Curso Introdutório de Python

Versão 2.3.2

Grupy-Sanca

14 ago, 2018

Conteúdo:

@chappos

CAPÍTULO 1

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, dentre eles:

1.1 Cursos e oficinas de programação

Oferecemos cursos básicos de Python e também sobre alguns assuntos específicos.

Fig. 1: Nosso primeiro 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!

Fig. 2: Oficina de interfaces gráficas em Python :)

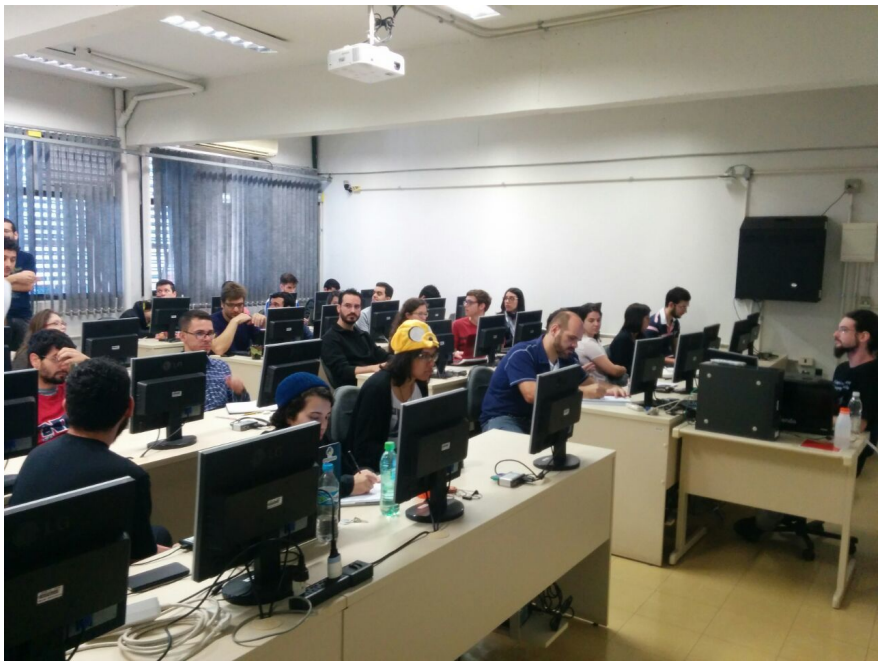
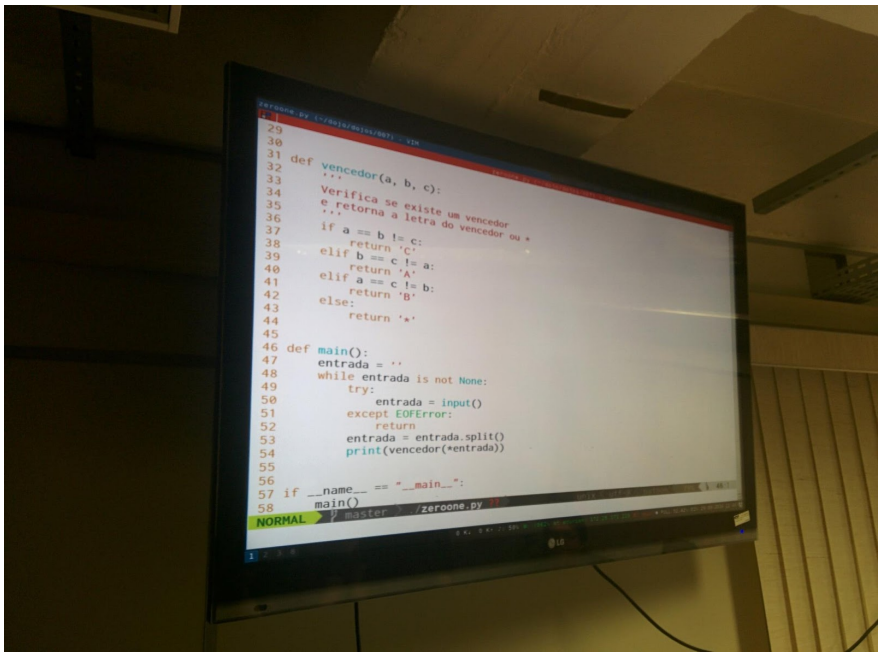


Fig. 3: Curso de Python básico no IFSC!
Realizado em 01 de Julho de 2017, no IFSC - USP - São Carlos. Tivemos ~100 inscritos! 38 participantes! 2 ministrantes! 3 monitores!! 2 *coffee-breaks*! 2 garrafas térmicas com apenas café :P

Fig. 4: Curso de Python básico na UNESP de Rio Claro!!

1.2 Pylestras

1.3 Coding Dojos


def vencedor(a, b, c):
'''
Verifica se existe um vencedor
e retorna a letra do vencedor ou *
'''
if a == b != c:
return 'C'
elif b == c != a:
return 'A'
elif a == c != b:
return 'B'
else:
return '*'
def main():
entrada = ''
while entrada is not None:
try:
entrada = input()
except EOFError:
return
entrada = entrada.split()
print(vencedor(*entrada))
if __name__ == "__main__":
main()
NORMAL
master
zeroone.py

1.4 Eventos

Fig. 5: Arduino Day em 2017

Fig. 6: Curso de Arduino no *Software Freedom Day* em 2017 no IFSC - USP

1.5 Sprints

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

Fig. 7: *Hacktoberfest* de 2017 :)
Fizemos cerca de 50 *commits* esse dia o/


I
pyladies

1.6 Encontros casuais

Também conhecidos como PyBares :)

A idéia 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*!

1.7 História

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

- 18+ Coding Dojos

- 20+ PyBares
- 15+ Cursos e workshops
- 6 PyLestras
- 3 Eventos
- 2 Sprints

Além disso, chegamos nos 942+ inscritos no Meetup!

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

- Site oficial[2]

  www.grupysanca.com.br

- Facebook[3]

  facebook.com/grupysanca

- Meetup[4]

  meetup.com/grupy-sanca

- Telegram[5]

  t.me/grupysanca

[2] http://www.grupysanca.com.br
[3] https://www.facebook.com/grupysanca/
[4] https://www.meetup.com/grupy-sanca
[5] https://t.me/grupysanca

CAPÍTULO 2

Guia de Instalação do Python

2.1 Linux - Ubuntu

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:

```
$ sudo apt install python3.6
```

2.2 Mac OS X

Obtenha o instalador na sessão de downloads para Mac OS X do Python[6]. 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
```

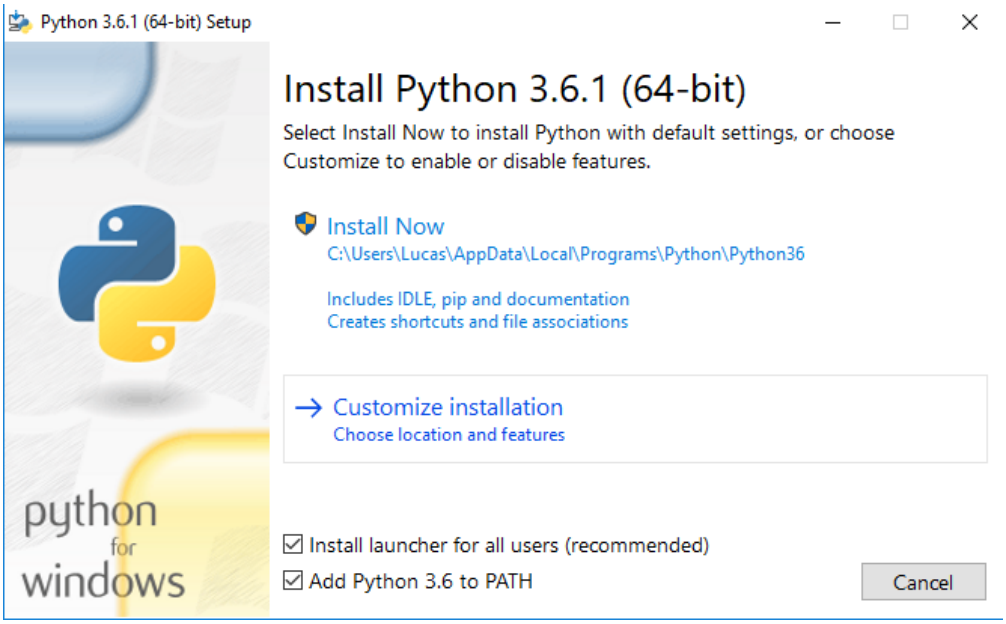
[6] https://www.python.org/downloads/mac-osx/

2.3 Windows

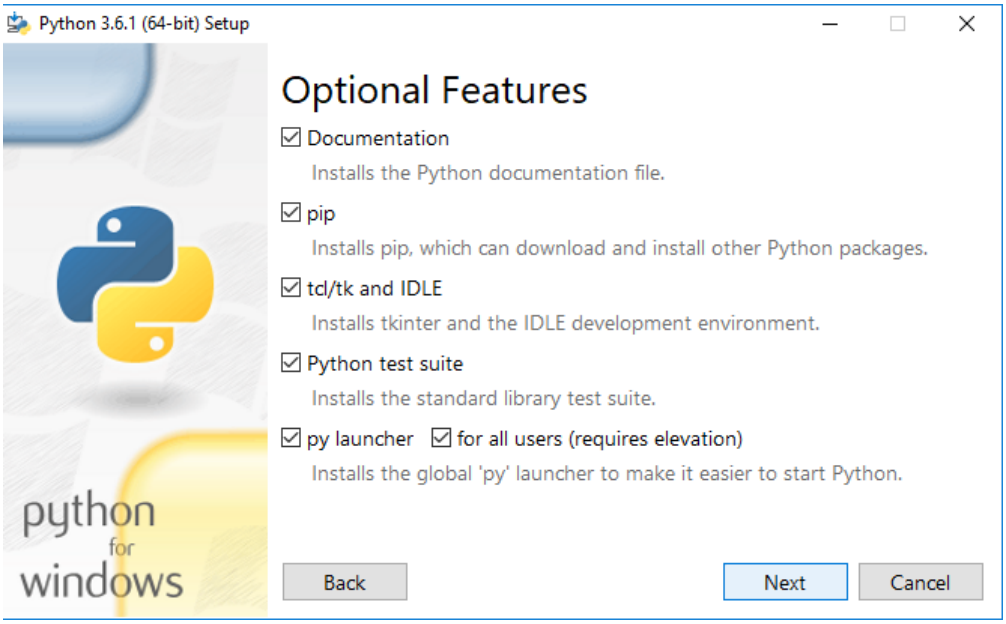
Primeiramente, deve-se obter o arquivo de instalação compatível com a arquitetura. O arquivo x86 provavelmente funcionará para todos computadores, os links estão a seguir:

- x86_64[7]
- x86[8]

A seguir, o aquivo deve ser executado e a seguinte imagem aparecerá:

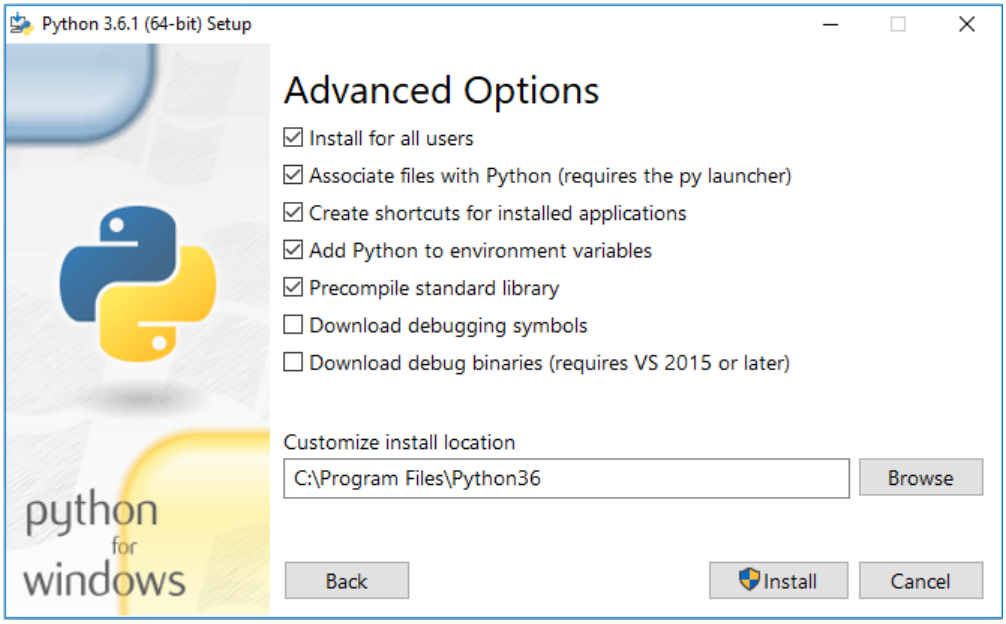


Deve ser selecionado a opção *Add Python 3.6 to PATH*, e deve ser clicado na opção "Customize installation"

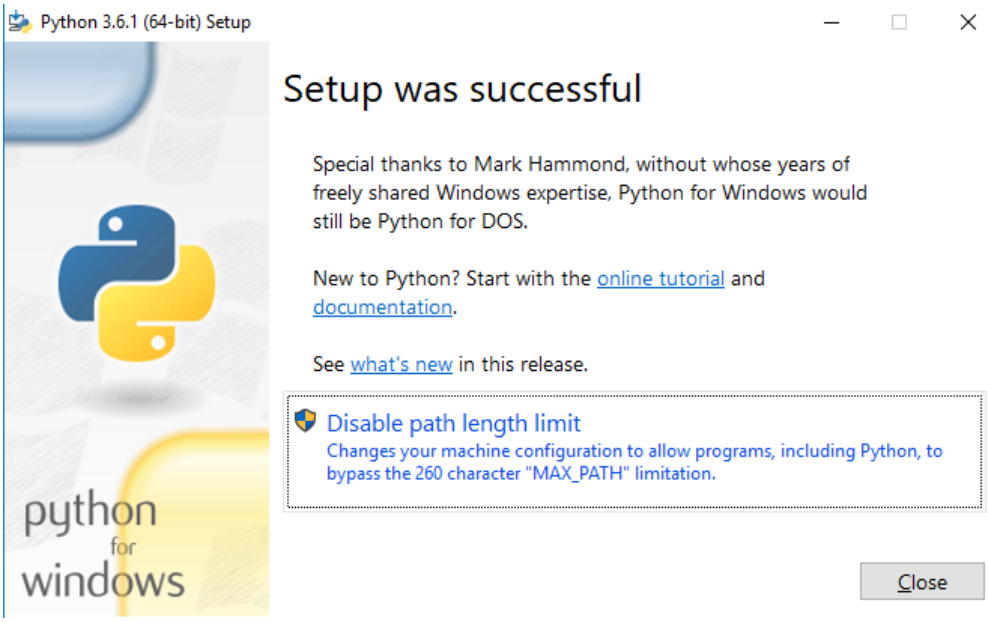


[7] https://www.python.org/ftp/python/3.6.5/python-3.6.5-amd64.exe
[8] https://www.python.org/ftp/python/3.6.5/python-3.6.5.exe

Clique em "Next"



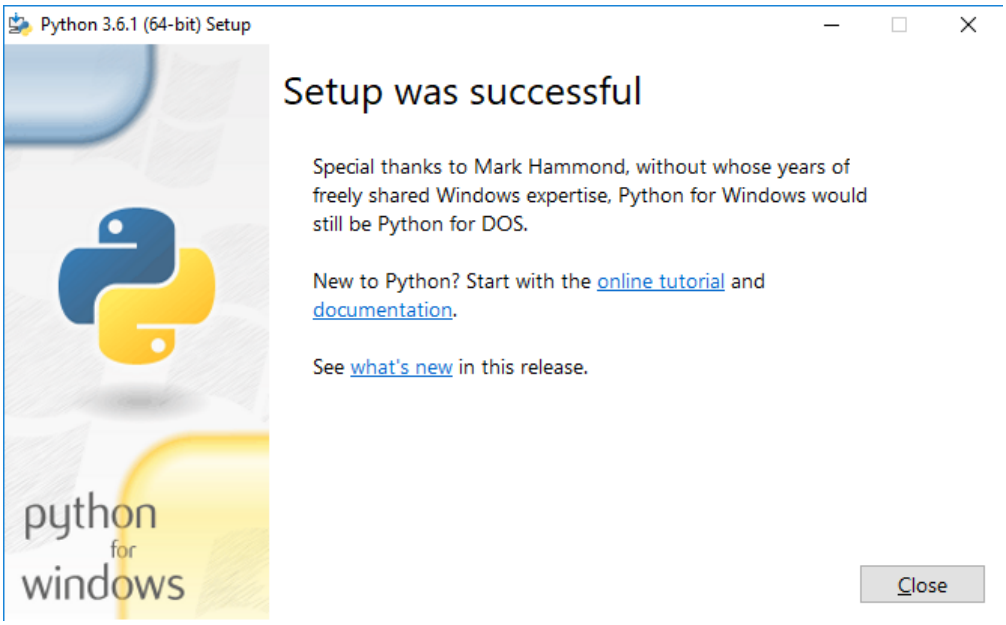
Selecionar a opção *Install for all users* e clicar em *Install*. Então, uma janela pedindo permissão de administrador deve aparecer, é necessário aceitar



Selecione *Disable path lenght limit* e, novamente, uma janela de permissão de administrador irá aparecer, é necessário aceitar

Clique em *Close*

Parabéns, agora o Python está intalado em sua máquina!


Python 3.6.1 (64-bit) Setup
Setup was successful
Special thanks to Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.
New to Python? Start with the online tutorial and documentation.
See what's new in this release.
python
for
windows
Close

CAPÍTULO 3

Ambientes de Desenvolvimento

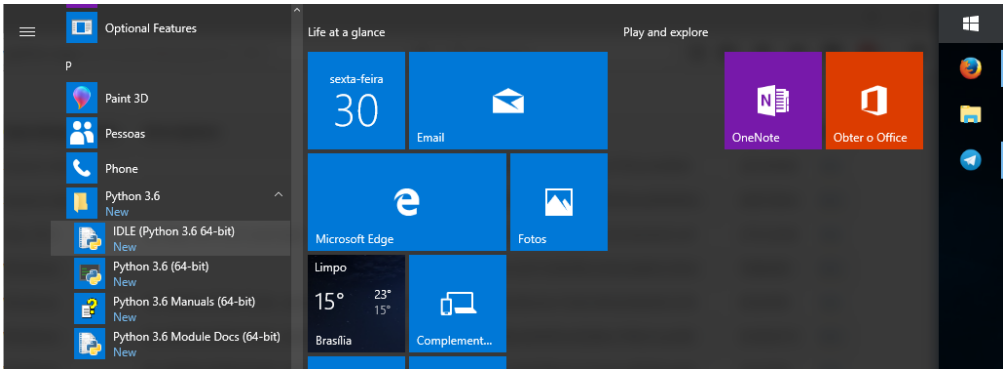
Há diversos programas para desenvolvermos códigos, alguns são mais bonitinhos, outros são mais poderosos, alguns são mais simples, outros são mais amigaveis. Dê uma olhada nesta seção e escolha o que você achar mais interessante. Somente você pode responder à pergunta «Qual o melhor ambiente de desenvolvimento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

3.1 Linha de comando

3.2 IDE

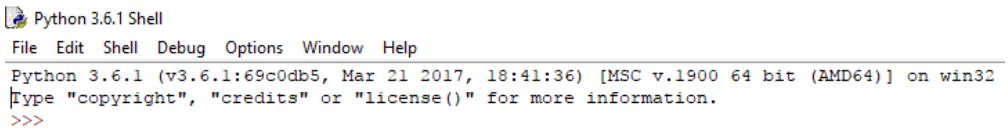
3.2.1 IDLE

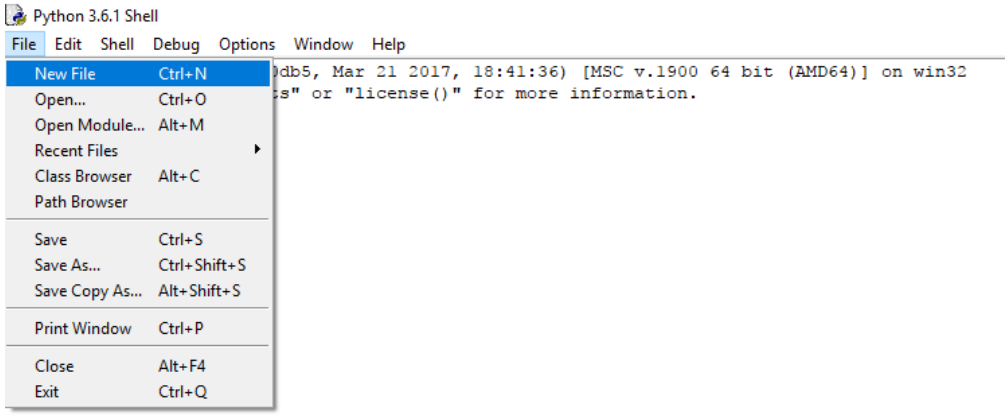
Para usuários Windows é recomendado utilizar o *IDLE*. Ele é composto pelo interpretador do Python e um editor de texto para criar programas. Após seguir o *Guia de Instalação do Python* (page 15), o menu inicial deve estar da seguinte forma:



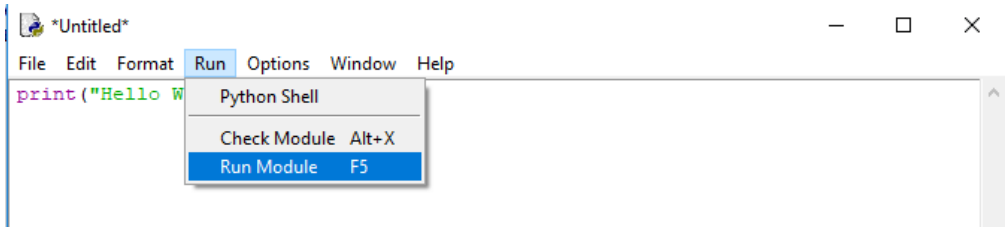
Ao abrir o *IDLE (Python 3.X)*, aparecerá uma janela como na imagem abaixo:

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)



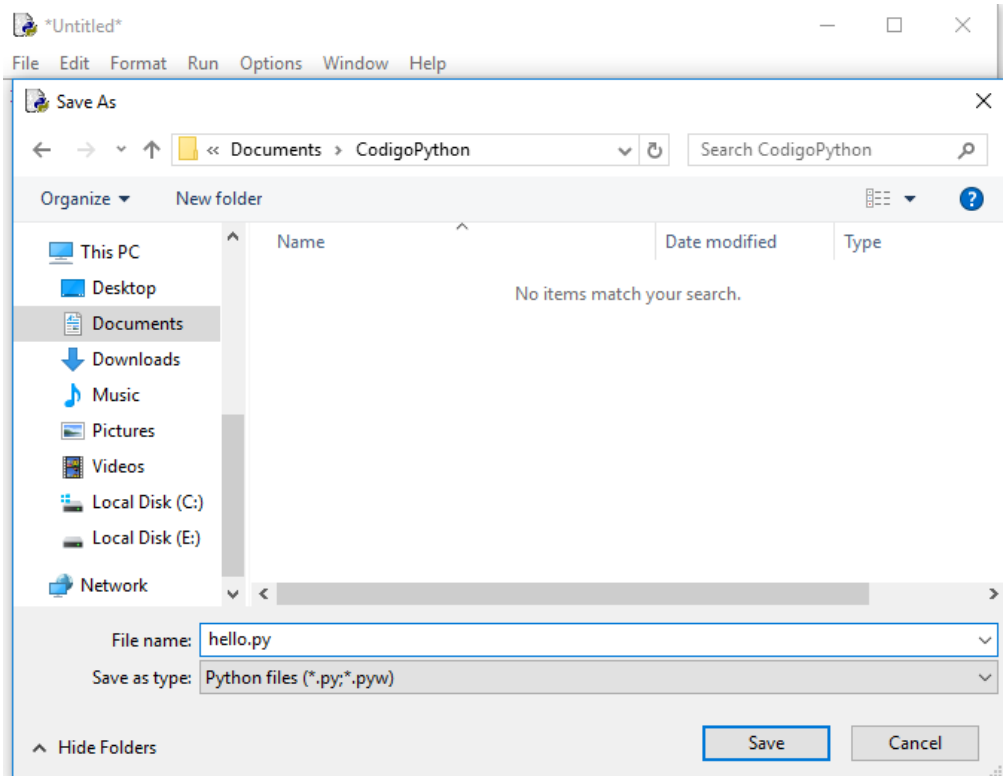


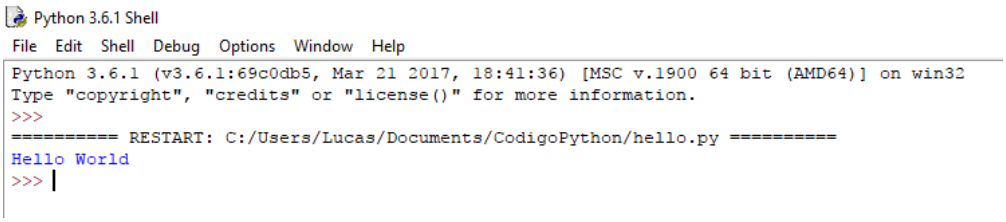
Para rodar um programa, clique em *Run -> Run Module* (Ou aperte a tecla *F5*)



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*:


Untitled
File Edit Format Run Options Window Help
Save As
Documents › CodigoPython
Search CodigoPython
Organize
New folder
This PC
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
Local Disk (E:)
Network
Name
Date modified
Type
No items match your search.
File name: hello.py
Save as type: Python files (*.py;*.pyw)
Hide Folders
Save
Cancel


Python 3.6.1 Shell
File Edit Shell Debug Options Window Help
Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 18:41:36) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Users/Lucas/Documents/CodigoPython/hello.py ==========
Hello World
>>>

CAPÍTULO 4

Introdução

4.1 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 linguangens de programação implementam esse exemplo:

4.1.1 C

```c
#include <stdio.h>

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

4.1.2 Java

```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*

4.1.3 Pascal

```
program HelloWorld;

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

4.1.4 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.

4.2 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 :)

4.2.1 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 áspas 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 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 carácter especial, a *quebra de linha*: *n*. Esse *n* é um caracter especial que significa *aqui acaba a linha, o que vier depois deve ficar na linha de baixo*. Por exemplo:

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

CAPÍTULO 5

Python como calculadora

5.1 Operadores matemáticos

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

5.1.1 Soma (+)

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

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

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

5.1.2 Subtração (−)

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

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

5.1.3 Multiplicação (∗)

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

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

5.1.4 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: divison by zero»*.

5.1.5 Divisão inteira (//)

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

5.1.6 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
```

5.1.7 Raíz 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 π?

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

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

5.2 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!

   obs: 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:

$$A_\circ = \pi\, r^2$$

5.3 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. **P** arênteses
2. **E** xponenciação
3. **M** ultiplicação e **D** ivisão (mesma precedência)
4. **A** dição e **S** subtração (mesma precedência)

5.4 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
```

5.5 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. Rondinelly 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 capacitância resultante é a soma:

$$C_p = C_1 + C_2 + C_3$$

Se ele ligar os três em série, a capacitância 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!

5.6 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 caracter `#`. 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.
```

5.7 Comparações

Os operadores de comparação em Python são:

Operação	Significado
<	menor que
<=	menor igual que
>	maior que
<=	maior igual que
==	igual
!=	diferente

```
>>> 2 < 10
True
>>> 2 > 11
False
>>> 10 > 10
False
>>> 10 >= 10
True
>>> 42 == 24
False
>>> 666 != 137
True
>>> 8**2 == 60 + 4
True
>>> 100 != 99 + 3
True
```

CAPÍTULO 6

Variáveis

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

6.1 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 atribuimos um número inteiro à variável de nome `numero`; no segundo uma frase à variável `frase`; no último um número de ponto flutuante à `pi`.

6.2 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.

```
>>> crieiumavariavelcomnomegiganteeeestoucompreguiçadeescrevertudodenovo = 10
>>> crieiumavariavelcomnomegiganteeeestoucompreguiçadeescrevertudodenovo # 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 caracter 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)

(continuação da página anterior)

```
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 atribuimos 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
```

6.3 Exercicios

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

2. Abelindo é um professor muito malvado. Ele quer decidir como reprovar Rondineli, 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 Rondineli? E qual das médias dá a pior nota?

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

   (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*?

6.4 Strings (sequência de caracteres)

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

```
>>> "Texto bonito"
'Texto bonito'
>>> "Texto coma centos de cedilhas: hoje é dia de caça!"
'Texto coma centos de cedilhas: hoje é dia de caça!'
```

As *strings* aceitam aspas simples também:

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

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 aí, '
>>> canto2 = 'lá '
>>> 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\ne em outra\ne mais uma\ne 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.
```

6.5 Tipos de objetos

Para saber o tipo de um objeto, 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'>
```

6.6 Tamanho

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

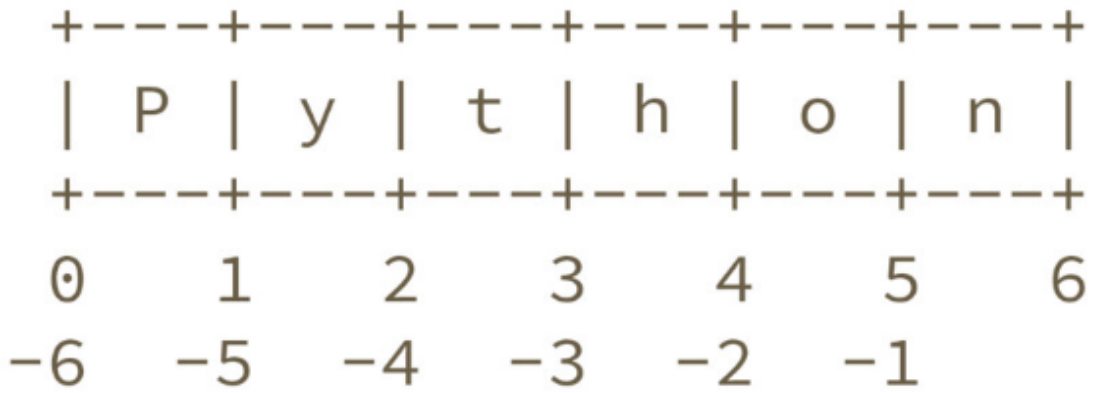
```
>>> len('Abracadabra')
11
>>> palavras = 'Faz um pull request lá'
>>> len(palavras)
22
```

6.7 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 caracter de dentro dessas estruturas, deve-se utilzar o acesso por índices, no qual o índice entre colchetes [] representa a posição do elemento que deseja-se 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'
```

6.8 Fatias

Se ao invés de obter apenas um elemento de uma estrutura (string, lista, ...) deseja-se obter multiplos 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 proximo índice do último elemento desejado, tudo entre chaves.

```
>>> frase = "Aprender Python é muito divertido!"
>>> frase[0:5] # do zero até o 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 indice, 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[2:-2]
'render Python é muito divertid'
>>> frase[2:-2:2] # pode-se ecolher o passo com que o slice é feito
'rne yhnémiodvri'
```

6.9 Buscando ajuda rapidamente

Está com dúvida em alguma coisa? Use a função help()!

```
>>> 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
```

(continues on next page)

(continuação da página anterior)

```
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.
```

```
>>> help(str)
Help on class str in module builtins:

class str(object)
 |  str(object='') -> str
 |  str(bytes_or_buffer[, encoding[, errors]]) -> str
 |
 |  Create a new string object from the given object. If encoding or
 |  errors is specified, then the object must expose a data buffer
 |  that will be decoded using the given encoding and error handler.
 |  Otherwise, returns the result of object.__str__() (if defined)
 |  or repr(object).
 |  encoding defaults to sys.getdefaultencoding().
 |  errors defaults to 'strict'.
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __format__(...)
 |      S.__format__(format_spec) -> str
 |
 |      Return a formatted version of S as described by format_spec.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
```

(continues on next page)

(continuação da página anterior)

```
|      Return getattr(self, name).
|
|  __getitem__(self, key, /)
|      Return self[key].
|
|  __getnewargs__(...)
|
|  __gt__(self, value, /)
|      Return self>value.
|
|  __hash__(self, /)
|      Return hash(self).
|
|  __iter__(self, /)
|      Implement iter(self).
|
|  __le__(self, value, /)
|      Return self<=value.
|
|  __len__(self, /)
|      Return len(self).
|
|  __lt__(self, value, /)
|      Return self<value.
|
|  __mod__(self, value, /)
|      Return self%value.
|
|  __mul__(self, value, /)
|      Return self*value.n
|
|  __ne__(self, value, /)
|      Return self!=value.
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __rmod__(self, value, /)
|      Return value%self.
|
|  __rmul__(self, value, /)
|      Return self*value.
|
|  __sizeof__(...)
|      S.__sizeof__() -> size of S in memory, in bytes
|
|  __str__(self, /)
|      Return str(self).
|
|  capitalize(...)
|      S.capitalize() -> str
|
|      Return a capitalized version of S, i.e. make the first character
|      have upper case and the rest lower case.
```

(continues on next page)

(continuação da página anterior)

```
|
|  casefold(...)
|      S.casefold() -> str
|
|      Return a version of S suitable for caseless comparisons.
|
|  center(...)
|      S.center(width[, fillchar]) -> str
|
|      Return S centered in a string of length width. Padding is
|      done using the specified fill character (default is a space)
|
|  count(...)
|      S.count(sub[, start[, end]]) -> int
|
|      Return the number of non-overlapping occurrences of substring sub in
|      string S[start:end].  Optional arguments start and end are
|      interpreted as in slice notation.
|
|  encode(...)
|      S.encode(encoding='utf-8', errors='strict') -> bytes
|
|      Encode S using the codec registered for encoding. Default encoding
|      is 'utf-8'. errors may be given to set a different error
|      handling scheme. Default is 'strict' meaning that encoding errors raise
|      a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and
|      'xmlcharrefreplace' as well as any other name registered with
|      codecs.register_error that can handle UnicodeEncodeErrors.
|
|  endswith(...)
|      S.endswith(suffix[, start[, end]]) -> bool
|
|      Return True if S ends with the specified suffix, False otherwise.
|      With optional start, test S beginning at that position.
|      With optional end, stop comparing S at that position.
|      suffix can also be a tuple of strings to try.
|
|  expandtabs(...)
|      S.expandtabs(tabsize=8) -> str
|
|      Return a copy of S where all tab characters are expanded using spaces.
|      If tabsize is not given, a tab size of 8 characters is assumed.
|
|  find(...)
|      S.find(sub[, start[, end]]) -> int
|
|      Return the lowest index in S where substring sub is found,
|      such that sub is contained within S[start:end].  Optional
|      arguments start and end are interpreted as in slice notation.
|
|      Return -1 on failure.
|
|  format(...)
|      S.format(*args, **kwargs) -> str
|
|      Return a formatted version of S, using substitutions from args and kwargs.
|      The substitutions are identified by braces ('{' and '}').
```

(continues on next page)

(continuação da página anterior)

```
|
|  format_map(...)
|      S.format_map(mapping) -> str
|
|      Return a formatted version of S, using substitutions from mapping.
|      The substitutions are identified by braces ('{' and '}').
|
|  index(...)
|      S.index(sub[, start[, end]]) -> int
|
|      Like S.find() but raise ValueError when the substring is not found.
|
|  isalnum(...)
|      S.isalnum() -> bool
|
|      Return True if all characters in S are alphanumeric
|      and there is at least one character in S, False otherwise.
|
|  isalpha(...)
|      S.isalpha() -> bool
|
|      Return True if all characters in S are alphabetic
|      and there is at least one character in S, False otherwise.
|
|  isdecimal(...)
|      S.isdecimal() -> bool
|
|      Return True if there are only decimal characters in S,
|      False otherwise.
|
|  isdigit(...)
|      S.isdigit() -> bool
|
|      Return True if all characters in S are digits
|      and there is at least one character in S, False otherwise.
|
|  isidentifier(...)
|      S.isidentifier() -> bool
|
|      Return True if S is a valid identifier according
|      to the language definition.
|
|      Use keyword.iskeyword() to test for reserved identifiers
|      such as "def" and "class".
|
|  islower(...)
|      S.islower() -> bool
|
|      Return True if all cased characters in S are lowercase and there is
|      at least one cased character in S, False otherwise.
|
|  isnumeric(...)
|      S.isnumeric() -> bool
|
|      Return True if there are only numeric characters in S,
|      False otherwise.
|
```

(continues on next page)

(continuação da página anterior)

```
|  isprintable(...)
|      S.isprintable() -> bool
|
|      Return True if all characters in S are considered
|      printable in repr() or S is empty, False otherwise.
|
|  isspace(...)
|      S.isspace() -> bool
|
|      Return True if all characters in S are whitespace
|      and there is at least one character in S, False otherwise.
|
|  istitle(...)
|      S.istitle() -> bool
|
|      Return True if S is a titlecased string and there is at least one
|      character in S, i.e. upper- and titlecase characters may only
|      follow uncased characters and lowercase characters only cased ones.
|      Return False otherwise.
|
|  isupper(...)
|      S.isupper() -> bool
|
|      Return True if all cased characters in S are uppercase and there is
|      at least one cased character in S, False otherwise.
|
|  join(...)
|      S.join(iterable) -> str
|
|      Return a string which is the concatenation of the strings in the
|      iterable.  The separator between elements is S.
|
|  ljust(...)
|      S.ljust(width[, fillchar]) -> str
|
|      Return S left-justified in a Unicode string of length width. Padding is
|      done using the specified fill character (default is a space).
|
|  lower(...)
|      S.lower() -> str
|
|      Return a copy of the string S converted to lowercase.
|
|  lstrip(...)
|      S.lstrip([chars]) -> str
|
|      Return a copy of the string S with leading whitespace removed.
|      If chars is given and not None, remove characters in chars instead.
|
|  partition(...)
|      S.partition(sep) -> (head, sep, tail)
|
|      Search for the separator sep in S, and return the part before it,
|      the separator itself, and the part after it.  If the separator is not
|      found, return S and two empty strings.
|
|  replace(...)
```

(continues on next page)

(continuação da página anterior)

```
|      S.replace(old, new[, count]) -> str
|
|      Return a copy of S with all occurrences of substring
|      old replaced by new.  If the optional argument count is
|      given, only the first count occurrences are replaced.
|
|  rfind(...)
|      S.rfind(sub[, start[, end]]) -> int
|
|      Return the highest index in S where substring sub is found,
|      such that sub is contained within S[start:end].  Optional
|      arguments start and end are interpreted as in slice notation.
|
|      Return -1 on failure.
|
|  rindex(...)
|      S.rindex(sub[, start[, end]]) -> int
|
|      Like S.rfind() but raise ValueError when the substring is not found.
|
|  rjust(...)
|      S.rjust(width[, fillchar]) -> str
|
|      Return S right-justified in a string of length width. Padding is
|      done using the specified fill character (default is a space).
|
|  rpartition(...)
|      S.rpartition(sep) -> (head, sep, tail)
|
|      Search for the separator sep in S, starting at the end of S, and return
|      the part before it, the separator itself, and the part after it.  If the
|      separator is not found, return two empty strings and S.
|
|  rsplit(...)
|      S.rsplit(sep=None, maxsplit=-1) -> list of strings
|
|      Return a list of the words in S, using sep as the
|      delimiter string, starting at the end of the string and
|      working to the front.  If maxsplit is given, at most maxsplit
|      splits are done. If sep is not specified, any whitespace string
|      is a separator.
|
|  rstrip(...)
|      S.rstrip([chars]) -> str
|
|      Return a copy of the string S with trailing whitespace removed.
|      If chars is given and not None, remove characters in chars instead.
|
|  split(...)
|      S.split(sep=None, maxsplit=-1) -> list of strings
|
|      Return a list of the words in S, using sep as the
|      delimiter string.  If maxsplit is given, at most maxsplit
|      splits are done. If sep is not specified or is None, any
|      whitespace string is a separator and empty strings are
|      removed from the result.
|
```

(continues on next page)

(continuação da página anterior)

```
 |  splitlines(...)
 |      S.splitlines([keepends]) -> list of strings
 |
 |      Return a list of the lines in S, breaking at line boundaries.
 |      Line breaks are not included in the resulting list unless keepends
 |      is given and true.
 |
 |  startswith(...)
 |      S.startswith(prefix[, start[, end]]) -> bool
 |
 |      Return True if S starts with the specified prefix, False otherwise.
 |      With optional start, test S beginning at that position.
 |      With optional end, stop comparing S at that position.
 |      prefix can also be a tuple of strings to try.
 |
 |  strip(...)
 |      S.strip([chars]) -> str
 |
 |      Return a copy of the string S with leading and trailing
 |      whitespace removed.
 |      If chars is given and not None, remove characters in chars instead.
 |
 |  swapcase(...)
 |      S.swapcase() -> str
 |
 |      Return a copy of S with uppercase characters converted to lowercase
 |      and vice versa.
 |
 |  title(...)
 |      S.title() -> str
 |
 |      Return a titlecased version of S, i.e. words start with title case
 |      characters, all remaining cased characters have lower case.
 |
 |  translate(...)
 |      S.translate(table) -> str
 |
 |      Return a copy of the string S in which each character has been mapped
 |      through the given translation table. The table must implement
 |      lookup/indexing via __getitem__, for instance a dictionary or list,
 |      mapping Unicode ordinals to Unicode ordinals, strings, or None. If
 |      this operation raises LookupError, the character is left untouched.
 |      Characters mapped to None are deleted.
 |
 |  upper(...)
 |      S.upper() -> str
 |
 |      Return a copy of S converted to uppercase.
 |
 |  zfill(...)
 |      S.zfill(width) -> str
 |
 |      Pad a numeric string S with zeros on the left, to fill a field
 |      of the specified width. The string S is never truncated.
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
```

(continues on next page)

(continuação da página anterior)

```
 |
 |  maketrans(x, y=None, z=None, /)
 |      Return a translation table usable for str.translate().
 |
 |      If there is only one argument, it must be a dictionary mapping Unicode
 |      ordinals (integers) or characters to Unicode ordinals, strings or None.
 |      Character keys will be then converted to ordinals.
 |      If there are two arguments, they must be strings of equal length, and
 |      in the resulting dictionary, each character in x will be mapped to the
 |      character at the same position in y. If there is a third argument, it
 |      must be a string, whose characters will be mapped to None in the result.
```

A documentação oficial[9] do Python contém toda a referência sobre a linguagem, detalhes sobre cada função e alguns exemplos (em inglês).

6.10 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!'

>>> string_a_ser_formatada.format("Não")  # também podemos passar valores diretamente␣
↪para formatação, apesar de ser desncessário
'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'
```

[9] https://docs.python.org/3/

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

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

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

6.12 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
```

6.13 Exercicios

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 7

Listas

Listas são estruturas de dados capazes de armazenar múltiplos elementos.

7.1 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
[]
```

7.2 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 com 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.

(d) Adicione bebidas ao final da listona, mas sem criar uma lista!

7.3 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**:

```
>>> lista[2:4]
[300, 400]

>>> lista[:3]
[100, 200, 300]

>>> lista[2:]
[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_estranha = ['duas palavras', 42, True, ['batman', 'robin'], -0.84, 'hipófise
↪']
>>> 42 in lista_estranha
True

>>> 'duas palavras' in lista_estranha
True

>>> 'dominó' in lista_estranha
False

>>> 'batman' in lista_estranha[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_estranha)
6

>>> len(lista_estranha[3])
2
```

7.4 Removendo itens da lista

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

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

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

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

7.5 Exercícios

1. Utilizando o `del`, remova todos os elementos da lista criada anteriormente até a lista ficar vazia.
2. 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 e sorvete e tire o sorvete da lista.

7.6 Operação 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]
```

7.7 Métodos de listas

Existem métodos que permitem alterar listas, como o `append`, que adiciona um elemento ao final da lista:

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

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

Para inserir numa posição qualquer: `list.insert(index, obj)`

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

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

```
>>> l1 = ['a', 'b', 'c']
>>> l2 = ['d', 'e']
>>> l1
['a', 'b', 'c']
```

(continues on next page)

(continuação da página anterior)

```
>>> l2
['d', 'e']

>>> l1.extend(l2)
>>> l1
['a', 'b', 'c', 'd', 'e']
```

`l2` não é modificado:

```
>>> l2
['d', 'e']
```

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

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

```
>>> l1 = ['a', 'b', 'c']
>>> l2 = l1.copy()
>>> l1
['a', 'b', 'c']
>>> l2
['a', 'b', 'c']
>>> l2.append('d')
>>> l1
['a', 'b', 'c']
>>> l2
['a', 'b', 'c', 'd']
```

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

```
>>> l1 = ['a', 'b', 'c']
>>> l2 = l1
>>> l1
['a', 'b', 'c']
>>> l2
['a', 'b', 'c']
>>> l2.append('d')
>>> l1
['a', 'b', 'c', 'd']
>>> l2
['a', 'b', 'c', 'd']
```

Tudo o que for feito com `l2` nesse exemplo também altera `l1` e vice-versa.

7.8 Exercícios

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

Nota: É possível transformatar 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'>
```

CAPÍTULO 8

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.

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'>
```

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

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

>>> valor_lido - 1
0.5
```

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.
```

CAPÍTULO 10

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 e Falso.

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

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

Qualquer expressão lógica retornará um valor em `bool`:

```
>>> 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, ele 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 11

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 identação (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("cai no else")
...
estou no if

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

11.1 Exercícios

1. Escreva um programa que, dados 2 números diferentes (a e b), encontre o menor deles.
2. Para doar sangue é necessário[1]:
   - 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 escrever o mês correspondente. Caso o usuário digite um número fora desse intervalo, deverá aparecer uma mensagem informando que não existe mês com este número.

[1] Para mais informações sobre doação de sangue, acesse http://www.prosangue.sp.gov.br/artigos/requisitos_basicos_para_doacao.html

CAPÍTULO 12

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 usando quado 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 e assim que ela se tornar falsa, o `while` para.

Nota: Na linguagem **python** a identação é obrigatória, assim como estruturs de controle, para as estruturas de repetição.

```
>>> # 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]
```

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

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

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.

```
"""
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).
"""
```

```
>>> 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
```

(continues on next page)

(continuação da página anterior)

```
Digite uma palavra: sair
Fim!
```

12.1 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 13

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 à função matemática! Mas funções em uma linguagem de programação não realizam necessariamente apenas cálculos.

Vimos o `type`, um tipo de função:

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

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

Criando uma função simples:

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

Aviso: Coloque os dois pontos após definir a função!

Nota: Faça a identação nas linhas abaixo da definição da função!

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

>>> def soma():
```

(continues on next page)

(continuação da página anterior)

```
...     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
```

13.1 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
```

(continues on next page)

(continuação da página anterior)

```
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
4 * 3 = 12
4 * 4 = 16
4 * 5 = 20
4 * 6 = 24
4 * 7 = 28
4 * 8 = 32
```

(continues on next page)

(continuação da página anterior)

```
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
9 * 10 = 90
10 * 1 = 10
10 * 2 = 20
10 * 3 = 30
10 * 4 = 40
10 * 5 = 50
```

(continues on next page)

(continuação da página anterior)

```
10 * 6 = 60
10 * 7 = 70
10 * 8 = 80
10 * 9 = 90
10 * 10 = 100
```

13.2 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 π 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. Crie uma função que receba 3 valores e calcula as raízes da fórmula de Bháskara.

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

   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)
5. Dada a função: $y = 5x + 2$, determine os valores de y para x entre -10 a +10, onde x é inteiro
6. 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.
7. Duas palavras são um "par inverso" se uma for o contrário da outra. Escreva uma função que dado duas palavras, retorne `True` caso sejam.
8. 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.
9. 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.
10. Escreva uma função que dado um número, calcule o fatorial desse número. Por exemplo, fatorial de 5:

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

11. 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 e 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:

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

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 ϵ que nossa aproximação tem com relação ao valor exato! Isso é feito comparando dois termos consecutivos da série: se a diferença entre eles (em valor absoluto!) for menor que ϵ, atingimos a precisão desejada.

Implemente, então, uma função que receba como argumentos:

- x: o ângulo (em radianos!!).
- $N_{\max}$: o número máximo de iterações.
- ϵ: 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 ϵ.

12. Calcule π 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 π.

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 π 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.

CAPÍTULO 14

Exercícios e Desafios!

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

14.1 Calculadora

14.1.1 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!

   obs: 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:

$$A_\circ = \pi\, r^2$$

14.1.2 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. Rondinelly 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 capacitância resultante é a soma:

$$C_p = C_1 + C_2 + C_3$$

Se ele ligar os três em série, a capacitância 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!

14.2 Variaveis

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

2. Abelindo é um professor muito malvado. Ele quer decidir como reprovar Rondineli, 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 Rondineli? 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*?

14.2.1 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 $3.7\,V$

2. D3yver50n resolveu minerar criptomoedas. Ele decidiu minerar *Ethereum* e viu que $1\ ETH$ =687.86 USD e 1 USD = R3.59. Ele comprou o seguinte computador:

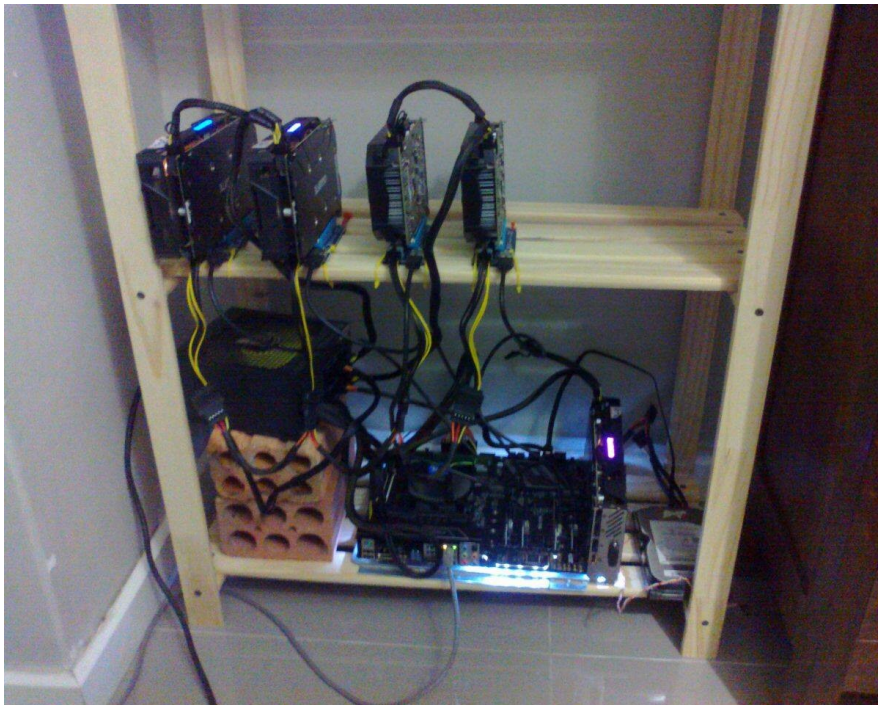
   - 5 placas de vídeo: GTX1080 TI, cada uma por R$5270,90
   - 1 placa mãe: ASRock H110 Pro, por R$920
   - 1 fonte: 1600 W, por R$2299,90
   - 1 HD: 1 TB, SATA3, 7200 RPM por R$208,90
   - 2 pentes de memória: 4 GB, DDR4, 2400 MHZ, cada um por R$259,90
   - 1 CPU: Intel Core i5-8500 por R$899,90

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

   Essas GPUs (placas de vídeo) conseguem minerar Ethereum a uma taxa de $\approx 27 Mh/s$ (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:

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



O cluster_ratio é calculado como:

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

onde n_{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 $0.008\text{centavos}/kW$.

(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?

14.3 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
```

(continues on next page)

(continuação da página anterior)

```
"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».

14.4 Listas

14.4.1 Declaração

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 com 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.

   (d) Adicione bebidas ao final da listona, mas sem criar uma lista!

14.4.2 Remoção

1. Utilizando o `del`, remova todos os elementos da lista criada anteriormente até a lista ficar vazia.
2. 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 e sorvete e tire o sorvete da lista.

14.4.3 Métodos

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

14.5 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.
```

14.6 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[1]:
   - 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 escrever o mês correspondente. Caso o usuário digite um número fora desse intervalo, deverá aparecer uma mensagem informando que não existe mês com este número.

14.6.1 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

[1] Para mais informações sobre doação de sangue, acesse http://www.prosangue.sp.gov.br/artigos/requisitos_basicos_para_doacao.html

14.7 Estruturas de repetiçao

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.

14.8 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 π 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. Crie uma função que receba 3 valores e calcula as raízes da fórmula de Bháskara.

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

   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)
5. Dada a função: $y = 5x + 2$, determine os valores de y para x entre -10 a +10, onde x é inteiro
6. 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.
7. Duas palavras são um "par inverso" se uma for o contrário da outra. Escreva uma função que dado duas palavras, retorne `True` caso sejam.

8. 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.

9. 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.

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

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

11. 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 e 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:

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

    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 ϵ que nossa aproximação tem com relação ao valor exato! Isso é feito comparando dois termos consecutivos da série: se a diferença entre eles (em valor absoluto!) for menor que ϵ, atingimos a precisão desejada.

    Implemente, então, uma função que receba como argumentos:

    - x: o ângulo (em radianos!!).
    - $N_{\max}$: o número máximo de iterações.
    - ϵ: 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 ϵ.

12. Calcule π 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 π.

    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 π 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.

CAPÍTULO 15

Contribuidores

Lista de pessoas que contribuiram com a criação deste material:

- Heitor de Bittencourt
- Juliana Karoline
- Lucas Carvalho
- Luiz Menezes
- Marcelo Miky Mine
- Tiago Martins