



## **Curso Introdutório de Python**

**Grupy-Sanca**

**11 de outubro de 2019**



---

## Sumário

---

<b>Prefácio</b>	<b>1</b>
<b>1 Introdução</b>	<b>3</b>
1.1 O que é <i>Python</i> ? . . . . .	3
1.2 Exemplos . . . . .	4
<b>2 Sobre o grupy-sanca</b>	<b>9</b>
2.1 Atividades . . . . .	9
2.2 Histórico . . . . .	18
<b>3 Guia de Instalação do Python</b>	<b>21</b>
3.1 Linux . . . . .	21
3.2 Mac OS X . . . . .	22
3.3 Windows . . . . .	22
<b>4 Ambientes de Desenvolvimento</b>	<b>23</b>
4.1 Ambientes gráficos . . . . .	23
4.2 Linha de comando . . . . .	28
<b>5 Hello World</b>	<b>29</b>
5.1 C . . . . .	29
5.2 Java . . . . .	29
5.3 Pascal . . . . .	29
5.4 Python . . . . .	30
5.5 Função <code>print()</code> . . . . .	30
<b>6 Python como calculadora</b>	<b>33</b>
6.1 Operadores matemáticos . . . . .	33
6.2 Exercícios . . . . .	35
6.3 Expressões Numéricas . . . . .	35
6.4 Notação Científica . . . . .	36
6.5 Pontos Flutuantes . . . . .	36
6.6 Exercícios . . . . .	37
6.7 Sobre Comentários . . . . .	38
6.8 Comparações . . . . .	38
<b>7 Variáveis</b>	<b>41</b>

7.1	Atribuição . . . . .	41
7.2	Nomes de Variáveis . . . . .	41
7.3	Atribuição múltipla . . . . .	43
7.4	Tipos de objetos . . . . .	44
7.5	Buscando ajuda rapidamente . . . . .	45
7.6	Listando variáveis criadas . . . . .	45
7.7	Exercícios . . . . .	46
<b>8</b>	<b>Strings (sequência de caracteres)</b>	<b>49</b>
8.1	Tamanho . . . . .	50
8.2	Índices . . . . .	51
8.3	Fatias . . . . .	51
8.4	Formatação de strings . . . . .	53
8.5	Separação de <i>Strings</i> . . . . .	54
8.6	Exercícios . . . . .	55
<b>9</b>	<b>Lendo valores do teclado</b>	<b>57</b>
9.1	Exercícios . . . . .	58
<b>10</b>	<b>Listas</b>	<b>59</b>
10.1	Declaração . . . . .	59
10.2	Índices . . . . .	59
10.3	Removendo itens da lista . . . . .	61
10.4	Trabalhando com listas . . . . .	61
10.5	Exercícios . . . . .	63
10.6	Função <code>range()</code> . . . . .	64
<b>11</b>	<b>Dicionários</b>	<b>67</b>
11.1	Declaração . . . . .	67
11.2	Função <code>dict()</code> . . . . .	67
11.3	Chaves . . . . .	68
11.4	Adicionando e removendo elementos . . . . .	69
11.5	Função <code>list()</code> . . . . .	70
11.6	Função <code>len()</code> . . . . .	70
11.7	Método <code>get()</code> . . . . .	70
11.8	Alguns métodos . . . . .	71
11.9	Ordem dos elementos . . . . .	71
11.10	Está no dicionário? . . . . .	72
11.11	Exercícios . . . . .	72
<b>12</b>	<b>Condicionais</b>	<b>73</b>
<b>13</b>	<b>Estruturas de controle</b>	<b>75</b>
13.1	Exercícios . . . . .	76
<b>14</b>	<b>Estruturas de repetição</b>	<b>77</b>
14.1	Exercícios . . . . .	79
<b>15</b>	<b>Funções</b>	<b>81</b>
15.1	Definindo funções . . . . .	81
15.2	Funções com argumentos . . . . .	82
15.3	Exercícios . . . . .	85
<b>16</b>	<b>Exercícios e Desafios!</b>	<b>89</b>
16.1	Calculadora . . . . .	89

16.2	Variáveis . . . . .	90
16.3	Strings . . . . .	92
16.4	Teclado . . . . .	93
16.5	Listas . . . . .	93
16.6	Dicionários . . . . .	94
16.7	Estruturas de Controle . . . . .	94
16.8	Estruturas de repetição . . . . .	95
16.9	Funções . . . . .	95
<b>17</b>	<b>Contribuidores</b>	<b>99</b>



---

## Prefácio

---

O objetivo deste curso é introduzir os conceitos básicos de programação para pessoas sem experiência em desenvolvimento ou iniciantes que não conheçam a linguagem Python.

O recomendado é cada participante ter acesso a um computador durante o curso para fazer os exercícios. O único modo de aprender programação é programando.

A duração estimada para este curso é de cerca de 7 horas, mas esse tempo pode variar dependendo do tamanho da turma e da disponibilidade de café.

Sugerimos que o curso seja realizado:

- em um dia inteiro: com pausas (~20 min) no meio da manhã e da tarde e um intervalo para o almoço; ou
- em três dias seguidos: por cerca de duas ou duas horas e meia em cada dia.

Este trabalho está licenciado sob a Licença Atribuição-NãoComercial-CompartilhaIgual 4.0 Internacional (BY-NC-SA 4.0 internacional) Creative Commons. Para visualizar uma cópia desta licença, visite <http://creativecommons.org/licenses/by-nc-sa/4.0/>



# CAPÍTULO 1

---

## Introdução

---

### 1.1 O que é Python?

Python é uma *linguagem de programação*. Isso significa basicamente duas coisas:

1. existem regras que determinam como as palavras são dispostas, já que é uma *linguagem*;
2. o texto descreve *instruções* para o computador realizar tarefas.

Ou seja, podemos escrever um documento - que chamamos de *código fonte* - em Python para o computador ler e realizar nossos desejos e tarefas. Python tem algumas características interessantes:

- é *interpretada*, ou seja, o interpretador do Python executa o código fonte diretamente, traduzindo cada trecho para instruções de máquina;
- é de *alto nível*, ou seja, o interpretador se vira com detalhes técnicos do computador. Assim, desenvolver um código é mais simples do que em linguagens de *baixo nível*, nas quais o programador deve se preocupar com detalhes da máquina;
- é de *propósito geral*, ou seja, podemos usar Python para desenvolver programas em diversas áreas. Ao contrário de linguagens de domínio específico, que são especializadas e atendem somente a uma aplicação específica;
- tem *tipos dinâmicos*, ou seja, o interpretador faz a magia de descobrir o que é cada variável.

Por essas e várias outras características, Python se torna uma linguagem simples, bela, legível e amigável. É uma linguagem utilizada por diversas empresas, como Wikipedia, Google, Yahoo!, CERN, NASA, Facebook, Amazon, Instagram, Spotify...<sup>1</sup>

O desenvolvimento de Python começou no final da década de 1980, por Guido van Rossum. Ele decidiu usar esse nome porque estava lendo um roteiro de *Monty Python*, um grupo de comédia inglês da década de 1970. A documentação oficial do Python contém muitas referências aos filmes e personagens desse grupo.

Um resumo das versões desta linguagem:

- Versão 1.0 foi publicada em Janeiro de 1994.

---

<sup>1</sup> Lista compilada pela página da [Wikipedia sobre Python](#)<sup>11</sup>, em inglês.  
<sup>11</sup> [https://en.wikipedia.org/wiki/Python\\_\(programming\\_language\)#Uses](https://en.wikipedia.org/wiki/Python_(programming_language)#Uses)

- Versão 2.0 foi publicada em Outubro de 2000.
- Versão 3.0 foi publicada em Dezembro de 2008. 3.7 é a versão mais recente.

## 1.2 Exemplos

Vamos ver alguns exemplos sobre o uso de Python no mundo real.

### 1.2.1 BitTorrent

O protocolo *Torrent* é muito utilizado para transferir quantidades grandes de dados para diversos computadores. O primeiro programa a implementar esse protocolo foi desenvolvido inteiramente em Python, pela *BitTorrent, Inc.*.<sup>2</sup>

### 1.2.2 Django

*Django* é um conjunto de pacotes para desenvolvimento web. É baseado em Python :)

Um objetivo de Django é desenvolver facilmente websites complexos e que lidam com bancos de dados grandões. Alguns sites desenvolvidos em Django: Instagram, The Washington Times, Disqus, Mozilla, National Geographic.<sup>6</sup>

### 1.2.3 Dropbox

O popular serviço de armazenamento de dados em Nuvem *Dropbox* tem diversas partes da infraestrutura feita em Python.<sup>5</sup> O aplicativo para computadores é feito em Python e grande parte da infra estrutura dos servidores deles também é!

### 1.2.4 Estudo sobre erupções solares

Não somente a indústria utiliza Python, muitos pesquisadores utilizam em diversas áreas científicas.

É possível de modo bem simples estudar as erupções solares desde 1992 até hoje. O Observatório Real da Bélgica tem um banco de dados sobre o número de manchas solares, e disponibilizam online para estudos.<sup>4</sup> Veja como é o código para visualizar a atividade solar desde 01/01/1992 em cada parte (norte e sul) do Sol:

```
import pandas as pd      # isso aqui gerencia os dados
import matplotlib.pyplot as plt     # isso aqui permite fazer gráficos

# pega os dados solares de WDC-SILSO, Royal Observatory of Belgium, Brussels
sun = pd.read_table('http://sidc.oma.be/silso/INFO/sndhemcsv.php', sep=';', encoding_= "ISO-8859-1", header=-1)

# faz o gráfico de cada hemisfério do sol
fig = plt.figure()
plt.scatter(sun[3], sun[5], label='Norte', alpha=0.5)
plt.scatter(sun[3], sun[6], label='Sul', alpha=0.5)

plt.title("Atividade solar diária\n")
```

(continues on next page)

<sup>2</sup> [https://en.wikipedia.org/wiki/BitTorrent\\_\(software\)#History](https://en.wikipedia.org/wiki/BitTorrent_(software)#History)

<sup>6</sup> <https://www.djangoproject.com/start/overview/>

<sup>5</sup> [https://en.wikipedia.org/wiki/Dropbox\\_\(service\)#Technology](https://en.wikipedia.org/wiki/Dropbox_(service)#Technology)

<sup>4</sup> <http://sidc.oma.be/silso/home>

(continued from previous page)

```
plt.ylabel('Número de manchas solares')
plt.xlabel('Ano')
plt.legend(loc='upper right')

plt.show()
```

E o resultado desse código é a seguinte imagem:

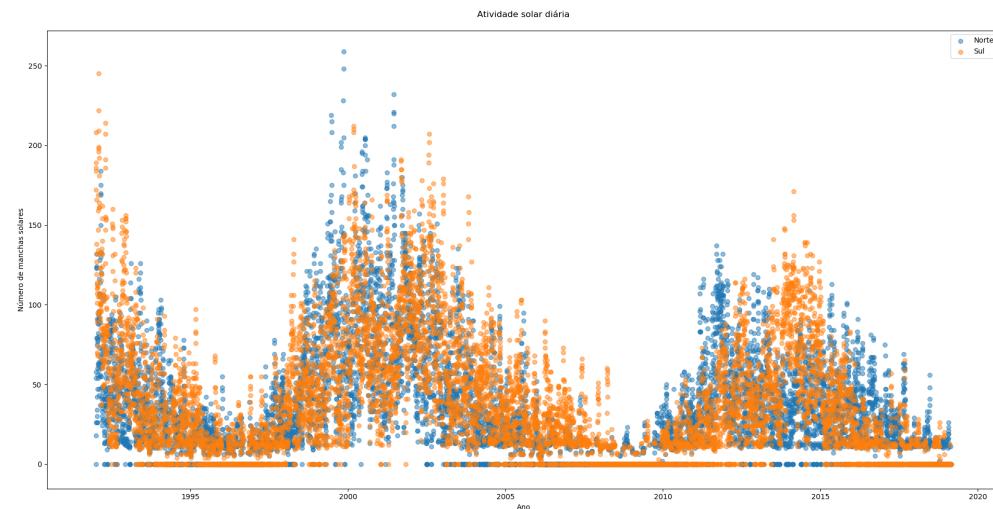


Figura 1: Erupções solares desde 1992, separadas por hemisférios (norte e sul) do Sol.

## 1.2.5 Física de Partículas

O premio Nobel de 2013 em Física foi para os cientistas que estudaram como as partículas elementares adquirem massa, conhecido como *Mecanismo de Higgs*.<sup>7</sup> Uma nova partícula foi descoberta em 2012: o *Bóson de Higgs*. Você pode ler todos os detalhes super técnicos neste artigo [aqui](#)<sup>8</sup> que os cientistas do CERN publicaram.

Caso você esteja interessado apenas na parte computacional, você pode refazer toda a análise dos dados experimentais utilizando Python! Existe uma [apostila online](#)<sup>9</sup> que ensina passo a passo como obter os dados experimentais e simulações teóricas para reproduzir o gráfico a seguir.

## 1.2.6 The Sims 4

O jogo *The Sims 4* tem partes feitas em Python<sup>3</sup>. Isso permite o desenvolvimento de *mods* para o jogo em Python \o/

<sup>7</sup> <https://www.nobelprize.org/prizes/physics/2013/summary/>

<sup>8</sup> <https://inspirehep.net/record/1124338?ln=en>

<sup>9</sup> <https://github.com/cms-opendata-education/cms-jupyter-materials-english/blob/master/Exercises-with-open-data/Advanced/Hunting-the-Higgs-4leptons.ipynb>

<sup>10</sup> <https://github.com/cms-opendata-education/cms-jupyter-materials-english/blob/master/Exercises-with-open-data/Advanced/Hunting-the-Higgs-4leptons.ipynb>

<sup>3</sup> [https://en.wikipedia.org/wiki/The\\_Sims\\_4#Development](https://en.wikipedia.org/wiki/The_Sims_4#Development)

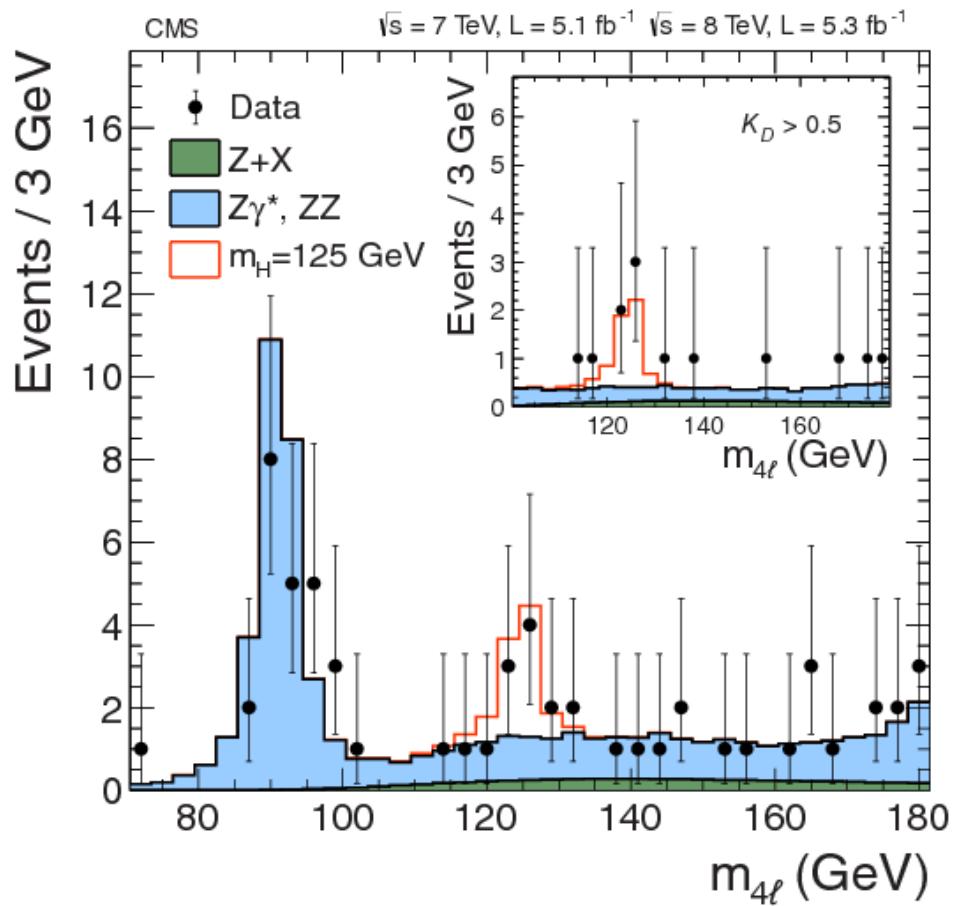


Figura2: Reprodução dos resultados científicos publicados em 2012 sobre a descoberta do Bóson de Higgs. Gráfico obtido com o código publicado na apostila online<sup>10</sup>, que utiliza dados públicos do experimento CMS, no CERN.



Figura3: Capa do jogo The Sims 4, da Electronic Arts.



# CAPÍTULO 2

---

## Sobre o grupy-sanca

---

O *grupy-sanca* (Grupo de Usuários Python de São Carlos) é uma comunidade que reúne pessoas interessadas em desenvolvimento de *software* e na linguagem Python. Prezamos pela troca de conhecimento, respeito mútuo e diversidade (tanto de opinião quanto de tecnologias).

Somos um grupo da cidade de São Carlos (SP) e região. Realizamos periodicamente diversos eventos.

### 2.1 Atividades

#### 2.1.1 Cursos e oficinas de programação

Oferecemos cursos básicos de Python e também sobre alguns assuntos específicos. Durante os cursos os membros do grupy se voluntariam para serem monitores e ajudam a solucionar eventuais dúvidas dos participantes.

Também aceitamos convites para ministrar nossos cursos em eventos, como semanas acadêmicas.

#### 2.1.2 Pylestras

Evento com o objetivo de reunir a comunidade, compartilhar conhecimento e divulgar novas tecnologias. É tradicionalmente composto por palestras rápidas (~17 minutos), onde obrigatoriamente uma delas não é sobre Python.

Qualquer pessoa interessada pode submeter uma palestra, independentemente do tema ser para iniciantes ou usuários avançados. Para deixar a escolha democrática, costumamos usar o Speakerfight<sup>12</sup>, onde a própria comunidade escolhe as palestras que acha mais interessantes.

#### 2.1.3 Coding Dojos

É um espaço para que as pessoas possam aprender, treinar e aprimorar suas habilidades em programação. O *coding dojo* é um ótimo lugar para aprender coisas novas.

---

<sup>12</sup> <https://speakerfight.com/profile/grupysanca/>



Figura1: Nossa primeira curso de Python \o/

Realizado em 25 de março de 2017, no ICMC - USP - São Carlos. Tivemos ~200 inscritos! 81 participantes! 4 ministrantes! 8 monitores!! 2 coffee-breaks! 4 garrafas térmicas: café e chá!



Figura2: Curso de Python básico no IFSC!

Realizado em 01 de Julho de 2017, no IFSC - USP - São Carlos. Tivemos ~100 inscritos! 38 participantes! 2 ministrantes! 3 monitores!! 2 coffee-breaks! 2 garrafas térmicas com apenas café :P



Figura3: Curso de Python básico na UNESP de Rio Claro!!  
Fez parte da programação da SECCOMP 2017, em 23 de Outubro de 2017.

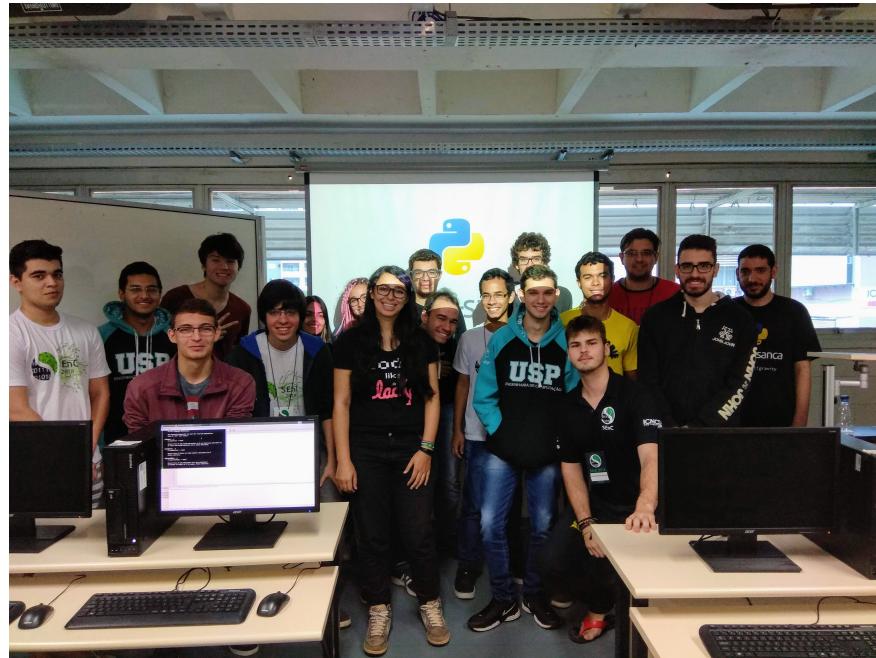
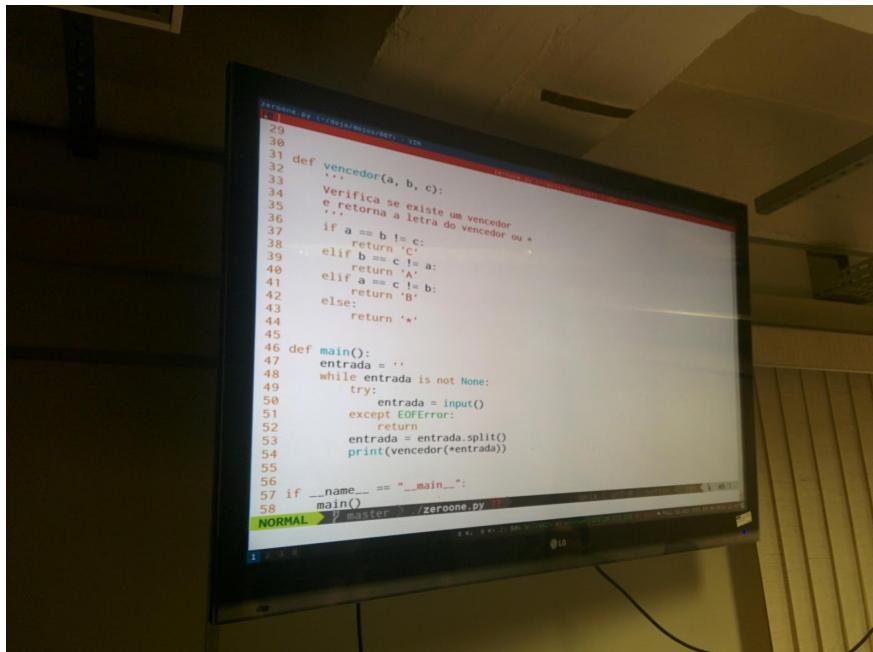


Figura4: Curso de Python básico na USP de São Carlos!!  
Fez parte da programação da SEnC 2018, em 24 de Outubro de 2018.



Em um *coding dojo* são propostos vários desafios e os participantes selecionam quais querem resolver. Após escolherem o desafio, as pessoas leem o problema, entendem e discutem como resolvê-lo. Com esses pontos resolvidos, começa a programação da solução. O desenvolvimento é feito em apenas um computador por duas pessoas programando por vez (*pair programming*). Sendo que a cada intervalo de 5 minutos as pessoas à frente do computador dão lugar para outras.



Para o dojo fluir bem, adota-se o TDD (*Test Driven Development*), pois assim as pessoas pensam melhor em como o código será utilizado antes de desenvolver. Também segue-se o desenvolvimento por *baby steps* em que tenta-se dividir o problema em partes menores para resolvê-lo.

Ao término do dojo acontece uma retrospectiva em que as pessoas respondem três perguntas básicas: *o que foi aprendido?*; *o que pode ser melhorado?* e *o que deve continuar a ser feito?*. Isso serve para os participantes melhorarem nos dojos futuros. Por experiência do grupy-sanca, fazer isso realmente vale a pena :)

## 2.1.4 Eventos

O grupy-sanca também organiza eventos singulares, únicos no universo!

Um dos maiores eventos realizados pela comunidade foi o [Caipyra 2018](#), o único evento de Python com quentão e paçoquinha, que tem por objetivo ser uma conferência de Python voltada ao público do interior do estado de São Paulo.

Após duas edições sediadas em Ribeirão Preto, o grupy-sanca trouxe o evento para São Carlos em 2018 (e já está organizando a edição de 2019).

## 2.1.5 Sprints

Sprints são maratonas de programação. Um grupo de pessoas se junta no mesmo lugar para desenvolver algum projeto novo, resolver *bugs* de algum software, ou implementar alguma funcionalidade nova.

Durante os sprints, usuários iniciantes podem aprender como contribuir com projetos open source, sendo um ambiente ideal para ter contato com pessoas mais experientes e aprender boas práticas.



Figura5: Antes ou depois do coding dojo acontece um coffee break comunitário. Mas já aconteceu de ser durante também...



Figura6: Tivemos 237 participantes, 16 palestras e 3 tutoriais



Figura7: Foram consumidos ~40 litros de quentão e 1625 paçoquinhas



Figura8: Oferecemos um espaço infantil para os papais e mamães poderem participar do evento



Figura9: Tudo isso feito com muito carinho pelos membros do grupy-sanca <3



Figura10: *Hacktoberfest de 2017 :)*  
Fizemos cerca de 50 commits esse dia o/



## 2.1.6 Encontros casuais

Também conhecidos como PyBares :)

A ideia é bater um papo sobre a vida, o universo e tudo mais e tomar uma cerveja (ou não).

A comunidade Python vai muito além de escrever código Python, participar de oficinas/minicursos ou realizar encontros técnicos. Os encontros no bar tem como objetivo conectar pessoas e prover uma conversa descontraída entre os participantes.



E em algumas situações, combinamos de conversar apenas *in english!*



## 2.2 Histórico

O grupo foi fundado em 28/06/2016 e desde então já tivemos:

- 21+ Coding Dojos
- 25+ PyBares

- 16+ Cursos e workshops
- 7 PyLestras
- 4 Eventos
- 3 Sprints

Além disso, chegamos nos 1082+ inscritos no Meetup!

Para saber mais sobre os eventos organizados pelo grupy-sanca acesse:

- Site oficial<sup>13</sup>  
[www.grupysanca.com.br](http://www.grupysanca.com.br)
- Facebook<sup>14</sup>  
[facebook.com/grupysanca](https://facebook.com/grupysanca)
- Instagram<sup>15</sup>  
[instagram.com/grupysanca/](https://instagram.com/grupysanca/)
- Telegram<sup>16</sup>  
[t.me/grupysanca](https://t.me/grupysanca)
- Meetup<sup>17</sup>  
[meetup.com/grupy-sanca](https://meetup.com/grupy-sanca)
- YouTube<sup>18</sup>  
<https://www.youtube.com/channel/UC9AED1x6Nn10lu-3rNELQnw>

Ou entre em contato através do nosso email: contato @ grupysanca . com . br

---

<sup>13</sup> <http://www.grupysanca.com.br>

<sup>14</sup> <https://facebook.com/grupysanca>

<sup>15</sup> <https://instagram.com/grupysanca/>

<sup>16</sup> <https://t.me/grupysanca>

<sup>17</sup> <https://meetup.com/grupy-sanca>

<sup>18</sup> <https://www.youtube.com/channel/UC9AED1x6Nn10lu-3rNELQnw>



# CAPÍTULO 3

---

## Guia de Instalação do Python

---

### 3.1 Linux

Provavelmente você já tem o Python instalado e configurado. Para ter certeza que ele está instalado e descobrir qual versão, abra um terminal e execute o comando:

```
$ python --version
```

Se o resultado do comando for *Python 3.6.5* (ou alguma versão igual ou superior a 3.5) o Python já está instalado corretamente.

Caso o resultado do comando anterior tenha sido *Python 2.7.13* (ou qualquer versão do *Python 2*) tente rodar o seguinte comando, pois seu computador pode ter ambas versões 2 e 3 instaladas:

```
$ python3 --version
```

Caso tenha aparecido a mensagem `bash: python: command not found`, você pode instalá-lo da seguinte maneira:

#### 3.1.1 No Ubuntu e Debian

```
$ sudo apt install python3
```

#### 3.1.2 No ArchLinux

```
$ sudo pacman -Sy python
```

## 3.2 Mac OS X

Obtenha o instalador na sessão de downloads para Mac OS X do Python<sup>19</sup>. Clique duas vezes no Python.mpkg para abrir o instalador.

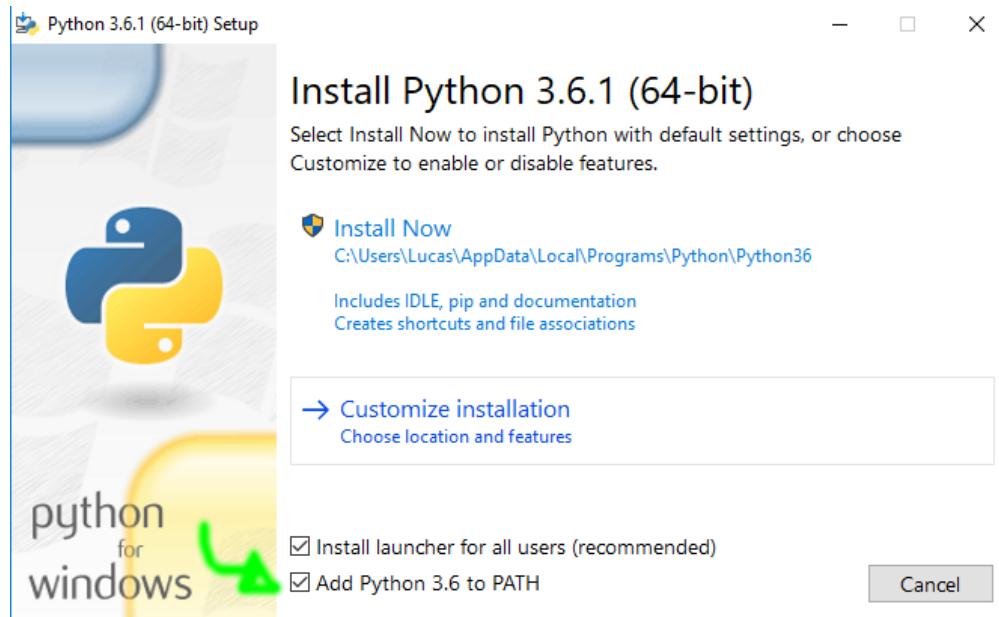
Para ter certeza que ele está instalado e descobrir qual versão, abra um terminal e execute o comando:

```
$ python --version  
Python 3.6.5
```

## 3.3 Windows

Obtenha o arquivo de instalação última versão compatível com a arquitetura do seu computador no site oficial do Python<sup>20</sup>. O arquivo x86 provavelmente funcionará para todos computadores.

A seguir, execute o instalador e uma imagem similar a essa aparecerá:



Deve ser selecionada a opção Add Python 3.6 to PATH e depois continuar a instalação até o fim.

---

<sup>19</sup> <https://www.python.org/downloads/>

<sup>20</sup> <https://www.python.org/downloads/>

# CAPÍTULO 4

---

## Ambientes de Desenvolvimento

---

Há diversos programas para desenvolvermos códigos, alguns são mais bonitinhos, outros são mais poderosos, alguns são mais simples, outros são mais amigáveis. Dê uma olhada nesta seção e escolha o que você achar mais interessante. Somente você pode responder à pergunta «Qual o melhor ambiente de desenvolvimento para *mim*?»

IDE (*Integrated Development Environment* - Ambiente de Desenvolvimento Integrado, em português) é um editor de texto que possui ferramentas e recursos que facilitam a vida do programador. Entre as ferramentas e recursos, podemos citar:

- Identificar quais variáveis foram declaradas.
- Identificar erros no código.
- Personalizar o ambiente de trabalho.
- Ocultar parte do código para melhor visualização.

### 4.1 Ambientes gráficos

#### 4.1.1 ATOM

O programa ATOM é um IDE *open-source* que apresenta diversos pacotes para personalizar.

No site oficial do ATOM<sup>21</sup>, você encontrará um link para a Documentação do programa. Na documentação, é possível acessar o manual<sup>22</sup> que mostrará passo a passo como instalar o programa (tanto para Windows como para Linux).

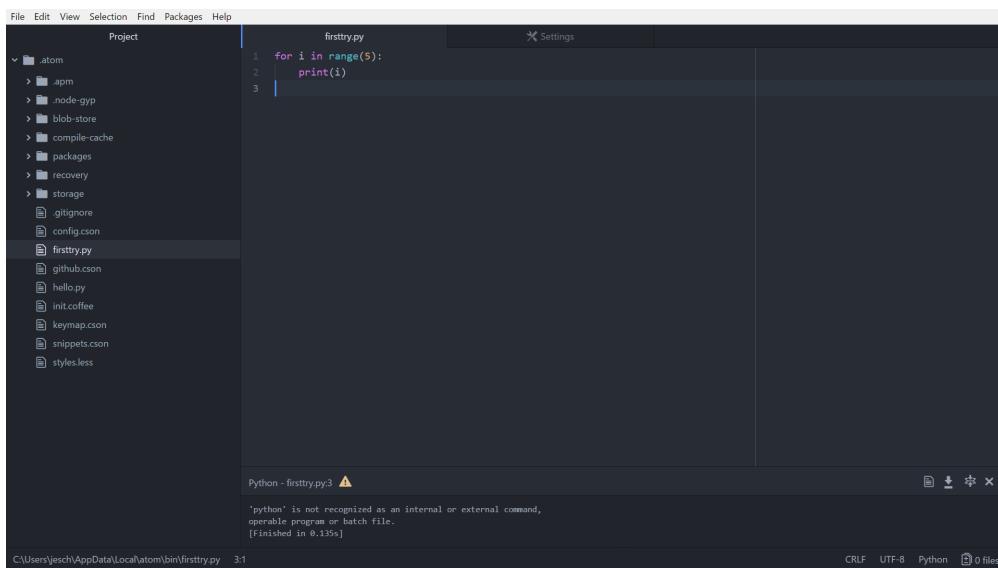
Caso você tenha alguma dúvida, é aconselhável entrar na seção de discussão<sup>23</sup>. Nessa página, você encontrará respostas para diversas dúvidas, e possivelmente, para a sua.

---

<sup>21</sup> <https://atom.io>

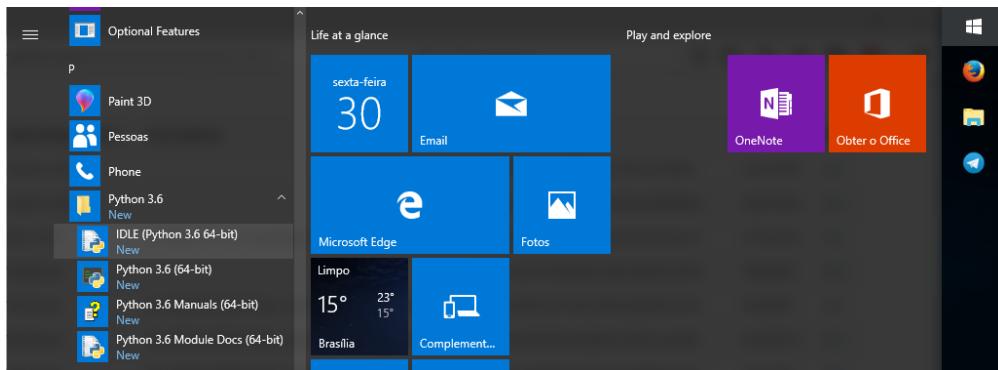
<sup>22</sup> <https://flight-manual.atom.io/getting-started/sections/installing-atom/#platform-windows>

<sup>23</sup> <https://discuss.atom.io/>

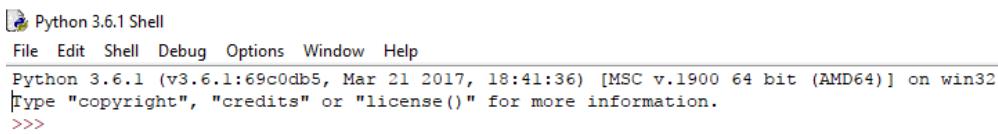


## 4.1.2 IDLE

Para usuários Windows é recomendado utilizar o *IDLE*. Ele é composto pelo interpretador do Python e um editor de texto para criar programas, e já vem junto com o Python. Após seguir o [Guia de Instalação do Python](#) (Página 21), o menu inicial deve estar da seguinte forma:



Ao abrir o *IDLE (Python 3.X)*, aparecerá uma janela como na imagem abaixo:

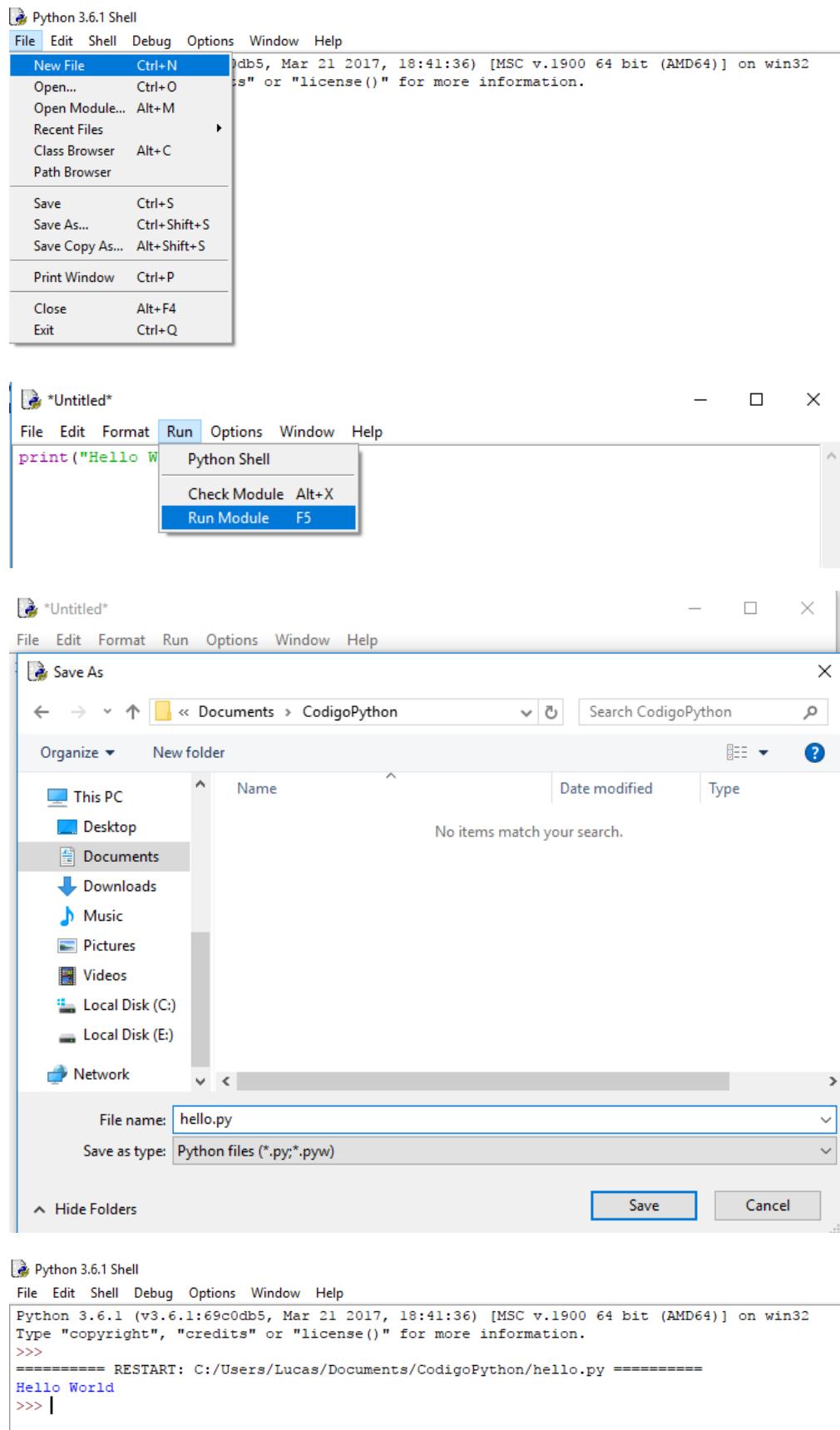


No *IDLE* é possível digitar comandos para o interpretador do Python e, também, é possível criar e digitar em um arquivo. Para fazer isso, no menu clique em *File -> New File* (Ou pressione as teclas *Ctrl + N* juntas)

Para rodar um programa, clique em *Run -> Run Module* (Ou aperte a tecla *F5*)

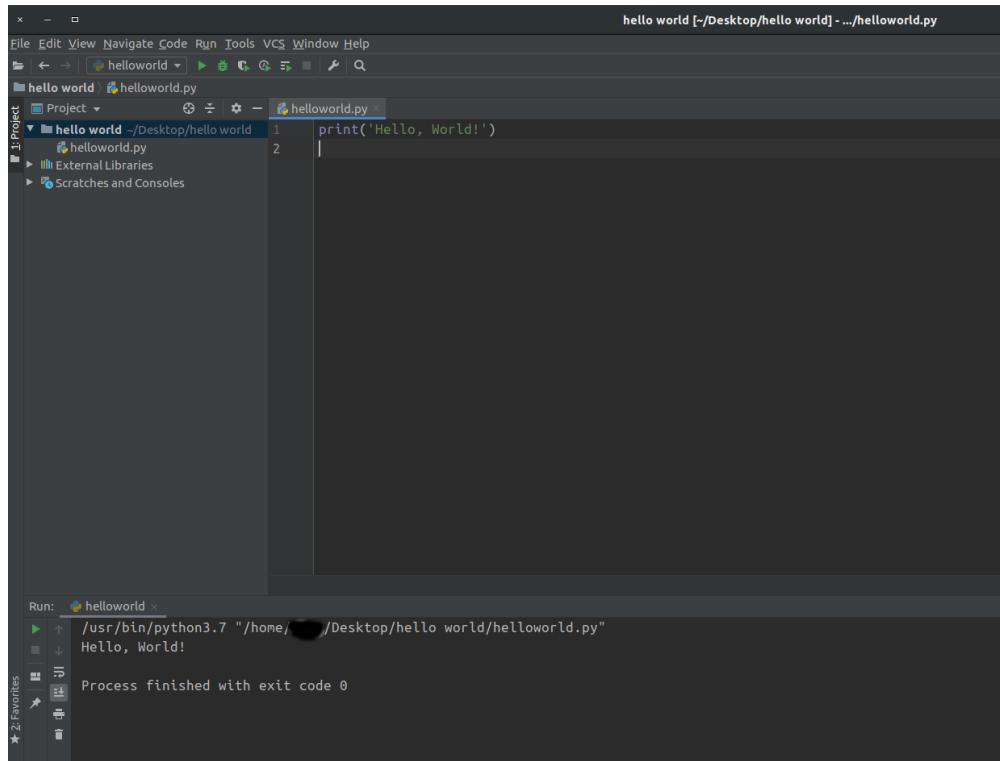
Caso o arquivo ainda não tenha sido salvo, é necessário salvá-lo antes de executá-lo. Não esqueça de prefixar o nome do arquivo com *.py* (extensão do Python):

Após isso, o resultado da execução do código deve aparecer na janela anterior do *IDLE*:



### 4.1.3 PyCharm

Esta IDE é voltada especificamente para a linguagem Python. No site oficial<sup>24</sup> é possível encontrar orientações para realizar o download e instalação (Linux, Mac, Windows).



É desenvolvido pela empresa tcheca JetBrains. Fornece análise de código, um depurador gráfico, teste de unidade integrado, integração com sistemas de controle de versão, ambiente virtual e suporta o desenvolvimento da Web com o Django, bem como Data Science com o Anaconda ([Wikipedia<sup>25</sup>](#)).

### 4.1.4 Spyder

O Spyder é uma IDE perfeita para quem utiliza o Python para Ciência de Dados, especialmente porque vem incluso no *Anaconda*, um pacote de várias bibliotecas e ferramentas voltadas para essa área.

Ele possui uma interface simples, com uma tabela de variáveis para fácil inspeção dos valores de cada variável no programa durante sua depuração.

A melhor forma de instalá-lo é pela instalação do [Anaconda<sup>26</sup>](#). Novamente, recomenda-se utilizar essa forma de instalação apenas para quem deseja utilizar todos os pacotes inclusos no Anaconda, e não apenas o Spyder.

### 4.1.5 Visual Studio Code

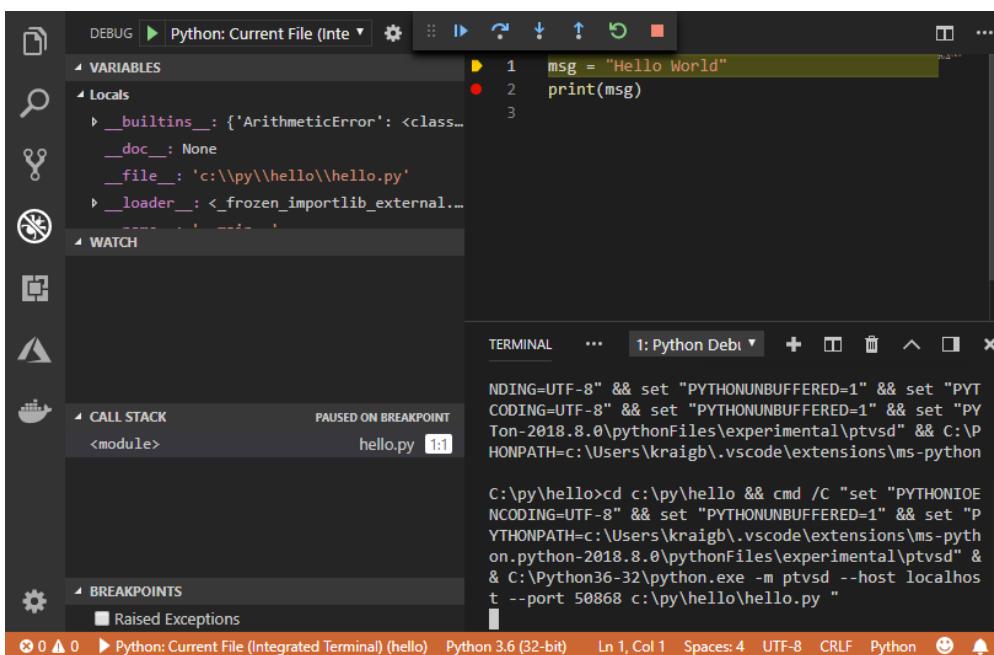
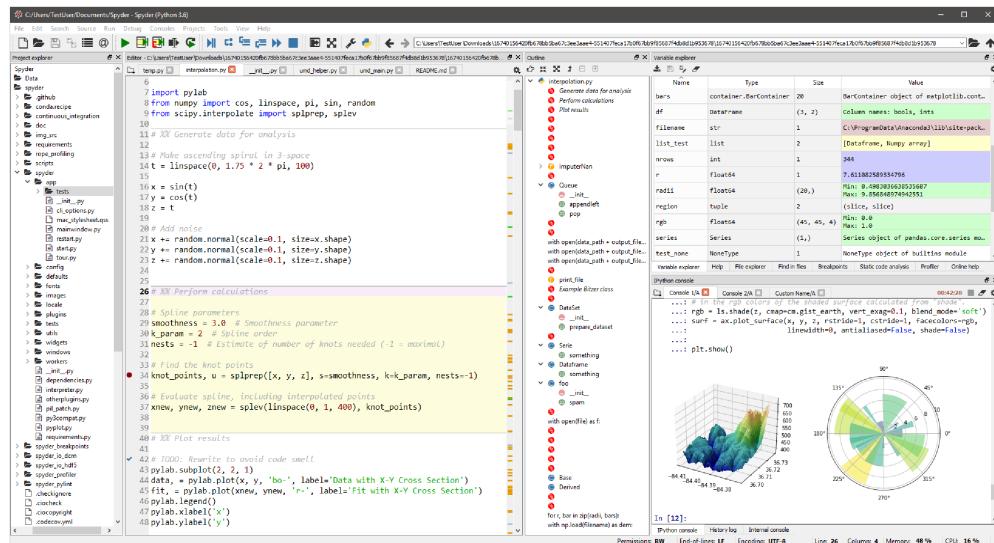
O Visual Studio Code por si só é um editor de texto criado pela Microsoft que apresenta diversos pacotes para personalizá-lo da forma que você precisa.

---

<sup>24</sup> <https://www.jetbrains.com/pycharm/>

<sup>25</sup> <https://en.wikipedia.org/wiki/PyCharm>

<sup>26</sup> <https://www.anaconda.com/distribution/>



No site oficial do [Visual Studio Code](#)<sup>27</sup>, você encontrará um link para a Documentação do programa.

Na documentação, é possível acessar o [manual de Python](#)<sup>28</sup> que mostra todos as funcionalidades que o programa possui relacionadas a Python.

Para um passo-a-passo da instalação, você pode encontrar os detalhes de cada sistema operacional [nesta página do manual](#)<sup>29</sup>.

## 4.2 Linha de comando

Existe também a possibilidade de trabalhar sem um ambiente gráfico, utilizando apenas a *interface de linha de comando*.

### 4.2.1 Python Shell

Se você instalou corretamente o Python, você tem à sua disposição um interpretador interativo, popularmente conhecido como *o Shell do Python*.

Se você utiliza Linux ou Mac, basta abrir um terminal e digitar `python`. Caso utilize Windows, deverá existir uma pasta no menu *iniciar* chamada Python 3.7 (ou Python 3.6) que permite abrir o interpretador. Caso tenha sucesso, você deverá encontrar algo similar a esta imagem:

```
$ python
Python 3.7.2 (default, Jan 10 2019, 23:51:51)
[GCC 8.2.1 20181127] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> □
```

Para sair do Python, digite `quit()`. Isso encerra a seção interativa e te retorna ao terminal (ou fecha a tela preta, no caso do Windows).

Note que, ao entrar no interpretador, a primeira linha começa com Python 3.7.2. Essa é a versão utilizada do Python. Neste curso utilizamos a versão 3.6 ou mais recente. Caso a versão que aparecer em seu interpretador seja 2.7.6, encerre a seção e tente o comando `python3` para utilizar a versão 3 do Python.

### 4.2.2 IPython

IPython<sup>30</sup> é um Shell alternativo para o Python, bem poderoso. E colorido também, ao contrário do *shell* nativo do Python.

```
$ ipython
Python 3.7.2 (default, Jan 10 2019, 23:51:51)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.3.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: □
```

---

<sup>27</sup> <https://code.visualstudio.com/>

<sup>28</sup> <https://code.visualstudio.com/docs/languages/python>

<sup>29</sup> <https://code.visualstudio.com/docs/setup/setup-overview>

<sup>30</sup> <https://ipython.org>

# CAPÍTULO 5

---

## Hello World

---

É muito comum, ao apresentar uma nova linguagem, começar com um exemplo simples que mostra na tela as palavras *Hello World*. Para não perder o costume, antes de adentrar o mundo do Python, vamos ver como outras linguagens de programação implementam esse exemplo:

### 5.1 C

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("Hello, World!\n");
    return 0;
}
```

### 5.2 Java

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

É obrigatório que o código acima esteja em um arquivo chamado *Hello.java*

### 5.3 Pascal

```
program HelloWorld;

begin
    writeln('Hello, World!');
end.
```

## 5.4 Python

Vamos ver como é o *Hello World* em Python. Para isso, abra o *shell* do Python e digite o texto abaixo (não esqueça de apertar *enter* no final):

```
>>> print("Hello, World!")
Hello, World!
```

Em programação, é muito comum utilizar a palavra *imprimir* (ou *print*, em inglês) como sinônimo de mostrar algo na tela.

## 5.5 Função `print()`

`print()` é uma função nativa do Python. Basta colocar algo dentro dos parênteses que o Python se encarrega de fazer a magia de escrever na tela :)

### 5.5.1 Erros comuns

Usar a letra *P* maiúscula ao invés de minúscula:

```
>>> Print("Hello, World!")
Traceback (most recent call last):
...
NameError: name 'Print' is not defined
```

Esquecer de abrir e fechar aspas no texto que é passado para a função `print()`:

```
>>> print(Hello, World!)
Traceback (most recent call last):
...
SyntaxError: invalid syntax
```

Esquecer de abrir ou fechar as aspas:

```
>>> print("Hello, World!
Traceback (most recent call last):
...
SyntaxError: EOL while scanning string literal
```

Começar com aspas simples e terminar com aspas duplas ou vice-versa:

```
>>> print('Hello, World")
Traceback (most recent call last):
...
SyntaxError: EOL while scanning string literal
```

Usar espaço ou tabulação (tab) antes do print ():

```
>>> print('Hello, World!')
Traceback (most recent call last):
...
IndentationError: unexpected indent

>>>     print('Hello, World!')
Traceback (most recent call last):
...
IndentationError: unexpected indent
```

Mas, e se eu precisar usar aspas dentro do texto a ser mostrado na tela? Bem, Caso queira imprimir aspas duplas, envolva tudo com aspas simples e use aspas duplas na parte desejada:

```
>>> print('Python é legal! Mas não o "legal" como dizem pra outras coisas')
Python é legal! Mas não o "legal" como dizem pra outras coisas
```

Caso deseje imprimir aspas simples, faça o contrário (envolva com aspas duplas e use aspas simples onde necessário):

```
>>> print("Python é legal! Mas não o 'legal' como dizem pra outras coisas")
Python é legal! Mas não o 'legal' como dizem pra outras coisas
```

E como faz para imprimir um texto em várias linhas? Bom, para isso precisamos lembrar de um caractere especial, a *quebra de linha*: *n*. Esse *n* é um caractere especial que significa *aqui acaba a linha, o que vier depois deve ficar na linha de baixo*. Por exemplo:

```
>>> print('Olha esse textão sobre aspas simples e dúplas.\nIsso aqui é aspas duplas:
  ↪"\nIsso aqui é aspas simples: \''
Olha esse textão sobre aspas simples e dúplas.
Isso aqui é aspas duplas: "
Isso aqui é aspas simples: '
```



# CAPÍTULO 6

---

## Python como calculadora

---

### 6.1 Operadores matemáticos

A linguagem Python possui operadores que utilizam símbolos especiais para representar operações de cálculos, assim como na matemática:

- Soma (+)

```
>>> 2 + 3  
5
```

Para utilizar números decimais, use o *ponto* no lugar de vírgula:

```
>>> 3.2 + 2.7  
5.9
```

- Subtração (-)

```
>>> 6 - 4  
2
```

```
>>> 7 - 8  
-1
```

- Multiplicação (\*)

```
>>> 7 * 8  
56
```

```
>>> 2 * 2 * 2  
8
```

- Divisão (/)

```
>>> 100 / 20  
5.0
```

```
>>> 10 / 3  
3.3333333333333335
```

E se fizermos uma divisão por zero?

```
>>> 2 / 0  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
ZeroDivisionError: division by zero
```

Como não existe um resultado para a divisão pelo número zero, o Python interrompe a execução do programa (no caso a divisão) e mostra o erro que aconteceu, ou seja, «*ZeroDivisionError: division by zero*».

- Divisão inteira (//)

```
>>> 10 // 3  
3  
>>> 666 // 137  
4  
>>> 666 / 137  
4.861313868613139
```

- Resto da divisão (%)

```
>>> 10 % 2  
0  
>>> 10 % 3  
1  
>>> 666 % 137  
118
```

Agora que aprendemos os operadores aritméticos básicos podemos seguir adiante. Como podemos calcular  $2^{10}$ ? O jeito mais óbvio seria multiplicar o número dois dez vezes:

```
>>> 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2  
1024
```

Porém, isso não é muito prático, pois há um operador específico para isso, chamado de *potenciação/exponenciação*: \*\*

```
>>> 2 ** 10  
1024
```

```
>>> 10 ** 3  
1000
```

```
>>> (10 ** 800 + 9 ** 1000) * 233  
40725400165137782505077408626536591293327155957239892465016990675188990003095518900491634747847069888
```

E a *raiz quadrada*?

Lembrando que  $\sqrt{x} = x^{\frac{1}{2}}$ , então podemos calcular a raiz quadrada do seguinte modo:

```
>>> 4 ** 0.5
2.0
```

Mas a maneira recomendada para fazer isso é usar a função `sqrt()` da biblioteca `math`:

```
>>> import math
>>> math.sqrt(16)
4.0
```

Na primeira linha do exemplo importamos, da biblioteca padrão do Python, o módulo `math` e então usamos a sua função `sqrt` para calcular  $\sqrt{16}$

E se precisarmos utilizar o número  $\pi$ ?

```
>>> math.pi
3.141592653589793
```

Não esqueça que é preciso ter executado `import math` antes de usar as funções e constantes dessa biblioteca.

## 6.2 Exercícios

1. Calcule o resto da divisão de 10 por 3.
2. Calcule a tabuada do 13.
3. Davinir não gosta de ir às aulas. Mas ele é obrigado a comparecer a pelo menos 75% delas. Ele quer saber quantas aulas pode faltar, sabendo que tem duas aulas por semana, durante quatro meses. Ajude o Davinir!

---

**Nota:** Um mês tem quatro semanas.

---

4. Calcule a área de um círculo de raio  $r = 2$ .

Lembrete: a área de um círculo de raio  $r$  é:

$$A_{\circ} = \pi r^2$$

## 6.3 Expressões Numéricas

Agora que já aprendemos diversos operadores, podemos combiná-los e resolver problemas mais complexos:

```
>>> 3 + 4 * 2
11
```

```
>>> 7 + 3 * 6 - 4 ** 2
9
```

```
>>> (3 + 4) * 2
14
```

```
>>> (8 / 4) ** (5 - 2)
8.0
```

Quando mais de um operador aparece em uma expressão, a ordem de avaliação depende das regras de precedência. O Python segue as mesmas regras de precedência da matemática. O acrônimo **PEMDAS** ajuda a lembrar essa ordem:

1. Parênteses
2. Exponenciação
3. Multiplicação e Divisão (mesma precedência)
4. Adição e Subtração (mesma precedência)

## 6.4 Notação Científica

Notação científica em Python usa a letra  $e$  como sendo a potência de 10:

```
>>> 10e6  
10000000.0  
>>> 1e6  
1000000.0  
>>> 1e-5  
1e-05
```

Também pode ser usada a letra E maiúscula:

```
>>> 1E6  
1000000.0
```

## 6.5 Pontos Flutuantes

Uma consideração importante sobre pontos flutuantes (números decimais). Por exemplo:

```
>>> 0.1  
0.1
```

É importante perceber que este valor, em um sentido real na máquina, não é exatamente  $1/10$ . Está arredondando a exibição do valor real da máquina.

```
>>> format(0.1, '.50f')  
'0.1000000000000000555111512312578270211815834045410'
```

Veja que somente após a 18ª casa que há diferença. Isso é mais dígitos do que a maioria das pessoas acham úteis, então o Python mantém o número de dígitos gerenciáveis exibindo um valor arredondado

Este fato se torna aparente assim que você tenta fazer aritmética com esses valores

```
>>> 0.1 + 0.2  
0.30000000000000004  
>>> 0.7 - 0.2  
0.4999999999999994
```

Note que isso é da mesma natureza do ponto flutuante binário, não é um bug no Python e muito menos um bug no seu código. Você verá o mesmo tipo de coisa em todos os idiomas que suportam a aritmética de ponto flutuante de seu hardware (embora alguns idiomas possam não exibir a diferença por padrão ou em todos os modos de saída).

Os erros de representação referem-se ao fato de que a maioria das frações decimais não podem ser representadas exatamente como frações binárias (base 2). Essa é a principal razão pela qual o Python (ou Perl, C, C++, Java, Fortran e muitos outros) geralmente não exibe o número decimal exato que é esperado.

O valor de 1/10 não é exatamente representável como uma fração binária. Quase todas as máquinas atualmente (considerando após novembro de 2000) usam aritmética de ponto flutuante IEEE-754<sup>31</sup>, e quase todas as plataformas mapeiam pontos flutuantes do Python para a «*dupla precisão*» IEEE-754, que contêm 53 bits de precisão. Portanto, na entrada, o computador se esforça para converter 0.1 na fração mais próxima possível da forma  $J/2^{N}$ , onde  $J$  é um inteiro contendo exatamente 53 bits.

## 6.6 Exercícios

1. Quantos segundos há em 3 horas, 23 minutos e 17 segundos?
2. Se você correr 65 quilômetros em 3 horas, 23 minutos e 17 segundos, qual é a sua velocidade média em m/s?
3. Resolva essa expressão:

$$\frac{100 - 413 \cdot (20 - 5 \times 4)}{5}$$

4. Enivaldo quer ligar três capacitores, de valores:

- $C_1 = 10 \mu F$
- $C_2 = 22 \mu F$
- $C_3 = 6.8 \mu F$

Se ele ligar os três em paralelo, a capacidade resultante é a soma:

$$C_p = C_1 + C_2 + C_3$$

Se ele ligar os três em série, a capacidade resultante é:

$$\frac{1}{C_s} = \frac{1}{C_1} + \frac{1}{C_2} + \frac{1}{C_3}$$

Ou seja:

$$C_s = \frac{1}{\frac{1}{C_1} + \frac{1}{C_2} + \frac{1}{C_3}}$$

Qual é o valor resultante em cada um desses casos?

5. Você e os outros integrantes da sua república (Joca, Moacir, Demival e Jackson) foram no supermercado e compraram alguns itens:

- 75 latas de cerveja: R\$ 2,20 cada (da ruim ainda, pra fazer o dinheiro render)
- 2 pacotes de macarrão: R\$ 8,73 cada
- 1 pacote de Molho de tomate: R\$ 3,45
- 420g Cebola: R\$ 5,40/kg
- 250g de Alho: R\$ 30/kg
- 450g de pães franceses: R\$ 25/kg

---

<sup>31</sup> [https://pt.wikipedia.org/wiki/IEEE\\_754](https://pt.wikipedia.org/wiki/IEEE_754)

Calcule quanto ficou para cada um.

6. Krissia gosta de bolinhas de queijo. Ela quer saber quantas bolinhas de queijo dá para colocar dentro de um pote de sorvete de  $2\text{ L}$ . Ela pensou assim:

*Um pote de sorvete tem dimensões  $15\text{ cm} \times 10\text{ cm} \times 13\text{ cm}$ . Uma bolinha de queijo é uma esfera de raio  $r = 1.2\text{ cm}$ . O fator de empacotamento ideal é 0.74, mas o pote de sorvete tem tamanho comparável às bolinhas de queijo, aí tem efeitos de borda, então o fator deve ser menor. Mas as bolinhas de queijo são razoavelmente elásticas, então empacota mais. Esse valor parece razoável.*

Sabendo que o volume de uma esfera de raio  $r$  é  $V = \frac{4}{3}\pi r^3$ , o volume do pote de sorvete é  $V = x \cdot y \cdot z$  e o fator de empacotamento é a fração de volume ocupado pelas bolinhas de queijo. Ou seja, 74% do pote de sorvete vai ser ocupado pelas bolinhas de queijo.

Ajude a Krissia descobrir quantas bolinhas de queijo cabem no pote de sorvete!

## 6.7 Sobre Comentários

Caso precise explicar alguma coisa feita no código, é possível escrever um texto (que não será executado), que ajuda a entender ou lembrar o que foi feito. Esse texto é chamado de comentário, e para escrever um basta utilizar o caractere `#`. Exemplo:

```
>>> 3 + 4 # será lido apenas o cálculo, do # para frente o interpretador do Python  
→irá ignorar!
```

7

```
>>> # Aqui vai um código só com comentários! Posso falar o que quiser que não será  
→interpretado, lalala, la-le-li-lo-lu. A job we hate to buy things we don't need.
```

## 6.8 Comparações

Os operadores de comparação em Python são:

Operação	Significado
<code>&lt;</code>	menor que
<code>&lt;=</code>	menor igual que
<code>&gt;</code>	maior que
<code>&gt;=</code>	maior igual que
<code>==</code>	igual
<code>!=</code>	diferente

```
>>> 2 < 10  
True  
>>> 2 > 11  
False  
>>> 10 > 10  
False  
>>> 10 >= 10  
True  
>>> 42 == 24  
False
```

(continues on next page)

(continued from previous page)

```
>>> 666 != 137
True
>>> 8**2 == 60 + 4
True
>>> 100 != 99 + 3
True
```



# CAPÍTULO 7

## Variáveis

Variável é um *nome* que se refere a um *valor*.

### 7.1 Atribuição

Atribuição é o processo de criar uma nova variável e dar um novo valor a ela. Alguns exemplos de atribuições:

```
>>> numero = 11  
>>> numero  
11
```

```
>>> frase = "Me dá um copo d'água"  
>>> frase  
"Me dá um copo d'água"
```

```
>>> pi = 3.141592  
>>> pi  
3.141592
```

No exemplo anterior realizamos três atribuições. No primeiro atribuímos um número inteiro à variável de nome `numero`; no segundo uma frase à variável `frase`; no último um número de ponto flutuante à `pi`.

### 7.2 Nomes de Variáveis

Bons programadores escolhem nomes significativos para as suas variáveis - eles documentam o propósito da variável.

Nomes de variáveis podem ter o tamanho que você achar necessário e podem conter tanto letras como números, porém não podem começar com números. É possível usar letras maiúsculas, porém a convenção é utilizar somente letras minúsculas para nomes de variáveis.

```
>>> crieiumavaravelcomnomegiganteestoucompreguiçadeescrevertudodenovo = 10
>>> crieiumavaravelcomnomegiganteestoucompreguiçadeescrevertudodenovo # use TAB
  ↵para autocompletar =D
10
```

Tentar dar um nome ilegal a uma variável ocasionará erro de sintaxe:

```
>>> 123voa = 10
Traceback (most recent call last):
...
    123voa = 10
          ^
SyntaxError: invalid syntax
```

```
>>> ol@ = "oi"
Traceback (most recent call last):
...
    ol@ = "oi"
          ^
SyntaxError: invalid syntax
```

```
>>> def = 2.7
Traceback (most recent call last):
...
    def = 2.7
          ^
SyntaxError: invalid syntax
```

123voa é ilegal pois começa com um número. ol@ é ilegal pois contém um caractere inválido (@), mas o que há de errado com def?

A questão é que def é uma palavra-chave da linguagem. O Python possui diversas palavras que são utilizadas na estrutura dos programas, por isso não podem ser utilizadas como nomes de variáveis.

Outro ponto importante: não é possível acessar variáveis que ainda não foram definidas:

```
>>> nao_definida
Traceback (most recent call last):
...
NameError: name 'nao_definida' is not defined
```

Tentar acessar uma variável sem definí-la anteriormente ocasiona em um «erro de nome».

Também podemos atribuir expressões a uma variável:

```
>>> x = 3 * 5 - 2
>>> x
13
>>> y = 3 * x + 10
>>> y
49
>>> z = x + y
>>> z
62
```

```
>>> n = 10
>>> n + 2 # 10 + 2
```

(continues on next page)

(continued from previous page)

```
12
>>> 9 - n # 9 - 10
-1
```

É importante lembrar que para mudar o valor de uma variável é preciso utilizar a atribuição. Nos dois exemplos anteriores não atribuímos as expressões à n, portanto seu valor continuou o mesmo.

Vamos alterar o valor de n:

```
>>> n
10
>>> n = n + 2
>>> n
12
>>> 9 - n
-3
```

Outra forma de somar na variável:

```
>>> num = 4
>>> num += 3
>>> num
7
```

Também funciona com multiplicação:

```
>>> x = 2
>>> x *= 3
>>> x
6
```

## 7.3 Atribuição múltipla

Uma funcionalidade interessante do Python é que ele permite atribuição múltipla. Isso é muito útil para trocar o valor de duas variáveis:

```
>>> a = 1
>>> b = 200
```

Para fazer essa troca em outras linguagens é necessário utilizar uma variável auxiliar para não perdemos um dos valores que queremos trocar. Vamos começar da maneira mais simples:

```
>>> a = b # perdemos o valor de a
>>> a
200
```

```
>>> b = a # como perdemos o valor de a, b vai continuar com seu valor original de 200
>>> b
200
```

A troca é bem sucedida se usamos uma variável auxiliar:

```
>>> a = 1
>>> b = 200
>>> print(a, b)
1 200

>>> aux = a
>>> a = b
>>> b = aux
>>> print(a, b)
200 1
```

Porém, como o Python permite atribuição múltipla, podemos resolver esse problema de uma forma muito mais simples:

```
>>> a = 1
>>> b = 200
>>> print(a, b)
1 200
```

```
>>> a, b = b, a
>>> print(a, b)
200 1
```

A atribuição múltipla também pode ser utilizada para simplificar a atribuição de variáveis, por exemplo:

```
>>> a, b = 1, 200
>>> print(a, b)
1 200
```

```
>>> a, b, c, d = 1, 2, 3, 4
>>> print(a, b, c, d)
1 2 3 4
```

```
>>> a, b, c, d = d, c, b, a
>>> print(a, b, c, d)
4 3 2 1
```

## 7.4 Tipos de objetos

Criamos muitas variáveis até agora. Você lembra o tipo de cada uma? Para saber o tipo de um objeto ou variável, usamos a função `type()`:

```
>>> x = 1
>>> type(x)
<class 'int'>
>>> y = 2.3
>>> type(y)
<class 'float'>
>>> type('Python')
<class 'str'>
>>> type(True)
<class 'bool'>
```

Python vem com alguns tipos básicos de objetos, dentre eles:

- `bool`: verdadeiro ou falso.

- `int`: números inteiros.
- `float`: números reais.
- `complex`: números complexos.
- `str`: *strings* (textos).
- `list`: *listas* (Página 59). Estudaremos em breve o que são.
- `dict`: *dicionários* (Página 67). Estudaremos em breve o que são.

## 7.5 Buscando ajuda rapidamente

Está com dúvida em alguma coisa? Use a função `help()` e depois digite o que você busca.

```
>>> help()

Welcome to Python 3.6's help utility!

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at http://docs.python.org/3.6/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules. To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics". Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".

help>
You are now leaving help and returning to the Python interpreter.
If you want to ask for help on a particular object directly from the
interpreter, you can type "help(object)". Executing "help('string')"
has the same effect as typing a particular string at the help> prompt.
```

E para buscar ajuda em uma coisa específica?

```
>>> help(len)
Help on built-in function len in module builtins:

len(obj, /)
    Return the number of items in a container.
```

Para sair do ambiente de ajuda, pressione a tecla `q`.

A [documentação oficial<sup>32</sup>](#) do Python contém toda a referência sobre a linguagem, detalhes sobre cada função e alguns exemplos (em inglês).

## 7.6 Listando variáveis criadas

Em algum momento durante o seu código você pode querer saber quais variáveis já foram declaradas, ou até mesmo o valor atual delas. Podemos listar todas as variáveis declaradas no código usando o comando `dir()`. Veja um

<sup>32</sup> <https://docs.python.org/3/>

exemplo:

```
>>> a = 1
>>> b = 2
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__',
 '__spec__', 'a', 'b']
```

Veja que nossas variáveis declaradas aparecem no final do resultado de `dir()`. Não se assuste com os outros elementos que aparecem nesse resultado. Essas variáveis são criadas e usadas pelo próprio Python, e não são importantes nesse momento.

Outra opção para visualizar as variáveis declaradas são os comandos `globals()` e `locals()`. Ambas mostram não só as variáveis declaradas, mas também seu valor atual. A diferença entre ambas está no escopo em que atuam, mas veja que seus resultados são semelhantes:

```
>>> a = 1
>>> b = 2
>>> locals()
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <class '__frozen_importlib.BuiltinImporter'>, '__spec__': None, '__annotations__': {}, '__builtins__': <module 'builtins' (built-in)>, 'a': 1, 'b': 2}
>>> globals()
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <class '__frozen_importlib.BuiltinImporter'>, '__spec__': None, '__annotations__': {}, '__builtins__': <module 'builtins' (built-in)>, 'a': 1, 'b': 2}
```

Caso você esteja usando o IPython, os comandos mágicos `%who` e `%whos` são ótimas alternativas ao que já vimos anteriormente, pois retiram do resultado as variáveis declaradas pelo próprio Python, permitindo uma melhor visualização das que você mesmo declarou. Olhe como o IPython pode simplificar nossa vida nesse caso:

```
>>> a = 1
>>> %who
a
>>> %whos
Variable  Type      Data/Info
-----
a          int       1
```

## 7.7 Exercícios

1. Supondo que a cotação do dólar esteja em R\$ 3,25, salve esse valor em uma variável e utilize-o para calcular quanto você teria ao cambiar R\$ 65,00 para dólares.
2. Abelindo é um professor muito malvado. Ele quer decidir como reprovar Rondinelly, que tirou 8.66, 5.35, 5 e 1, respectivamente, nas provas P1, P2, P3 e P4. Para isso, ele pode calcular a nota final usando média aritmética (M.A.), média geométrica (M.G.) ou média harmônica (M.H.).

$$M.A. = \frac{P_1 + P_2 + P_3 + P_4}{4}$$

$$M.G. = \sqrt[4]{|P_1 P_2 P_3 P_4|}$$

$$M.H. = \frac{4}{\frac{1}{P_1} + \frac{1}{P_2} + \frac{1}{P_3} + \frac{1}{P_4}}$$

Qual dessas médias dá a maior nota pra Rondinelly? E qual das médias dá a pior nota?

3. Josefson deseja fazer compras na China. Ela quer comprar um celular de USD 299,99, uma chaleira de USD 23,87, um gnomo de jardim de USD 66,66 e 6 adesivos de unicórnio de USD 1,42 cada um. O frete de tudo isso para a cidade de Rolândia, no Paraná, ficou em USD 12,34.
- Calcule o valor total da compra em dólares.
  - Usando o mesmo valor do dólar do exercício anterior, calcule o preço final em Reais. Lembre-se que o valor do *IOF* é de 6,38 %.
  - Quanto ela pagou apenas de *IOF*?



# CAPÍTULO 8

## Strings (sequência de caracteres)

Strings são tipos que armazenam uma *sequência de caracteres*:

```
>>> "Texto bonito"  
'Texto bonito'  
>>> "Texto com acentos de cedilhas: hoje é dia de caça!"  
'Texto com acentos de cedilhas: hoje é dia de caça!'
```

As strings aceitam aspas simples também:

```
>>> nome = 'Silvio Santos'  
>>> nome  
'Silvio Santos'
```

**Nota:** Strings aceitam os dois tipos de aspas, desde que seja consistente. Se começou com uma, termine com aquela!

```
>>> cor_da_caneta = "azul brilhante"  
File "<stdin>", line 1  
    cor_da_caneta = "azul brilhante"  
          ^  
  
SyntaxError: EOL while scanning string literal
```

Também é possível fazer algumas operações com strings:

```
>>> nome * 3  
'Silvio SantosSilvio SantosSilvio Santos'  
>>> nome * 3.14  
Traceback (most recent call last):  
...  
TypeError: can't multiply sequence by non-int of type 'float'
```

```
>>> canto1 = 'vem ai,'  
>>> canto2 = 'lá '
```

(continues on next page)

(continued from previous page)

```
>>> nome + ' ' + canto1 + canto2 * 6 + '!!!'  
'Silvio Santos vem aí, lá lá lá lá lá lá !!!'
```

Para strings em várias linhas, utilize 3 aspas:

```
>>> string_grande = '''Aqui consigo inserir um textão com várias linhas, posso  
→iniciar em uma...  
... e posso continuar em outra  
... e em outra  
... e mais uma  
... e acabou.'''  
>>> string_grande  
'Aqui consigo inserir um textão com várias linhas, posso iniciar em uma...\ne posso  
→continuar em outra\n'e em outra\n'e mais uma\n'e acabou.'  
>>> print(string_grande)  
Aqui consigo inserir um textão com várias linhas, posso iniciar em uma...  
e posso continuar em outra  
e em outra  
e mais uma  
e acabou.
```

Caso queira um texto que dentro tem aspas, como Me dá um copo d'água, é necessário utilizar aspas duplas para formar a *string*:

```
>>> agua = "Me dá um copo d'água"  
>>> agua  
"Me dá um copo d'água"
```

E também é possível utilizar aspas simples, duplas e triplas ao mesmo tempo! Olha só:

```
>>> todas_as_aspas = """Essa é uma string que tem:  
... - aspas 'simples'  
... - aspas "duplas"  
... - aspas '''triplas'''  
... Legal né?"""  
>>> todas_as_aspas  
'Essa é uma string que tem:\n- aspas \'simples\'\n- aspas "duplas"\n- aspas \'\'\'\nLegal né?'  
>>> print(todas_as_aspas)  
Essa é uma string que tem:  
- aspas 'simples'  
- aspas "duplas"  
- aspas '''triplas'''  
Legal né?
```

## 8.1 Tamanho

A função embutida `len()` nos permite, entre outras coisas, saber o tamanho de uma *string*:

```
>>> len('Abracadabra')  
11  
>>> palavras = 'Faz um pull request lá'  
>>> len(palavras)  
22
```

Assim, vemos que a palavra Abracadabra tem 11 letras.

## 8.2 Índices

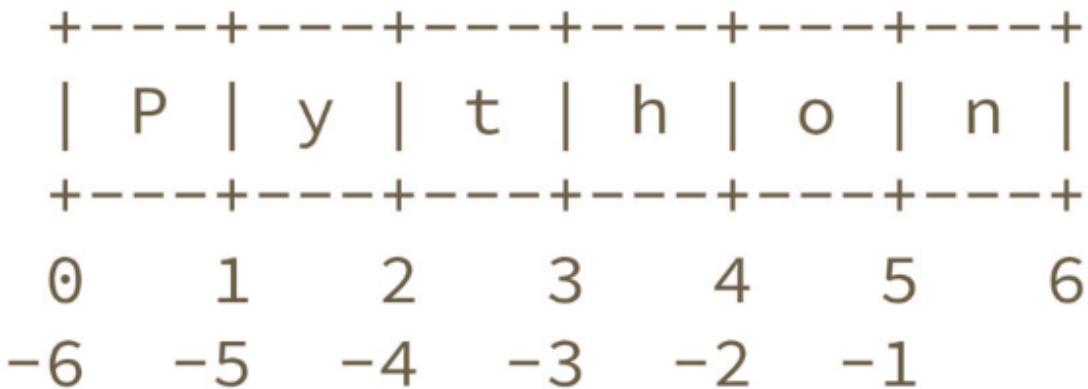
Como visto anteriormente, o método `len()` pode ser utilizado para obter o tamanho de estruturas, sejam elas *strings*, listas, etc. Esse tamanho representa a quantidade de elementos na estrutura.

Para obter somente um caractere de dentro dessas estruturas, deve-se utilizar o acesso por índices, no qual o índice entre colchetes `[]` representa a posição do elemento que se deseja acessar.

---

**Nota:** Os índices começam em zero.

---



```
>>> palavra = 'Python'
>>> palavra[0] # primeira
'P'
>>> palavra[5] # última
'n'
```

Índices negativos correspondem à percorrer a estrutura (*string*, lista, ...) na ordem reversa:

```
>>> palavra[-1] # última também
'n'
>>> palavra[-3] # terceira de tras pra frente
'h'
```

## 8.3 Fatiadas

Se, ao invés de obter apenas um elemento de uma estrutura (*string*, lista, ...), deseja-se obter múltiplos elementos, deve-se utilizar *slicing* (fatiamento). No lugar de colocar o índice do elemento entre chaves, deve-se colocar o índice do primeiro elemento, dois pontos `:` e o próximo índice do último elemento desejado, tudo entre colchetes. Por exemplo:

```
>>> frase = "Aprender Python é muito divertido!"  
>>> frase[0]  
'A'  
>>> frase[5]  
'd'  
>>> frase[0:5] # do zero até o antes do 5  
'Apren'  
>>> frase[:] # tudo!  
'Aprender Python é muito divertido!'  
>>> frase  
'Aprender Python é muito divertido!'  
>>> frase[6:] # Se omitido o segundo índice significa 'obter até o final'  
'er Python é muito divertido!'  
>>> frase[:6] # se omitido o primeiro índice, significa 'obter desde o começo'  
'Aprend'  
>>> frase[2:-3] # funciona com números negativos também  
'render Python é muito diverti'  
>>> frase[0:-5]  
'Aprender Python é muito diver'  
>>> frase[0:-6]  
'Aprender Python é muito dive'  
>>> frase[0:-7]  
'Aprender Python é muito div'  
>>> frase[2:-2]  
'render Python é muito divertid'
```

É possível controlar o *passo* que a fatia usa. Para isso, coloca-se mais um dois pontos (:) depois do segundo índice e o *tamanho do passo*:

```
>>> frase[::-1] # do começo, até o fim, de 1 em 1. Ou seja, tudo do jeito que ja era, ↵  
→não faz diferença nenhuma.  
'Aprender Python é muito divertido!'  
>>> frase[::-2] # do começo, até o fim, de 2 em 2  
'Arne yhnémiodvrio  
>>> frase[2:-2:2] # Do terceiro, até o ante penúltimo, de 2 em dois  
'rne yhnémiodvri'
```

Resumindo: para fazer uma fatia de nossa *string*, precisamos saber de onde começa, até onde vai e o tamanho do passo.

```
fatiável[começo : fim : passo]
```

---

**Nota:** As fatias incluem o índice do primeiro elemento e *não incluem* o elemento do índice final. Por isso que `frase[0:-1]` perde o último elemento.

---

Caso o final da fatia seja antes do começo, obtemos um resultado vazio:

```
>>> frase[15:2]  
''
```

E se quisermos uma fatia fora da string?

```
>>> frase[123:345]  
''
```

Mas e se o final da fatia for mais para frente que o tamanho da *string*? Não tem problemas, o Python vai até o onde der:

```
>>> frase[8:123456789]
' Python é muito divertido!'
>>> frase[8:]
' Python é muito divertido!'
>>> frase[123456789]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

Tamanhos negativos de passo também funcionam. Passos positivos significam *para frente* e passos negativos significam *para trás*:

```
>>> "Python" [::-1]
'nohtyP'
```

Quando usamos passos negativos, a fatia começa no fim e termina no começo e é percorrida ao contrário. Ou seja, invertemos a ordem. Mas tome cuidado:

```
>>> "Python" [2:6]
'thon'
>>> "Python" [2:6:-1]
''
>>> "Python" [6:2]
''
>>> "Python" [6:2:-1]
'noh'
```

No caso de "Python" [6:2], o começo é depois do fim. Por isso a *string* fica vazia.

No caso de "Python" [2:6:-1], o começo é o índice 6, o fim é o índice 2, percorrida ao contrário. Ou seja, temos uma *string* vazia ao contrário, que continua vazia.

Quando fazemos "Python" [6:2:-1], o começo é o índice 2, o fim é o índice 6, percorrida ao contrário. Lembre que o índice final nunca é incluído. Ou seja, temos a *string* hon a ser invertida. O que resulta em noh.

## 8.4 Formatação de strings

A formatação de string nos permite criar frases dinâmicas, utilizando valores de quaisquer variáveis desejadas. Por exemplo:

```
>>> nome = input('Digite seu nome ')
Digite seu nome Silvio Santos
>>> nome
'Silvio Santos'
>>> frase = 'Olá, {}'.format(nome)
>>> frase
'Olá, Silvio Santos'
```

Vale lembrar que as chaves {} só são trocadas pelo valor após a chamada do método `str.format()`:

```
>>> string_a_ser_formatada = '{} me formate!'
>>> string_a_ser_formatada
'{} me formate!'
```

(continues on next page)

(continued from previous page)

```
>>> string_a_ser_formatada.format("Não") # também podemos passar valores diretamente  
→para formatação, apesar de ser desnecessário  
'Não me formate!'
```

A string a ser formatada não é alterada nesse processo, já que não foi feita nenhuma atribuição:

```
>>> string_a_ser_formatada  
'{} me formate!'
```

É possível formatar uma quantidade arbitrária de valores:

```
>>> '{} x {} = {}'.format(7, 6, 7 * 6)  
'7 x 6 = 42'
```

```
>>> palavra = 'Python'  
>>> numero = 10  
>>> booleano = False  
>>> '{} é {}. E as outras linguagens? {}'.format(palavra, numero, booleano)  
'Python é 10. E as outras linguagens? False'
```

Além disso, também é possível usar nomes para identificar quais valores serão substituídos:

```
>>> '{a}, {a}, {a}. {b}, {b}, {b}'.format(a='oi', b='tchau')  
'oi, oi, oi. tchau, tchau, tchau'
```

#### 8.4.1 Alternativa ao `.format()`

Uma maneira mais recente de formatar strings foi introduzida a partir da versão 3.6 do Python: *PEP 498 – Literal String Interpolation*, carinhosamente conhecida como *fstrings* e funciona da seguinte forma:

```
>>> nome = 'Silvio'  
>>> f'Olá, {name}.'  
'Olá, Silvio.'
```

É também possível fazer operações:

```
>>> f'4654 * 321 é {4654 * 321}'  
'4654 * 321 é 1493934'
```

### 8.5 Separação de Strings

Se tivermos a frase Sílvio Santos vem aí, oleoleolá! e quisermos separar cada palavra, como fazer? Pode-se usar o fatiamento:

```
>>> frase = "Sílvio Santos vem aí, oleoleolá!"  
>>> frase[:6]  
'Sílvio'  
>>> frase[7:13]  
'Santos'  
>>> frase[14:17]
```

(continues on next page)

(continued from previous page)

```
'vem'
>>> frase[18:21]
'aí,' 
>>> frase[22:]
'oleoleolá!'
```

Mas também podemos usar a função `split()`:

```
>>> frase.split()
['Sílvio', 'Santos', 'vem', 'aí,', 'oleoleolá!']
```

---

**Nota:** É possível transformar uma string em número, dado que seja um número:

```
>>> numero = int("2")
>>> numero
2
```

---

**Nota:** A volta também é possível:

```
>>> numero_string = str(1900)
>>> numero_string
'1900'
>>> type(numero_string)
<class 'str'>
```

---

## 8.6 Exercícios

1. Dada a frase `Python é muito legal.`, use fatiamento para dar nome às variáveis contendo cada palavra. O resultado final deve ser:

```
>>> frase = "Python é muito legal."
# resolução do problema aqui
>>> palavra1
"Python"
>>> palavra2
"é"
>>> palavra3
"muito"
>>> palavra4
"legal"
```

2. Qual o tamanho dessa frase? E qual o tamanho de cada palavra?
3. Agora que conhecemos atribuição múltipla e o método `str.split()` refaça os dois exercícios anteriores usando essas técnicas.
4. Use *slicing* (mais especificamente o passo do fatiamento) para inverter a string «Python».



# CAPÍTULO 9

## Lendo valores do teclado

Em Python também é possível ler do teclado as informações digitadas pelo usuário. E isso é feito por meio da função embutida `input()` da seguinte forma:

```
>>> valor_lido = input("digite um valor: ")
digite um valor: 10

>>> type(valor_lido)  # deve-se notar que o valor lido é SEMPRE do tipo string
<class 'str'>
```

A função `input()` «termina» de ser executada quando pressionamos *enter*.

---

**Nota:** O valor lido é sempre do tipo string.

---

Mas, como realizar operações com os valores lidos?

```
>>> valor_lido + 10  # para trabalhar com esse valor, é preciso converter para o tipo_
  ↵correto
Traceback (most recent call last):
...
TypeError: must be str, not int
```

Para poder fazer isso pode-se usar os operadores `int()` e `float()`, que converte o valor lido para o tipo de dado esperado:

```
>>> valor_lido = int(input("digite um valor inteiro: "))
digite um valor inteiro: 10

>>> type(valor_lido)
<class 'int'>

>>> valor_lido + 10
20
```

(continues on next page)

(continued from previous page)

```
>>> valor_lido = float(input("digite um valor decimal: "))
digite um valor decimal: 1.5

>>> valor_lido - 1
0.5
```

Tudo o que for digitado no teclado, até pressionar a tecla *enter*, será capturado pela função `input()`. Isso significa que podemos ler palavras separadas por um espaço, ou seja, uma frase inteira:

```
>>> frase = input()
Rosas são vermelhas, violetas são azuis, girassóis são legais.
>>> frase
'Rosas são vermelhas, violetas são azuis, girassóis são legais.'
```

## 9.1 Exercícios

1. Leia um nome pelo teclado e imprima "Olá, <nome lido>! "
2. Leia outro nome pelo teclado e imprima:

```
<nome lido> roubou pão na cassa do <nome2 lido>!
<nome2 lido> ficou triste e com fome,
porque o bandejão estava fechado.
```

3. Leia uma frase pelo teclado e a imprima ao contrário.

Por exemplo, se a frase for "Manjo muito de Python!", a saída deverá ser '!nohtyP ed otium ojnaM'.

# CAPÍTULO 10

---

## Listas

---

Listas são estruturas de dados capazes de armazenar múltiplos elementos.

### 10.1 Declaração

Para a criação de uma lista, basta colocar os elementos separados por vírgulas dentro de colchetes [], como no exemplo abaixo:

```
>>> nomes_frutas = ["maçã", "banana", "abacaxi"]
>>> nomes_frutas
['maçã', 'banana', 'abacaxi']

>>> numeros = [2, 13, 17, 47]
>>> numeros
[2, 13, 17, 47]
```

A lista pode conter elementos de tipos diferentes:

```
>>> ['lorem ipsum', 150, 1.3, [-1, -2]]
['lorem ipsum', 150, 1.3, [-1, -2]]

>>> vazia = []
>>> vazia
[]
```

### 10.2 Índices

Assim como nas *strings*, é possível acessar separadamente cada item de uma lista a partir de seu índice:

```
>>> lista = [100, 200, 300, 400, 500]
>>> lista[0]  # os índices sempre começam em 0
100

>>> lista[2]
300

>>> lista[4]  # último elemento
500

>>> lista[-1]  # outra maneira de acessar o último elemento
500
```

Conforme visto anteriormente, ao utilizar um índice negativo os elementos são acessados de trás pra frente, a partir do final da lista:

```
>>> lista[-2]  # penúltimo elemento
400

>>> lista[-3]  # terceiro
300

>>> lista[-4]  # segundo
200

>>> lista[-5]  # primeiro
100
```

Ou pode-se acessar através de *slices* (fatias), como nas *strings* (Página 51):

```
>>> lista[2:4]  # da posição 2 até a 4 (não inclusa)
[300, 400]

>>> lista[:3]    # até a posição 3 (não incluso)
[100, 200, 300]

>>> lista[2:]    # da posição 2 até o final
[300, 400, 500]

>>> lista[:]     # do começo até o final
[100, 200, 300, 400, 500]
```

Tentar acessar uma posição inválida de uma lista causa um erro:

```
>>> lista[10]
Traceback (most recent call last):
...
IndexError: list index out of range

>>> lista[-10]
Traceback (most recent call last):
...
IndexError: list index out of range
```

Podemos avaliar se os elementos estão na lista com a palavra `in`:

```

>>> lista_estranya = ['duas palavras', 42, True, ['batman', 'robin'], -0.84, 'hipófise'
->']
>>> 42 in lista_estranya
True

>>> 'duas palavras' in lista_estranya
True

>>> 'dominó' in lista_estranya
False

>>> 'batman' in lista_estranya[3] # note que o elemento com índice 3 também é uma_
->lista
True

```

É possível obter o tamanho da lista utilizando o método `len()`:

```

>>> len(lista)
5

>>> len(lista_estranya)
6

>>> len(lista_estranya[3])
2

```

## 10.3 Removendo itens da lista

Devido à lista ser uma estrutura mutável, é possível remover seus elementos utilizando o comando `del`:

```

>>> lista_estranya
['duas palavras', 42, True, ['batman', 'robin'], -0.84, 'hipófise']

>>> del lista_estranya[2]
>>> lista_estranya
['duas palavras', 42, ['batman', 'robin'], -0.84, 'hipófise']

>>> del lista_estranya[-1] # Remove o último elemento da lista
>>> lista_estranya
['duas palavras', 42, ['batman', 'robin'], -0.84]

```

## 10.4 Trabalhando com listas

O operador `+` concatena listas:

```

>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> c
[1, 2, 3, 4, 5, 6]

```

O operador `*` repete a lista dado um número de vezes:

```
>>> [0] * 3
[0, 0, 0]

>>> [1, 2, 3] * 2
[1, 2, 3, 1, 2, 3]
```

O método `append()` adiciona um elemento ao final da lista:

```
>>> lista = ['a', 'b', 'c']
>>> lista
['a', 'b', 'c']

>>> lista.append('e')
>>> lista
['a', 'b', 'c', 'e']
```

Temos também o `insert()`, que insere um elemento na posição especificada e move os demais elementos para direita:

```
>>> lista.insert(3, 'd')
>>> lista
['a', 'b', 'c', 'd', 'e']
```

**Aviso:** Cuidado com `lista.insert(-1, algo)`! Nesse caso, inserimos algo na posição -1 e o elemento que estava previamente na posição -1 é movido para a direita:

```
>>> lista.insert(-1, 'ç')
>>> lista
['a', 'b', 'c', 'd', 'ç', 'e']
```

Use `append()` caso queira algo adicionado ao final da lista.

`extend()` recebe uma lista como argumento e adiciona todos seus elementos a outra:

```
>>> lista1 = ['a', 'b', 'c']
>>> lista2 = ['d', 'e']
>>> lista1
['a', 'b', 'c']

>>> lista2
['d', 'e']

>>> lista1.extend(lista2)
>>> lista1
['a', 'b', 'c', 'd', 'e']
```

`lista2` não é modificado:

```
>>> lista2
['d', 'e']
```

O método `sort()` ordena os elementos da lista em ordem ascendente:

```
>>> lista_desordenada = ['b', 'z', 'k', 'a', 'h']
>>> lista_desordenada
```

(continues on next page)

(continued from previous page)

```
[ 'b', 'z', 'k', 'a', 'h']

>>> lista_desordenada.sort()
>>> lista_desordenada # Agora está ordenada!
[ 'a', 'b', 'h', 'k', 'z']
```

Para fazer uma cópia de uma lista, devemos usar o método `copy()`:

```
>>> lista1 = [ 'a', 'b', 'c']
>>> lista2 = lista1.copy()
>>> lista1
[ 'a', 'b', 'c']
>>> lista2
[ 'a', 'b', 'c']
>>> lista2.append('d')
>>> lista1
[ 'a', 'b', 'c']
>>> lista2
[ 'a', 'b', 'c', 'd']
```

Se não usarmos o `copy()`, acontece algo bem estranho:

```
>>> lista1 = [ 'a', 'b', 'c']
>>> lista2 = lista1
>>> lista1
[ 'a', 'b', 'c']
>>> lista2
[ 'a', 'b', 'c']
>>> lista2.append('d')
>>> lista1
[ 'a', 'b', 'c', 'd']
>>> lista2
[ 'a', 'b', 'c', 'd']
```

Tudo o que for feito com `lista2` nesse exemplo também altera `lista1` e vice-versa.

## 10.5 Exercícios

1. Crie uma lista com o nome das 3 pessoas mais próximas.
2. Crie três listas, uma lista de cada coisa a seguir:

- frutas
- docinhos de festa (não se esqueça de brigadeiros!!)
- ingredientes de feijoada

Lembre-se de salvá-las em alguma variável!

- a. Agora crie uma lista que contém essas três listas.

Nessa lista de listas (vou chamar de *listona*):

- b. você consegue acessar o elemento *brigadeiro*?
- c. Adicione mais *brigadeiros* à segunda lista de *listona*. O que aconteceu com a lista de *docinhos de festa*?
- d. Adicione bebidas ao final da *listona*, mas sem criar uma lista!

3. Utilizando o `del`, remova todos os elementos da lista criada anteriormente até a lista ficar vazia.
  4. Faça uma lista de compras do mês, não se esqueça de comprar produtos de limpeza e sorvete!
- Agora «vá ao mercado» e delete apenas os produtos de limpeza da lista.
- Agora «vá à sorveteria» e se empanturre de sorvete e tire o sorvete da lista.
5. Dado uma lista de números, faça com que os números sejam ordenados e, em seguida, inverta a ordem da lista usando *slicing*.

## 10.6 Função range()

Aprendemos a adicionar itens a uma lista mas, e se fosse necessário produzir uma lista com os números de 1 até 200?

```
>>> lista_grande = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13] # ???
>>> lista_grande
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
```

Em Python existe a função embutida `range()`, com ela é possível produzir uma lista extensa de uma maneira bem simples:

```
>>> print(list(range(1, 200)))
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,
 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65,
 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86,
 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105,
 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122,
 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138,
 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155,
 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171,
 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188,
 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199]
```

Além disso, o `range()` também oferece algumas coisas interessantes. Por exemplo, imprimir os números espaçados de 5 em 5, entre 0 e 30:

```
>>> print(list(range(0, 30, 5)))
[0, 5, 10, 15, 20, 25]
```

Mas, como na maior parte das vezes apenas queremos uma lista começando em 0 e indo até o número desejado, a função `range()` também funciona da seguinte maneira:

```
>>> print(list(range(10)))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

---

**Nota:** O intervalo do `range()` é aberto, ou seja, quando passamos o valor 10, ele vai até o 9 ( $n - 1$ ). Caso deseje criar a lista até o 10 de fato, deve-se passar o valor 11.

---

Mas por que precisamos transformar o `range()` em `list`? O que acontece se não fizermos isso?

```
>>> print(range(200))
range(0, 200)
```

Mas o que é que essa função retorna?

```
>>> type(range(200))
<class 'range'>"
```

AHA! A função `range()` retorna algo do tipo `range`, por isso precisamos transformar em uma lista para vermos todos os números no `print()`!



# CAPÍTULO 11

---

## Dicionários

---

Dicionário é uma coleção de itens (chamados *chaves*) e seus respectivos significados (chamados de *valores*):

{chave: valor}

Cada chave do dicionário deve ser única! Ao contrário de *listas*, *dicionários*, não podem ter chaves repetidas.

---

**Nota:** As chaves devem ser únicas.

---

### 11.1 Declaração

Declaramos um dicionário colocando entre colchetes {} cada chave e o seu respectivo valor, da seguinte forma:

```
>>> telefones = {"ana": 123456, "yudi": 40028922, "julia": 4124492}
>>> telefones
{'ana': 123456, 'yudi': 40028922, 'julia': 4124492}
```

No caso acima, a chave "ana", por exemplo, está relacionada ao valor 123456. Cada par chave-valor é separado por uma vírgula , .

### 11.2 Função dict()

A função dict() constrói um dicionário. Existem algumas formas de usá-la:

- Com uma lista de listas:

```
>>> lista1 = ["brigadeiro", "leite condesado, achocolatado"]
>>> lista2 = ["omelete", "ovos, azeite, condimentos a gosto"]
>>> lista3 = ["ovo frito", "ovo, óleo, condimentos a gosto"]
>>> lista_receitas = [lista1, lista2, lista3]
```

(continues on next page)

(continued from previous page)

```
>>> print(lista_receitas)
[['brigadeiro', 'leite condesado, achocolatado'], ['omelete', 'ovos, azeite,',
    ↪condimentos a gosto'], ['ovo frito', 'ovo, óleo, condimentos a gosto']]
>>> receitas = dict(lista_receitas)
>>> print(receitas)
{'brigadeiro': 'leite condesado, achocolatado', 'omelete': 'ovos, azeite,',
    ↪condimentos a gosto', 'ovo frito': 'ovo, óleo, condimentos a gosto'}
```

- Atribuindo os valores diretamente:

```
>>> constantes = dict(pi=3.14, e=2.7, alpha=1/137)
>>> print(constantes)
{'pi': 3.14, 'e': 2.7, 'alpha': 0.0072992700729927005}
```

Neste caso, o nome das chaves deve ser um identificador válido. As mesmas regras de *nomes de variáveis* (Página 41) se aplicam.

- Usando as chaves {}:

```
>>> numerinhos = dict({"um": 1, "dois": 2, "três": 3})
>>> print(numerinhos)
{'um': 1, 'dois': 2, 'três': 3}
```

E nesse caso se não houvesse a função *dict()*, o resultado seria exatamente o mesmo...

## 11.3 Chaves

Acessamos um determinado valor do dicionário através de sua chave:

```
>>> capitais = {"SP": "São Paulo", "AC": "Rio Branco", "TO": "Palmas", "RJ": "Rio de",
    ↪Janeiro", "SE": "Aracaju", "MG": "Belo Horizonte"}
>>> capitais["MG"]
'Belo Horizonte'
```

Até o momento, usamos apenas *strings*, mas podemos colocar todo tipo de coisa dentro dos dicionários, incluindo listas e até mesmo outros dicionários:

```
>>> numeros = {"primos": [2, 3, 5], "pares": [0, 2, 4], "ímpares": [1, 3, 5]}
>>> numeros["ímpares"]
[1, 3, 5]
```

Mesmo que os pares chave-valor estejam organizados na ordem que foram colocados, não podemos acessá-los por *índices* como faríamos em listas:

```
>>> numeros[2]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 2
```

O mesmo erro ocorre se tentarmos colocar uma chave que não pertence ao dicionário:

```
>>> numeros["negativos"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'negativos'
```

Assim como os valores não precisam ser do tipo *string*, o mesmo vale para as chaves:

```
>>> numeros_por_extenso = {2: "dois", 1: "um", 3: "três", 0: "zero"}
>>> numeros_por_extenso[0]
'zero'
>>> numeros_por_extenso[2]
'dois'
```

---

**Nota:** Listas e outros dicionários *não* podem ser usados como chaves por serem de tipos *mutáveis*.

---

## 11.4 Adicionando e removendo elementos

Podemos alterar o valor relacionado a uma chave da seguinte forma:

```
>>> pessoa = {"nome": "Cleiton", "idade": 34, "família": {"mãe": "Maria", "pai": "Enzo
   "}
>>> pessoa["idade"]
34
>>> pessoa["idade"] = 35
>>> pessoa["idade"]
35
```

Para adicionar um elemento novo à um dicionário, podemos simplesmente fazer o seguinte:

```
>>> meses = {1: "Janeiro", 2: "Fevereiro", 3: "Março"}
>>> meses[4] = "Abril"
>>> meses
{1: "Janeiro", 2: "Fevereiro", 3: "Março", 4: "Abril"}
```

Aqui nos referimos a uma chave que *não* está no dicionário e associamos um valor a ela. Desta forma, *adicionando* esse conjunto chave-valor ao dicionário.

Removemos um conjunto chave-elemento de um dicionário com o comando `del`:

```
>>> meses
{1: "Janeiro", 2: "Fevereiro", 3: "Março", 4: "Abril"}
>>> del(meses[4])
>>> meses
{1: "Janeiro", 2: "Fevereiro", 3: "Março"}
```

Para apagar *todos* os elementos de um dicionário, usamos o método *clear*:

```
>>> lixo = {"plástico": ["garrafa", "copinho", "canudo"], "papel": ["folha amassada",
   "guardanapo"], "orgânico": ["batata", "resto do bandeco", "casca de banana"]}
>>> lixo
{"plástico": ["garrafa", "copinho", "canudo"], "papel": ["folha amassada", "guardanapo
   "], "orgânico": ["batata", "resto do bandeco", "casca de banana"]}
>>> lixo.clear()
>>> lixo
{}
```

## 11.5 Função list()

A função `list()` recebe um conjunto de objetos e retorna uma lista. Ao passar um dicionário, ela retorna uma lista contendo todas as suas *chaves*:

```
>>> institutos_uspsc = {"IFSC": "Instituto de Física de São Carlos", "ICMC":  
    "Instituto de Ciências Matemáticas e de Computação", "EESC": "Escola de Engenharia  
    de São Carlos", "IAU": "Instituto de Arquitetura e Urbanismo", "IQSC": "Instituto  
    de Química de São Carlos"}  
>>> list(institutos_uspsc)  
['IQSC', 'IFSC', 'ICMC', 'IAU', 'EESC']
```

## 11.6 Função len()

A função `len()` retorna o número de elementos («tamanho») do objeto passado para ela. No caso de uma lista, fala quantos elementos há. No caso de dicionários, retorna o *número de chaves* contidas nele:

```
>>> institutos_uspsc  
{'IQSC': 'Instituto de Química de São Carlos', 'IFSC': 'Instituto de Física de São  
    Carlos', 'ICMC': 'Instituto de Ciências Matemáticas e de Computação', 'IAU':  
    'Instituto de Arquitetura e Urbanismo', 'EESC': 'Escola de Engenharia de São Carlos  
    '}  
>>> len(institutos_uspsc)  
5
```

Você pode contar o número de elementos na lista gerada pela função `list()` para conferir:

```
>>> len(list(institutos_uspsc))  
5
```

## 11.7 Método get()

O método `get(chave, valor)` pode ser usado para retornar o valor associado à respectiva chave! O segundo parâmetro `<valor>` é opcional e indica o que será retornado caso a chave desejada *não* esteja no dicionário:

```
>>> institutos_uspsc.get("IFSC")  
'Instituto de Física de São Carlos'
```

Dá para ver que ele é muito parecido com fazer assim:

```
>>> institutos_uspsc["IFSC"]  
'Instituto de Física de São Carlos'
```

Mas ao colocarmos uma chave que não está no dicionário:

```
>>> institutos_uspsc.get("Poli", "Não tem!")  
'Não tem!'  
>>> institutos_uspsc["Poli"]  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
KeyError: 'Poli'
```

## 11.8 Alguns métodos

- O método `items()` pode ser comparado com o *inverso* da função `dict()`:

```
>>> pessoa = { "nome": "Enzo", "RA": 242334, "curso": "fiscomp"}
>>> pessoa.items()
dict_items([('curso', 'fiscomp'), ('nome', 'Enzo'), ('RA', 242334)])
```

Usando a função `list()` nesse resultado, obtemos:

```
>>> pessoa.items()
dict_items([('curso', 'fiscomp'), ('nome', 'Enzo'), ('RA', 242334)])
>>> itens = list(pessoa.items())
>>> itens
[('curso', 'fiscomp'), ('nome', 'Enzo'), ('RA', 242334)]
```

Experimente usar a função `dict()` na lista `itens`!

- O método `values()` nos retorna os *valores* do dicionário:

```
>>> pessoa.values()
dict_values(['fiscomp', 'Enzo', 242334])
>>> valores = list(pessoa.values())
>>> valores
['fiscomp', 'Enzo', 242334]
```

- O método `keys()` nos retorna as *chaves* do dicionário:

```
>>> pessoa.keys()
dict_keys(['curso', 'nome', 'RA'])
>>> chaves = list(pessoa.keys())
>>> chaves
['curso', 'nome', 'RA']
```

Repare que nesse último obtemos o mesmo que se tivéssemos usado a função `list()`:

```
>>> list(pessoa)
['curso', 'nome', 'RA']
```

## 11.9 Ordem dos elementos

Dicionários não tem sequência dos seus elementos. As listas têm. Dicionários mapeiam um valor a uma chave. Veja este exemplo:

```
>>> numerinhos = dict({"um": 1, "dois": 2, "três": 3})
>>> numeritos = {"três": 3, "dois": 2, "um": 1}
>>> numerinhos == numeritos
True
>>> numeritos
{'três': 3, 'dois': 2, 'um': 1}
>>> numerinhos
{'um': 1, 'dois': 2, 'três': 3}
```

Vemos que `numerinhos` e `numeritos` têm as mesmas chaves com os mesmos valores e por isso são iguais. Mas quando imprimimos cada um, a ordem que aparece é a que os itens foram inseridos.

## 11.10 Está no dicionário?

Podemos checar se uma chave está ou não em um dicionário utilizando o comando `in`. Voltando para o dicionário que contem os institutos da USP São Carlos:

```
>>> institutos_uspsc
{'IQSC': 'Instituto de Química de São Carlos', 'IFSC': 'Instituto de Física de São
    ↵Carlos', 'ICMC': 'Instituto de Ciências Matemáticas e de Computação', 'IAU':
    ↵'Instituto de Arquitetura e Urbanismo', 'EESC': 'Escola de Engenharia de São Carlos
    ↵'}
>>> "IFSC" in institutos_uspsc
True
>>> "ESALQ" in institutos_uspsc
False
```

E checamos se uma chave *não está* no dicionário com o comando `not in`:

```
>>> institutos_uspsc
{'IQSC': 'Instituto de Química de São Carlos', 'IFSC': 'Instituto de Física de São
    ↵Carlos', 'ICMC': 'Instituto de Ciências Matemáticas e de Computação', 'IAU':
    ↵'Instituto de Arquitetura e Urbanismo', 'EESC': 'Escola de Engenharia de São Carlos
    ↵'}
>>> "IFSC" not in institutos_uspsc
False
>>> "ESALQ" not in institutos_uspsc
True
```

## 11.11 Exercícios

1. Faça um dicionário com as 5 pessoas mais perto de você, tendo o nome como chave e a cor da camisa que está usando como valor.
2. Crie um dicionário vazio `semana = {}` e o complete com uma chave para cada dia da semana, tendo como seu valor uma lista com as aulas que você tem nesse dia (sábado e domingo recebem listas vazias, ou você tem aula?).
3. Crie um dicionário vazio `filmes = {}`. Utilize o nome de um filme como chave. E, como valor, *outro* dicionário contendo o vilão e o ano em que o filme foi lançado. Preencha 5 filmes.

# CAPÍTULO 12

---

## Condicionais

---

O tipo de dado booleano (`bool`) refere-se a uma unidade lógica sobre a qual podemos realizar operações, particularmente úteis para o controle de fluxo de um programa.

A unidade booleana assume apenas 2 valores: Verdadeiro (`True`) e Falso (`False`).

---

**Nota:** Essa estrutura binária é a forma com a qual o computador opera (0 e 1).

---

```
>>> True  
True  
>>> type(False)  
<class 'bool'>
```

Qualquer expressão lógica retornará um valor booleano:

```
>>> 2 < 3  
True  
>>> 2 == 5  
False
```

Os operadores lógicos utilizados em programação são:

- `>`: maior a, por exemplo  $5 > 3$
- `<`: menor a
- `>=`: maior ou igual a
- `<=`: menor ou igual a
- `==`: igual a
- `!=`: diferente de

Para realizar operações com expressões lógicas, existem:

- **and (e):** opera segundo a seguinte tabela:

Valor 1	Valor 2	Resultado
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Falso
Falso	Verdadeiro	Falso
Falso	Falso	Falso

- **or (ou):**

Valor 1	Valor 2	Resultado
Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Verdadeiro
Falso	Verdadeiro	Verdadeiro
Falso	Falso	Falso

- **not (não):**

Valor	Resultado
Verdadeiro	Falso
Falso	Verdadeiro

```
>>> 10 > 3 and 2 == 4
False

>>> 10 > 3 or 2 == 4
True

>>> not not not 1 == 1
False
```

Assim como os operadores aritméticos, os operadores booleanos também possuem uma ordem de prioridade:

- not tem maior prioridade que and que tem maior prioridade que or:

```
>>> not False and True or False
True
```

# CAPÍTULO 13

---

## Estruturas de controle

---

As estruturas de controle servem para decidir quais blocos de código serão executados.

### Exemplo:

Se estiver nublado:

Levarei guarda-chuva

Senão:

Não levarei

---

**Nota:** Na linguagem Python, a indentação (espaço dado antes de uma linha) é utilizada para demarcar os blocos de código, e são obrigatórios quando se usa estruturas de controle.

---

```
>>> a = 7
>>> if a > 3:
...     print("estou no if")
... else:
...     print("caí no else")
...
estou no if
```

Também é possível checar mais de uma condição com o `elif`. É a abreviatura para `else if`. Ou seja, se o `if` for falso, testa outra condição antes do `else`:

```
>>> valor_entrada = 10
>>> if valor_entrada == 1:
...     print("a entrada era 1")
... elif valor_entrada == 2:
...     print("a entrada era 2")
... elif valor_entrada == 3:
...     print("a entrada era 3")
... elif valor_entrada == 4:
...     print("a entrada era 4")
```

(continues on next page)

(continued from previous page)

```
... else:
...     print("o valor de entrada não era esperado em nenhum if")
...
o valor de entrada não era esperado em nenhum if
```

Note que quando uma condição for verdadeira, aquele bloco de código é executado e as demais condições (`elif` e `else`) são puladas:

```
>>> a = 1
>>> if a == 1:
...     print("é 1")
... elif a >= 1:
...     print("é maior ou igual a 1")
... else:
...     print("é qualquer outra coisa")
...
é 1
```

## 13.1 Exercícios

1. Escreva um programa que, dados 2 números diferentes (a e b), encontre o menor deles.
  2. Para doar sangue é necessário<sup>1</sup>:
    - Ter entre 16 e 69 anos.
    - Pesar mais de 50 kg.
    - Estar descansado (ter dormido pelo menos 6 horas nas últimas 24 horas).

Faça um programa que pergunte a idade, o peso e quanto dormiu nas últimas 24 h para uma pessoa e diga se ela pode doar sangue ou não.
  3. Considere uma equação do segundo grau  $f(x) = a \cdot x^2 + b \cdot x + c$ . A partir dos coeficientes, determine se a equação possui duas raízes reais, uma, ou se não possui.
- Dica:**  $\Delta = b^2 - 4 \cdot a \cdot c$ : se delta é maior que 0, possui duas raízes reais; se delta é 0, possui uma raiz; caso delta seja menor que 0, não possui raiz real
4. Leia dois números e efetue a adição. Caso o valor somado seja maior que 20, este deverá ser apresentado somando-se a ele mais 8; caso o valor somado seja menor ou igual a 20, este deverá ser apresentado subtraindo-se 5.
  5. Leia um número e imprima a raiz quadrada do número caso ele seja positivo ou igual a zero e o quadrado do número caso ele seja negativo.
  6. Leia um número inteiro entre 1 e 12 e escreva o mês correspondente. Caso o usuário digite um número fora desse intervalo, deverá aparecer uma mensagem informando que não existe mês com este número.

<sup>1</sup> Para mais informações sobre doação de sangue, acesse [http://www.prosangue.sp.gov.br/artigos/requisitos\\_basicos\\_para\\_doacao.html](http://www.prosangue.sp.gov.br/artigos/requisitos_basicos_para_doacao.html)

# CAPÍTULO 14

---

## Estruturas de repetição

---

As estruturas de repetição são utilizadas quando queremos que um bloco de código seja executado várias vezes.

Em Python existem duas formas de criar uma estrutura de repetição:

- O `for` é usado quando se quer iterar sobre um bloco de código um número determinado de vezes.
- O `while` é usado quando queremos que o bloco de código seja repetido até que uma condição seja satisfeita. Ou seja, é necessário que uma expressão booleana dada seja verdadeira. Assim que ela se tornar falsa, o `while` para.

---

**Nota:** Na linguagem Python a indentação é obrigatória. assim como nas estruturas de controle, as estruturas de repetição também precisam.

```
>>> # Aqui repetimos o print 3 vezes
>>> for n in range(0, 3):
...     print(n)
...
0
1
2

>>> # Aqui iniciamos o n em 0, e repetimos o print até que seu valor seja maior ou_
... igual a 3
>>> n = 0
>>> while n < 3:
...     print(n)
...     n += 1
...
0
1
2
```

---

O `loop for` em Python itera sobre os itens de um conjunto, sendo assim, o `range(0, 3)` precisa ser um conjunto de elementos. E na verdade ele é:

```
>>> list(range(0, 3))
[0, 1, 2]
```

Para iterar sobre uma lista:

```
>>> lista = [1, 2, 3, 4, 10]
>>> for numero in lista:
...     print(numero ** 2)
...
1
4
9
16
100
```

Isso se aplica para *strings* também:

```
>>> # Para cada letra na palavra, imprimir a letra
>>> palavra = "casa"
>>> for letra in palavra:
...     print(letra)
...
c
a
s
a
```

Em dicionários podemos fazer assim:

```
>>> gatinhos = {"Português": "gato", "Inglês": "cat", "Francês": "chat", "Finlandês":
...             "Kissa"}
>>> for chave, valor in gatinhos.items():
...     print(chave, "->", valor)
...
Português -> gato
Inglês -> cat
Francês -> chat
Finlandês -> Kissia
```

Para auxiliar as estruturas de repetição, existem dois comandos:

- **break**: É usado para sair de um *loop*, não importando o estado em que se encontra.
- **continue**: Funciona de maneira parecida com a do **break**, porém no lugar de encerrar o *loop*, ele faz com que todo o código que esteja abaixo (porém ainda dentro do *loop*) seja ignorado e avança para a próxima iteração.

Veja a seguir um exemplo de um código que ilustra o uso desses comandos. Note que há uma *string* de documentação no começo que explica a funcionalidade.

```
"""
Esse código deve rodar até que a palavra "sair" seja digitada.
* Caso uma palavra com 2 ou menos caracteres seja digitada, um aviso
  deve ser exibido e o loop será executado do início (devido ao
  continue), pedindo uma nova palavra ao usuário.
* Caso qualquer outra palavra diferente de "sair" seja digitada, um
  aviso deve ser exibido.
* Por fim, caso a palavra seja "sair", uma mensagem deve ser exibida e o
  loop deve ser encerrado (break).

```

(continues on next page)

(continued from previous page)

```
"""
>>> while True:
...     string_digitada = input("Digite uma palavra: ")
...     if string_digitada.lower() == "sair":
...         print("Fim!")
...         break
...     if len(string_digitada) < 2:
...         print("String muito pequena")
...         continue
...     print("Tente digitar \"sair\"")
...
Digite uma palavra: oi
Tente digitar "sair"
Digite uma palavra: ?
String muito pequena
Digite uma palavra: sair
Fim!
```

## 14.1 Exercícios

1. Calcule a tabuada do 13.
2. Ler do teclado uma lista com 5 inteiros e imprimir o menor valor.
3. Ler do teclado uma lista com 5 inteiros e imprimir True se a lista estiver ordenada de forma crescente ou False caso contrário.
4. Exiba em ordem decrescente todos os números de 500 até 10.
5. Ler do teclado 10 números e imprima a quantidade de números entre 10 e 50.
6. Ler do teclado a idade e o sexo de 10 pessoas, calcule e imprima:
  - a) idade média das mulheres
  - b) idade média dos homens
  - c) idade média do grupo
7. Calcule o somatório dos números de 1 a 100 e imprima o resultado.



# CAPÍTULO 15

---

## Funções

---

Função é uma sequência de instruções que executa uma operação de computação. Ao definir uma função, você especifica o nome e a sequência de instruções. Depois, pode utilizar (“chamar”) a função pelo nome.

A ideia é similar às funções matemáticas! Mas funções em uma linguagem de programação não realizam necessariamente apenas cálculos.

Vimos o `type()`, um exemplo de função:

```
>>> type(23)
<class 'int'>

>>> type('textinho')
<class 'str'>
```

### 15.1 Definindo funções

Se quisermos uma função que ainda não existe em Python, temos que *definí-la*.

Criando uma função simples:

```
def NOME_DA_FUNÇÃO(LISTA DE PARÂMETROS):
    COMANDOS
```

**Aviso:** Coloque os dois pontos após definir a função!

---

**Nota:** Faça a indentação nas linhas abaixo da definição da função!

---

```
>>> def soma():
...     print(1 + 1)
...
>>> soma()
2

>>> def soma():
...     return 1 + 1
...
>>> soma()
2
```

Qual a diferença entre utilizar `print()` e `return()` aqui em cima?!?

```
>>> def imprime_letra():
...     print("If you didn't care what happened to me. And I didn't care for you")
...
>>> imprime_letra()
If you didn't care what happened to me. And I didn't care for you

>>> type(imprime_letra)
<class 'function'>

>>> def repete_letra():
...     imprime_letra()
...     imprime_letra()
...
>>> repete_letra()
If you didn't care what happened to me. And I didn't care for you
If you didn't care what happened to me. And I didn't care for you
```

## 15.2 Funções com argumentos

Queremos somar 3 com um número qualquer que insiro na função. Bora lá:

```
>>> def soma_valor(x):
...     return 3 + x
...
>>> soma_valor(5)
8

>>> z = soma_valor(10)
>>> z
13
```

Que sem graça! Quero somar dois números quaisquer!

```
>>> def soma_dois_numeros(x, y):
...     return x + y
...
>>> soma_dois_numeros(7, 4)
11
```

Tenho dificuldade com a tabuada do 7! Ajude-me!

```
>>> def tabuada_do_7():
...     for i in range(11):
...         print (7 * i)
...
>>> tabuada_do_7()
0
7
14
21
28
35
42
49
56
63
70
```

Mai tá legal isso! Quero a tabuada do 1 ao 10 agora! Bora!

```
>>> def tabuadas():
...     for i in range(1, 11):
...         for j in range(1, 11):
...             print("{} * {} = {}".format(i, j, i * j))
...
>>> tabuadas()
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
1 * 6 = 6
1 * 7 = 7
1 * 8 = 8
1 * 9 = 9
1 * 10 = 10
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
2 * 10 = 20
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
3 * 7 = 21
3 * 8 = 24
3 * 9 = 27
3 * 10 = 30
4 * 1 = 4
4 * 2 = 8
```

(continues on next page)

(continued from previous page)

```
4 * 3 = 12
4 * 4 = 16
4 * 5 = 20
4 * 6 = 24
4 * 7 = 28
4 * 8 = 32
4 * 9 = 36
4 * 10 = 40
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
6 * 1 = 6
6 * 2 = 12
6 * 3 = 18
6 * 4 = 24
6 * 5 = 30
6 * 6 = 36
6 * 7 = 42
6 * 8 = 48
6 * 9 = 54
6 * 10 = 60
7 * 1 = 7
7 * 2 = 14
7 * 3 = 21
7 * 4 = 28
7 * 5 = 35
7 * 6 = 42
7 * 7 = 49
7 * 8 = 56
7 * 9 = 63
7 * 10 = 70
8 * 1 = 8
8 * 2 = 16
8 * 3 = 24
8 * 4 = 32
8 * 5 = 40
8 * 6 = 48
8 * 7 = 56
8 * 8 = 64
8 * 9 = 72
8 * 10 = 80
9 * 1 = 9
9 * 2 = 18
9 * 3 = 27
9 * 4 = 36
9 * 5 = 45
9 * 6 = 54
9 * 7 = 63
9 * 8 = 72
9 * 9 = 81
```

(continues on next page)

(continued from previous page)

```

9 * 10 = 90
10 * 1 = 10
10 * 2 = 20
10 * 3 = 30
10 * 4 = 40
10 * 5 = 50
10 * 6 = 60
10 * 7 = 70
10 * 8 = 80
10 * 9 = 90
10 * 10 = 100

```

## 15.3 Exercícios

1. Faça uma função que determina se um número é par ou ímpar. Use o `%` para determinar o resto de uma divisão.  
Por exemplo:  $3 \% 2 = 1$  e  $4 \% 2 = 0$
2. Faça uma função que calcule a área de um círculo. Insira o raio como argumento.

**Dica:** faça a importação de `math` e use  $\pi$  de lá.

$$A = \pi R^2$$

3. Crie uma função que receba um valor de temperatura em Fahrenheit e transforme em Celsius.

Relembrar é viver:

$$\frac{C}{5} = \frac{F - 32}{9}$$

4. Utilizando a função anterior, faça a impressão da temperatura, em graus Celsius, de  $0^{\circ}\text{C}$  a  $100^{\circ}\text{C}$ , e todos os valores correspondentes em Fahrenheit.
5. Alanderson quer saber se um endereço IP é válido. Faça um programa para ajudar Alanderson a testar se um endereço é válido.

Para isso, a entrada deve ser um endereço IP (digitado pelo usuário) e o programa deve escrever na tela se é válido ou não. Um endereço IPv4 é composto por 4 números inteiros entre 0 e 255, separados por um ponto.

Por exemplo, o endereço 123.123.123.123 é válido, mas 666.123.k.3 não é.

6. Crie uma função que receba 3 valores e calcula as raízes da fórmula de Bhāskara.

$$x = \frac{-b \pm \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}$$

**Dica:** raiz quadrada é `sqrt()`, importando `math: math.sqrt()`

Faça um teste com `bhaskara(1, -4, -5)` e o programa deve obter as raízes: (5.0, -1.0)

7. Dada a função:  $y = 5x + 2$ , determine os valores de  $y$  para  $x$  entre -10 a +10, onde  $x$  é inteiro
8. Escreva uma função chamada `has_duplicates` que tome uma lista e retorne `True` se houver algum elemento que apareça mais de uma vez. Ela não deve modificar a lista original.
9. Duas palavras são um “par inverso” se uma for o contrário da outra. Escreva uma função que dado duas palavras, retorne `True` caso sejam.

10. Escreva uma função que imprime todos os números primos entre 1 e 50

**Dica:** um número é primo se ele for divisível apenas por 1 e ele mesmo, use o operador % (resto da divisão) para isso.

11. Duas palavras são anagramas se você puder soletrar uma rearranjando as letras da outra. Escreva uma função chamada `is_anagram` que tome duas strings e retorne `True` se forem anagramas ou `False` caso contrário.

12. Escreva uma função que retorne uma lista com todas as chaves de um dicionário que contém um certo valor.

Por exemplo, se o dicionário for `{'a': 1, 'b': 2, 'c': 1, 'd': 4}`, a função deve retornar `['a', 'c']` caso procure pelo valor 1; `[]` caso procure pelo valor 666.

13. Escreva uma função que dado um número, calcule o fatorial desse número. Por exemplo, fatorial de 5:

$$5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$$

14. Crie uma função que aproxima a função matemática seno, utilizando a seguinte equação:

$$\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

Essa é a expansão em *Série de Taylor* da função. Note que esta é uma série infinita! A sua função deve truncar a série em algum momento, ou seja, sua função vai calcular uma aproximação para o seno de um ângulo:

$$\sin(x) \approx \sum_{n=0}^N \frac{(-1)^n}{(2n+1)!} x^{2n+1} = \sum_{n=0}^N a_n = S_N$$

Note que, quanto maior o valor de N, melhor é a aproximação. Mas isso tem um custo: maior vai ser o número de termos nessa série e consequentemente, maior o tempo de execução desse código.

Uma possibilidade é estipular previamente uma *precisão* a ser atingida pelo código. Ou seja, definimos o desvio máximo  $\epsilon$  que nossa aproximação tem com relação ao valor exato! Isso é feito comparando dois termos consecutivos da série: se a diferença  $\delta$  entre eles (em valor absoluto!) for menor que  $\epsilon$ , atingimos a precisão desejada:

$$\delta = |S_N - S_{N-1}|$$

Implemente, então, uma função que receba como argumentos:

- $x$ : o ângulo (em radianos!!).
- $N_{\max}$ : o número máximo de iterações.
- $\epsilon$ : a precisão da aproximação.

e calcule uma aproximação para  $\sin(x)$  usando duas condições de parada: número máximo de termos na série é  $N_{\max}$  e precisão  $\epsilon$ . Ou seja, sua aproximação terá no máximo  $N_{\max}$  termos, mas pode ter menos termos caso a precisão desejada seja atingida ( $\delta < \epsilon$ ).

15. Calcule  $\pi$  usando um método de Monte Carlo.

Monte Carlo é uma classe de métodos para resolver problemas usando estatística. Aqui você vai implementar uma função usando um desses algoritmos para calcular o número  $\pi$ .

Dado um círculo de raio  $R$  dentro de um quadrado de lados  $2R$ , a razão entre a área do círculo para a área do quadrado é:

$$\frac{A_{\bigcirc}}{A_{\square}} = \frac{\pi R^2}{4R^2} = \frac{\pi}{4}$$

Ou seja, se você escolher aleatoriamente um ponto dentro do quadrado, a probabilidade dele cair dentro do círculo é de  $\pi/4$ . Se você escolher  $N$  pontos aleatórios dentro do quadrado, cerca de  $N\pi/4$  estarão dentro do círculo.

Então, basta escolher pontos aleatórios dentro do quadrado e ver se estão dentro do círculo

Um ponto  $(x, y)$  está dentro do círculo se  $x^2 + y^2 \leq R^2$ .

Faça uma função que receba como argumento um número  $N$  de pontos  $(x, y)$  (aleatórios) a serem sorteados. Dentro dessa função, você deve fazer um laço que sorteie esses  $N$  pontos e veja quantos estão dentro do círculo. Se  $M$  pontos caírem dentro do círculo, então a probabilidade de um ponto aleatório estar dentro do círculo é aproximadamente  $M/N$ . Então, podemos estimar  $\pi$  como:

$$\pi \approx \frac{4M}{N}$$

Para sortear um número aleatório entre  $a$  e  $b$  utilize a função `uniform(a, b)` do módulo `random`. Exemplo:

```
>>> import random
>>> random.uniform(1, 2) # número aleatório entre 1 e 2
1.8740445361226983
```

Perceba que ao executar a função `pi()` várias vezes seguidas, o resultado é sempre diferente. Então faça um laço para calcular `pi()`  $K$  vezes, salve os resultados em uma lista e calcule o valor médio e o desvio padrão.

16. O RNA é o responsável por levar as informações contidas no DNA para fora do núcleo da célula, para então ser feita a codificação para as bases púricas: *U*, *A*, *C* e *G*. Quando arranjadas em sequência de trincas (chamadas *códons*), formam um *polipeptídeo*, cadeia de aminoácido. O final de uma cadeia é determinado por um dos seguintes códons: UGA, UAA ou UAG.

A tabela a seguir mostra alguns códons e qual aminoácido codifica:

Códon RNA	Aminoácido
UAU	Ile
UGU	Thr
UUG	Asn
UCG	Ser
GUG	His
GCU	Arg
CAU	Val

Por exemplo:

DNA	->	TATTCGCATTGA                     ATAAGCGTAACCT
RNA	->	UAUUCGCAUUGA
Códons RNA	->	UAU UCG CAU UGA
Polipeptídeo	->	Ile-Ser-Val

Faça uma função cuja entrada seja uma string com tamanho múltiplo de 3 que representa o RNA. A saída deverá ser uma string com o nome de cada aminoácido separado por hífen, que representa o polipeptídeo.

**Dica:** faça um dicionário para trocar os códons por aminoácidos.

**Dica2:** faça uma lista para facilitar a saída.

Exemplos de entradas e saídas:

```
>>> polipeptideo("UAUCUCAUCAUUAUUCGUAG")
"Ile-Arg-Val-Val-Ile-Ser"

>>> polipeptideo("GCUUAUUCGCAUGCUUCGGCUGCUUAG")
"Arg-Ile-Ser-Val-Arg-Ser-Arg-Arg"

>>> polipeptideo("CAUUCGGUGGCUUCGGUGUGUCAUUCGCAUUAG")
"Val-Ser-His-Arg-Ser-His-Thr-Val-Ser-Val"

>>> polipeptideo("GCUAUUGUUGUUUGCAUUGUGUGGCGUGCAUUUGUAG")
"Arg-Val-Thr-Thr-Asn-Val-Thr-His-Arg-His-Val-Asn"
```

# CAPÍTULO 16

---

## Exercícios e Desafios!

---

Neste capítulo estão listados todos os exercícios apresentados no curso e também alguns desafios a mais!

### 16.1 Calculadora

#### 16.1.1 Operadores Matemáticos

1. Calcule o resto da divisão de 10 por 3.
2. Calcule a tabuada do 13.
3. Davinir não gosta de ir às aulas. Mas ele é obrigado a comparecer a pelo menos 75% delas. Ele quer saber quantas aulas pode faltar, sabendo que tem duas aulas por semana, durante quatro meses. Ajude o Davinir!

---

**Nota:** Um mês tem quatro semanas.

---

4. Calcule a área de um círculo de raio  $r = 2$ .

Lembrete: a área de um círculo de raio  $r$  é:

$$A_{\circ} = \pi r^2$$

#### 16.1.2 Expressões Numéricas

1. Quantos segundos há em 3 horas, 23 minutos e 17 segundos?
2. Se você correr 65 quilômetros em 3 horas, 23 minutos e 17 segundos, qual é a sua velocidade média em m/s?
3. Resolva essa expressão:

$$\frac{100 - 413 \cdot (20 - 5 \times 4)}{5}$$

4. Enivaldo quer ligar três capacitores, de valores:

- $C_1 = 10 \mu F$
- $C_2 = 22 \mu F$
- $C_3 = 6.8 \mu F$

Se ele ligar os três em paralelo, a capacidade resultante é a soma:

$$C_p = C_1 + C_2 + C_3$$

Se ele ligar os três em série, a capacidade resultante é:

$$\frac{1}{C_s} = \frac{1}{C_1} + \frac{1}{C_2} + \frac{1}{C_3}$$

Ou seja:

$$C_s = \frac{1}{\frac{1}{C_1} + \frac{1}{C_2} + \frac{1}{C_3}}$$

Qual é o valor resultante em cada um desses casos?

5. Você e os outros integrantes da sua república (Joca, Moacir, Demival e Jackson) foram no supermercado e compraram alguns itens:

- 75 latas de cerveja: R\$ 2,20 cada (da ruim ainda, pra fazer o dinheiro render)
- 2 pacotes de macarrão: R\$ 8,73 cada
- 1 pacote de Molho de tomate: R\$ 3,45
- 420g Cebola: R\$ 5,40/kg
- 250g de Alho: R\$ 30/kg
- 450g de pães franceses: R\$ 25/kg

Calcule quanto ficou para cada um.

6. Krissia gosta de bolinhas de queijo. Ela quer saber quantas bolinhas de queijo dá para colocar dentro de um pote de sorvete de 2 L. Ela pensou assim:

*Um pote de sorvete tem dimensões 15 cm x 10 cm x 13 cm. Uma bolinha de queijo é uma esfera de raio r = 1.2 cm. O fator de empacotamento ideal é 0.74, mas o pote de sorvete tem tamanho comparável às bolinhas de queijo, aí tem efeitos de borda, então o fator deve ser menor. Mas as bolinhas de queijo são razoavelmente elásticas, então empacota mais. Esse valor parece razoável.*

Sabendo que o volume de uma esfera de raio r é  $V = \frac{4}{3}\pi r^3$ , o volume do pote de sorvete é  $V = x \cdot y \cdot z$  e o fator de empacotamento é a fração de volume ocupado pelas bolinhas de queijo. Ou seja, 74% do pote de sorvete vai ser ocupado pelas bolinhas de queijo.

Ajude a Krissia descobrir quantas bolinhas de queijo cabem no pote de sorvete!

## 16.2 Variáveis

1. Supondo que a cotação do dólar esteja em R\$ 3,25, salve esse valor em uma variável e utilize-o para calcular quanto você teria ao cambiar R\$ 65,00 para dólares.

2. Abelindo é um professor muito malvado. Ele quer decidir como reprovar Rondinelly, que tirou 8.66, 5.35, 5 e 1, respectivamente, nas provas P1, P2, P3 e P4. Para isso, ele pode calcular a nota final usando média aritmética (M.A.), média geométrica (M.G.) ou média harmônica (M.H.).

$$M.A. = \frac{P_1 + P_2 + P_3 + P_4}{4}$$

$$M.G. = \sqrt[4]{|P_1 P_2 P_3 P_4|}$$

$$M.H. = \frac{4}{\frac{1}{P_1} + \frac{1}{P_2} + \frac{1}{P_3} + \frac{1}{P_4}}$$

Qual dessas médias dá a maior nota pra Rondinelly? E qual das médias dá a pior nota?

3. Josefson deseja fazer compras na China. Ela quer comprar um celular de USD 299,99, uma chaleira de USD 23,87, um gnomo de jardim de USD 66,66 e 6 adesivos de unicórnio de USD 1,42 cada um. O frete de tudo isso para a cidade de Rolândia, no Paraná, ficou em USD 12,34.

- a. Calcule o valor total da compra em dólares.
- b. Usando o mesmo valor do dólar do exercício anterior, calcule o preço final em Reais. Lembre-se que o valor do *IOF* é de 6,38 %.
- c. Quanto ela pagou apenas de *IOF*?

### **16.2.1 Desafios**

1. Joilson está aprendendo Arduino. Ele quer ligar LEDs nas saídas digitais do Arduino. Cada pino fornece 5 V. Joilson sabe que tem que ligar um resistor em série com o LED para não queimar. Calcule o valor do resistor que deve ser ligado para cada um desses LEDs, sabendo que a corrente de operação de cada um dos LEDs é de 20 mA:

- LED vermelho: opera em 2.0 V
- LED verde: opera em 3.2 V
- LED roxo: opera em 3.7 V

Lembre-se que a voltagem é a corrente multiplicada pela resistência:

$$V = RI$$

2. D3yver50n resolveu minerar criptomoedas. Ele decidiu minerar *Ethereum* e viu que 1 ETH = \$687.86 e \$1 = R\$3.59. Ele comprou o seguinte computador:

- 5 placas de vídeo: GTX1080 TI, cada uma por R\$5270,90
- 1 placa mãe: ASRock H110 Pro, por R\$920
- 1 fonte: 1600 W, por R\$2299,90
- 1 HD: 1 TB, SATA3, 7200 RPM por R\$208,90
- 2 pentes de memória: 4 GB, DDR4, 2400 MHZ, cada um por R\$259,90
- 1 CPU: Intel Core i5-8500 por R\$899,90

E resolveu montar usando uma estante de madeira e dois tijolos, para refrigerar melhor:

Essas GPUs (placas de vídeo) conseguem minerar Ethereum a uma taxa de  $\approx 27Mh/s$  (mega hash / s =  $10^6$  hash / s). Cada bloco minerado dá uma recompensa de 3 ETH. Considere a dificuldade da rede de  $3.29 \cdot 10^{15}$ , o *block time* médio de 15.44 s.



Para calcular quantos dólares por segundo ele vai ganhar com esse computador, D3yver50n fez as seguintes contas:

$$ETH/s = \text{cluster\_ratio} \frac{\text{recompensa}}{\text{block\_time}}$$

O cluster\_ratio é calculado como:

$$\text{cluster\_ratio} = n_{\text{GPU}} \frac{\text{GPU\_hashrate}}{\text{network\_hashrate}}$$

onde  $n_{\text{GPU}}$  é o número de placas de vídeo que ele tem. O network\_hashrate é calculado como:

$$\text{network\_hashrate} = \frac{\text{dificuldade}}{\text{block\_time}}$$

- a. Calcule quantos ETH por segundo D3yver50n vai ganhar com esse PC.
- b. Calcule quantos dólares por segundo ele vai ganhar.
- c. Calcule quanto ele vai pagar de energia elétrica por segundo para manter esse computador ligado, sabendo que o custo de energia elétrica é de 0.008centavos/ $kW$ .
- d. Após um mês, quantos ETH ele vai ganhar? Isso equivale a quantos reais? Quanto de energia elétrica ele vai gastar? Deu lucro ou prejuízo?
- e. Se ele teve lucro, após quanto tempo ele ganha o dinheiro que investiu no computador de volta?

## 16.3 Strings

1. Dada a frase Python é muito legal., use fatiamento para dar nome às variáveis contendo cada palavra. O resultado final deve ser:

```
>>> frase = "Python é muito legal."
# resolução do problema aqui
>>> palavra1
"Python"
>>> palavra2
"é"
>>> palavra3
"muito"
>>> palavra4
"legal"
```

2. Qual o tamanho dessa frase? E qual o tamanho de cada palavra?
3. Agora que conhecemos atribuição múltipla e o método `str.split()` refaça os dois exercícios anteriores usando essas técnicas.
4. Use *slicing* (mais especificamente o passo do fatiamento) para inverter a string «Python».

### 16.3.1 Desafios

1. Por que o seguinte código retorna uma *string* vazia?

```
>>> texto = "No alto da montanha havia uma rosa"
>>> texto[0:-1:-1]
''
```

**Nota:**

```
>>> texto = "No alto da montanha havia uma rosa"
>>> texto[0:-1]
'No alto da montanha havia uma ros'
```

## 16.4 Teclado

1. Leia um nome pelo teclado e imprima "Olá, <nome lido>!"
2. Leia outro nome pelo teclado e imprima:

```
<nome lido> roubou pão na cassa do <nome2 lido>!
<nome2 lido> ficou triste e com fome,
porque o bandejão estava fechado.
```

3. Leia uma frase pelo teclado e a imprima ao contrário.

Por exemplo, se a frase for "Manjo muito de Python!", a saída deverá ser '!nohtyP ed otium ojnaM'.

## 16.5 Listas

1. Crie uma lista com o nome das 3 pessoas mais próximas.

2. Crie três listas, uma lista de cada coisa a seguir:

- frutas
- docinhos de festa (não se esqueça de brigadeiros!!)
- ingredientes de feijoada

Lembre-se de salvá-las em alguma variável!

a. Agora crie uma lista que contém essas três listas.

Nessa lista de listas (vou chamar de *listona*):

- b. você consegue acessar o elemento *brigadeiro*?
- c. Adicione mais *brigadeiros* à segunda lista de listona. O que aconteceu com a lista de *docinhos de festa*?
- d. Adicione bebidas ao final da *listona*, mas sem criar uma lista!

3. Utilizando o `del`, remova todos os elementos da lista criada anteriormente até a lista ficar vazia.

4. Faça uma lista de compras do mês, não se esqueça de comprar produtos de limpeza e sorvete!

Agora «vá ao mercado» e delete apenas os produtos de limpeza da lista.

Agora «vá à sorveteria» e se empanturre de sorvete e tire o sorvete da lista.

5. Dado uma lista de números, faça com que os números sejam ordenados e, em seguida, inverta a ordem da lista usando *slicing*.

## 16.6 Dicionários

1. Faça um dicionário com as 5 pessoas mais perto de você, tendo o nome como chave e a cor da camisa que está usando como valor.
2. Crie um dicionário vazio `semana = {}` e o complete com uma chave para cada dia da semana, tendo como seu valor uma lista com as aulas que você tem nesse dia (sábado e domingo recebem listas vazias, ou você tem aula?).
3. Crie um dicionário vazio `filmes = {}`. Utilize o nome de um filme como chave. E, como valor, *outro* dicionário contendo o vilão e o ano em que o filme foi lançado. Preencha 5 filmes.

## 16.7 Estruturas de Controle

1. Escreva um programa que, dados 2 números diferentes (a e b), encontre o menor deles.

2. Para doar sangue é necessário<sup>1</sup>:

- Ter entre 16 e 69 anos.
- Pesar mais de 50 kg.
- Estar descansado (ter dormido pelo menos 6 horas nas últimas 24 horas).

Faça um programa que pergunte a idade, o peso e quanto dormiu nas últimas 24 h para uma pessoa e diga se ela pode doar sangue ou não.

---

<sup>1</sup> Para mais informações sobre doação de sangue, acesse [http://www.prosangue.sp.gov.br/artigos/requisitos\\_basicos\\_para\\_doacao.html](http://www.prosangue.sp.gov.br/artigos/requisitos_basicos_para_doacao.html)

3. Considere uma equação do segundo grau  $f(x) = a \cdot x^2 + b \cdot x + c$ . A partir dos coeficientes, determine se a equação possui duas raízes reais, uma, ou se não possui.  
**Dica:**  $\Delta = b^2 - 4 \cdot a \cdot c$ : se delta é maior que 0, possui duas raízes reais; se delta é 0, possui uma raiz; caso delta seja menor que 0, não possui raiz real
4. Leia dois números e efetue a adição. Caso o valor somado seja maior que 20, este deverá ser apresentado somando-se a ele mais 8; caso o valor somado seja menor ou igual a 20, este deverá ser apresentado subtraindo-se 5.
5. Leia um número e imprima a raiz quadrada do número caso ele seja positivo ou igual a zero e o quadrado do número caso ele seja negativo.
6. Leia um número inteiro entre 1 e 12 e escreva o mês correspondente. Caso o usuário digite um número fora desse intervalo, deverá aparecer uma mensagem informando que não existe mês com este número.

### 16.7.1 Desafios

1. Escreva um programa que, dados 3 números diferentes (a, b e c), encontre o menor deles.
2. Dado 3 valores inteiros lidos do teclado: A, B e C, retorne a soma deles. Porém, caso algum desses valores seja 13, então ele não conta para a soma, e os valores a sua direita também não.

**Por exemplo:**

1, 2, 3 -> 6  
1, 2, 13 -> 3  
1, 13, 3 -> 1  
13, 2, 3 -> 0

## 16.8 Estruturas de repetição

1. Calcule a tabuada do 13.
2. Ler do teclado uma lista com 5 inteiros e imprimir o menor valor.
3. Ler do teclado uma lista com 5 inteiros e imprimir True se a lista estiver ordenada de forma crescente ou False caso contrário.
4. Exiba em ordem decrescente todos os números de 500 até 10.
5. Ler do teclado 10 números e imprima a quantidade de números entre 10 e 50.
6. Ler do teclado a idade e o sexo de 10 pessoas, calcule e imprima:
  - a) idade média das mulheres
  - b) idade média dos homens
  - c) idade média do grupo
7. Calcule o somatório dos números de 1 a 100 e imprima o resultado.

## 16.9 Funções

1. Faça uma função que determina se um número é par ou ímpar. Use o % para determinar o resto de uma divisão.  
Por exemplo:  $3 \% 2 = 1$  e  $4 \% 2 = 0$

2. Faça uma função que calcule a área de um círculo. Insira o raio como argumento.

**Dica:** faça a importação de `math` e use  $\pi$  de lá.

$$A = \pi R^2$$

3. Crie uma função que receba um valor de temperatura em Fahrenheit e transforme em Celsius.

Relembre é viver:

$$\frac{C}{5} = \frac{F - 32}{9}$$

4. Utilizando a função anterior, faça a impressão da temperatura, em graus Celsius, de 0 °C a 100 °C, e todos os valores correspondentes em Fahrenheit.
5. Alanderson quer saber se um endereço IP é válido. Faça um programa para ajudar Alanderson a testar se um endereço é válido.

Para isso, a entrada deve ser um endereço IP (digitado pelo usuário) e o programa deve escrever na tela se é válido ou não. Um endereço IPv4 é composto por 4 números inteiros entre 0 e 255, separados por um ponto.

Por exemplo, o endereço 123.123.123.123 é válido, mas 666.123.k.3 não é.

6. Crie uma função que receba 3 valores e calcula as raízes da fórmula de Bhāskara.

$$x = \frac{-b \pm \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}$$

**Dica:** raiz quadrada é `sqrt()`, importando `math: math.sqrt()`

Faça um teste com `bhaskara(1, -4, -5)` e o programa deve obter as raízes: (5.0, -1.0)

7. Dada a função:  $y = 5x + 2$ , determine os valores de  $y$  para  $x$  entre -10 a +10, onde  $x$  é inteiro
  8. Escreva uma função chamada `has_duplicates` que tome uma lista e retorne `True` se houver algum elemento que apareça mais de uma vez. Ela não deve modificar a lista original.
  9. Duas palavras são um “par inverso” se uma for o contrário da outra. Escreva uma função que dado duas palavras, retorne `True` caso sejam.
  10. Escreva uma função que imprime todos os números primos entre 1 e 50
- Dica:** um número é primo se ele for divisível apenas por 1 e ele mesmo, use o operador `%` (resto da divisão) para isso.
11. Duas palavras são anagramas se você puder soletrar uma rearranjando as letras da outra. Escreva uma função chamada `is_anagram` que tome duas strings e retorne `True` se forem anagramas ou `False` caso contrário.
  12. Escreva uma função que retorne uma lista com todas as chaves de um dicionário que contém um certo valor.
- Por exemplo, se o dicionário for `{'a': 1, 'b': 2, 'c': 1, 'd': 4}`, a função deve retornar `['a', 'c']` caso procure pelo valor 1; `[]` caso procure pelo valor 666.
13. Escreva uma função que dado um número, calcule o fatorial desse número. Por exemplo, fatorial de 5:

$$5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$$

14. Crie uma função que aproxima a função matemática seno, utilizando a seguinte equação:

$$\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

Essa é a expansão em *Série de Taylor* da função. Note que esta é uma série infinita! A sua função deve truncar a série em algum momento, ou seja, sua função vai calcular uma aproximação para o seno de um ângulo:

$$\sin(x) \approx \sum_{n=0}^N \frac{(-1)^n}{(2n+1)!} x^{2n+1} = \sum_{n=0}^N a_n = S_N$$

Note que, quanto maior o valor de  $N$ , melhor é a aproximação. Mas isso tem um custo: maior vai ser o número de termos nessa série e consequentemente, maior o tempo de execução desse código.

Uma possibilidade é estipular previamente uma *precisão* a ser atingida pelo código. Ou seja, definimos o desvio máximo  $\epsilon$  que nossa aproximação tem com relação ao valor exato! Isso é feito comparando dois termos consecutivos da série: se a diferença  $\delta$  entre eles (em valor absoluto!) for menor que  $\epsilon$ , atingimos a precisão desejada:

$$\delta = |S_N - S_{N-1}|$$

Implemente, então, uma função que receba como argumentos:

- $x$ : o ângulo (em radianos!!).
- $N_{\max}$ : o número máximo de iterações.
- $\epsilon$ : a precisão da aproximação.

e calcule uma aproximação para  $\sin(x)$  usando duas condições de parada: número máximo de termos na série é  $N_{\max}$  e precisão  $\epsilon$ . Ou seja, sua aproximação terá no máximo  $N_{\max}$  termos, mas pode ter menos termos caso a precisão desejada seja atingida ( $\delta < \epsilon$ ).

15. Calcule  $\pi$  usando um método de Monte Carlo.

Monte Carlo é uma classe de métodos para resolver problemas usando estatística. Aqui você vai implementar uma função usando um desses algoritmos para calcular o número  $\pi$ .

Dado um círculo de raio  $R$  dentro de um quadrado de lados  $2R$ , a razão entre a área do círculo para a área do quadrado é:

$$\frac{A_{\bigcirc}}{A_{\square}} = \frac{\pi R^2}{4R^2} = \frac{\pi}{4}$$

Ou seja, se você escolher aleatoriamente um ponto dentro do quadrado, a probabilidade dele cair dentro do círculo é de  $\pi/4$ . Se você escolher  $N$  pontos aleatórios dentro do quadrado, cerca de  $N\pi/4$  estarão dentro do círculo.

Então, basta escolher pontos aleatórios dentro do quadrado e ver se estão dentro do círculo

Um ponto  $(x, y)$  está dentro do círculo se  $x^2 + y^2 \leq R^2$ .

Faça uma função que receba como argumento um número  $N$  de pontos  $(x, y)$  (aleatórios) a serem sorteados. Dentro dessa função, você deve fazer um laço que sorteie esses  $N$  pontos e veja quantos estão dentro do círculo. Se  $M$  pontos caírem dentro do círculo, então a probabilidade de um ponto aleatório estar dentro do círculo é aproximadamente  $M/N$ . Então, podemos estimar  $\pi$  como:

$$\pi \approx \frac{4M}{N}$$

Para sortear um número aleatório entre  $a$  e  $b$  utilize a função *uniform(a, b)* do módulo *random*. Exemplo:

```
>>> import random
>>> random.uniform(1, 2) # número aleatório entre 1 e 2
1.8740445361226983
```

Perceba que ao executar a função `pi()` várias vezes seguidas, o resultado é sempre diferente. Então faça um laço para calcular `pi()`  $K$  vezes, salve os resultados em uma lista e calcule o valor médio e o desvio padrão.

16. O RNA é o responsável por levar as informações contidas no DNA para fora do núcleo da célula, para então ser feita a codificação para as bases púricas: *U*, *A*, *C* e *G*. Quando arranjadas em sequência de trincas (chamadas *códons*), formam um *polipeptídeo*, cadeia de aminoácido. O final de uma cadeia é determinado por um dos seguintes códons: UGA, UAA ou UAG.

A tabela a seguir mostra alguns códons e qual aminoácido codifica:

Códon RNA	Aminoácido
UAU	Ile
UGU	Thr
UUG	Asn
UCG	Ser
GUG	His
GCU	Arg
CAU	Val

Por exemplo:

DNA	->	TATTCGCATTGA                     ATAAGCGTAACT
RNA	->	UAUUCGCAUUGA
Códons RNA	->	UAU UCG CAU UGA
Polipeptídeo	->	Ile-Ser-Val

Faça uma função cuja entrada seja uma string com tamanho múltiplo de 3 que representa o RNA. A saída deverá ser uma string com o nome de cada aminoácido separado por hífen, que representa o polipeptídeo.

**Dica:** faça um dicionário para trocar os códons por aminoácidos.

**Dica2:** faça uma lista para facilitar a saída.

Exemplos de entradas e saídas:

```
>>> polipeptideo("UAUGCUCAUCAUUAUUCGUAG")
"Ile-Arg-Val-Val-Ile-Ser"

>>> polipeptideo("GCUUAUUCGCAUGCUUCGGCUGCUUAG")
"Arg-Ile-Ser-Val-Arg-Ser-Arg-Arg"

>>> polipeptideo("CAUUCGGUGGCUCGGUGUGUCAUUCGCAUUAG")
"Val-Ser-His-Arg-Ser-His-Thr-Val-Ser-Val"

>>> polipeptideo("GCUCAUUGUUGUUUGCAUUGUGUGGCUGUGCAUUUGUAG")
"Arg-Val-Thr-Thr-Asn-Val-Thr-His-Arg-His-Val-Asn"
```

# CAPÍTULO 17

---

## Contribuidores

---

Lista de pessoas que contribuíram com a criação deste material:

- Alex Soares Prestes
- Guilherme Martins
- Heitor de Bittencourt
- Juliana Karoline
- Lucas Carvalho
- Luiz Menezes
- Luiz Fernando S. E. Santos
- Marcelo Miky Mine
- Tiago Martins
- Víctor Cora Colombo