizamos *alias* para importar os pacotes. Os *alias* funcionariam como “atalhos” ou “abreviações” para os pacotes. Com os *alias,* não será necessário digitar *matplotlib* toda vez que desejamos chamar uma função do pacote. Para definir *alias,* você deve digitar `as` após o nome do pacote. Veja alguns exemplos:

Você pode definir a *alias* que desejar, entretanto, existem convenções que são adotadas pela comunidade Python. Nesses pacotes do exemplo, foram utilizadas as *alias* mais utilizadas pela comunidade. Recomendo que você siga essas convenções para facilitar o intercâmbio de seu código.

```
import pandas as pd
import matplotlib as plt
import numpy as np
import seaborn as sns
import geopy as gp
```

Por fim, vamos aprender a importar os módulos. Para “chamar” um módulo, basta adicionar a palavra **from** antes do nome do pacote e digitar o resto do comando. Veja o exemplo:

```
from geopy import geocode
from math import pi, sqrt
```

Dessa forma, você pode importar apenas uma ou mais funções daquele módulo, assim como eu fiz no exemplo acima.

Referências

BARRY, P. *Use a Cabeça! Python.* Tradução: Eveline Machado. 2. ed. Rio de Janeiro: Alta Books, 2018.

BEAZLEY, D.; JONES, B. K. *Python Cookbook.* 1. ed. São Paulo: Novatec Editora, 2013.

CHEN, D. Y. *Análise de Dados com Python e Pandas*. 2. ed. São Paulo: Novatec, 2018.

DANJOU, J. *Python Levado a Sério:* Conselhos de um Faixa-preta Sobre Implantação, Escalabilidade, Testes e Outros Assuntos. 1. ed. São Paulo: Novatec Editora, 2020.

DOWNEY, A. B. *Pense em Python:* Pense Como um Cientista da Computação. São Paulo: Novatec Editora, 2016.

GRUS, J. *Data Science do Zero*: Primeiras Regras com o Python. 1. ed. Rio de Janeiro: Editora Alta Books, 2016.

MATTHES, E. *Curso Intensivo de Python:* Uma Introdução Prática e Baseada em Projetos à Programação. 1. ed. São Paulo: Novatec Editora, 2016.

MCKINNEY, W. *Python Para Análise de Dados:* Tratamento de Dados com Pandas, NumPy e IPython. 1. ed. São Paulo: Novatec Editora, 2018.

MENEZES, N. N. C**.** *Introdução à Programação com Python:* Algoritmos e Lógica de Programação Para Iniciantes. 3. ed. São Paulo: Novatec Editora, 2019.

MITCHELL, R. *Web Scraping com Python*: Coletando Mais Dados da web Moderna. 2. ed. São Paulo: Novatec Editora, 2019.

RAMALHO, L. *Python Fluente:* Programação Clara, Concisa e Eficaz. 1. ed. São Paulo: Novatec Editora, 2015.

REITZ, K.; SCHLUSSER, T. *O Guia do Mochileiro Python:* Melhores Práticas Para Desenvolvimento. 1. ed. São Paulo: Novatec Editora, 2017.

RHODES, B.; GOERZEN, J. *Programação de Redes com Python:* Guia Abrangente de Programação e Gerenciamento de Redes com Python 3. 1. ed. São Paulo: Novatec Editora, 2015.

SHAW, Z. A. *Aprenda Python 3 do Jeito Certo:* uma Introdução Muito Simples ao Incrível Mundo dos Computadores e da Codificação. Tradução: Eveline Vieira Machado. 1. ed. Rio de Janeiro: Editora Alta Books, 2019.