UNIVERSIDADE FEDERAL
DE SERGIPE

Introdução à programação com Python

Igor Terriaga Santos

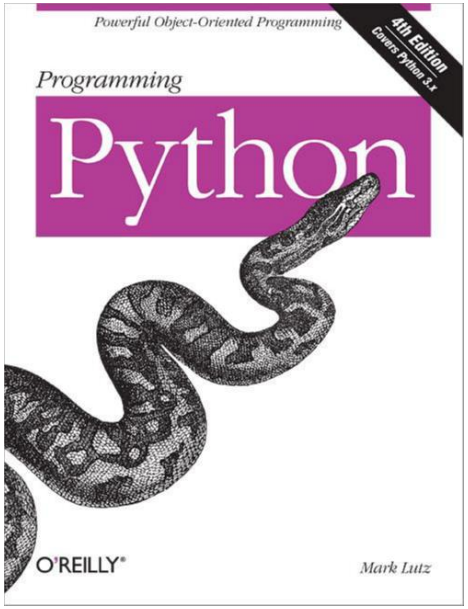
Python: Fundamentos

História

- Criada em 1991 pelo holandês Guido von Rossum no Centrum Wiskunde & Informatica(CWI) em Amsterdã, Holanda.
- Concebida a partir da linguagem ABC.
- Implementação oficial mantida pela PSF em C, conhecida como CPython.
- Outras implementações:
  - Python para .NET(IronPython)
  - JVM(Jython)
  - Python(PyPy)

História

- Nome inspirado no grupo humorístico britânico Monty Python Flying Circus.
- Ao contrário que muita gente pensa o nome não faz referência a cobra Python, o símbolo da cobra foi adotado graças a editora O’reilly.



Características

- Linguagem de programação livre, de altíssimo nível, interpretada, orientada a objetos, de tipagem forte e dinâmica.
- Linguagem simples, focada para priorizar o esforço do programador ao invés do esforço computacional, a ênfase da linguagem está em sua legibilidade.
- Uso de identação para marcar o código.
- Linguagem case sensitive
  - Ou seja: variavel, Variavel, VARIAVEL são diferentes!
- Multi-paradígma:
  - Imperativa
  - Funcional
  - Orientação a objetos

Características

- Multiplataforma
  - Pode ser usada no Linux, Mac, Windows, Web, e até para dispositivos móveis, como Android e IOS.
- Baterias inclusas, muito do que você precisa já pode estar incluso na instalação básica do python.

Versões

- **Python 1.0 - 1994**
  - Python 1.2 - 1995
  - Python 1.6. - 2000

- **Python 2.0 - 2000**
  - Python 2.1 - 2001
  - Python 2.2 - 2001
  - Python 2.3 - 2003
  - Python 2.4 - 2004
  - Python 2.5 - 2006
  - Python 2.6 - 2006
  - Python 2.7 - 2010 (Atual)

- **Python 3.0 - 2008**
  - Python 3.1 - 2001
  - Python 3.2 - 2011
  - Python 3.3 - 2012
  - Python 3.4 - 2014
  - Python 3.5 - 2015
  - Python 3.6 - 2016
  - Python 3.6.4 - 2017
  - Pytthon 3.7.2 - 2018
  - **Python 3.7.2 - 2019 (atual)**

Neste curso utilizaremos a partir da versão 3.7.x

Python 2.7 will retire in...

0	3	5	12	59	28
Years	Months	Days	Hours	Minutes	Seconds

Enable Guido Mode Huh?

What's all this, then?

Python 2.7 will not be maintained past 2020. Originally, there was no official date. Recently, that date has been updated to January 1, 2020. This clock has been updated accordingly. My original idea was to throw a Python 2 Celebration of Life party at PyCon 2020, to celebrate everything Python 2 did for us. That idea still stands. (If this sounds interesting to you, email pythonclockorg@gmail.com).

Python 2, thank you for your years of faithful service.

Python 3, your time is now.

How do I get started?

If the code you care about is still on Python 2, that's totally understandable. Most of PyPI's popular packages now work on Python 2 and 3, and more are being added every day. Additionally, a number of critical Python projects have pledged to stop supporting Python 2 soon. To ease the transition, the official porting guide has advice for running Python 2 code in Python 3.

O Zen do Python

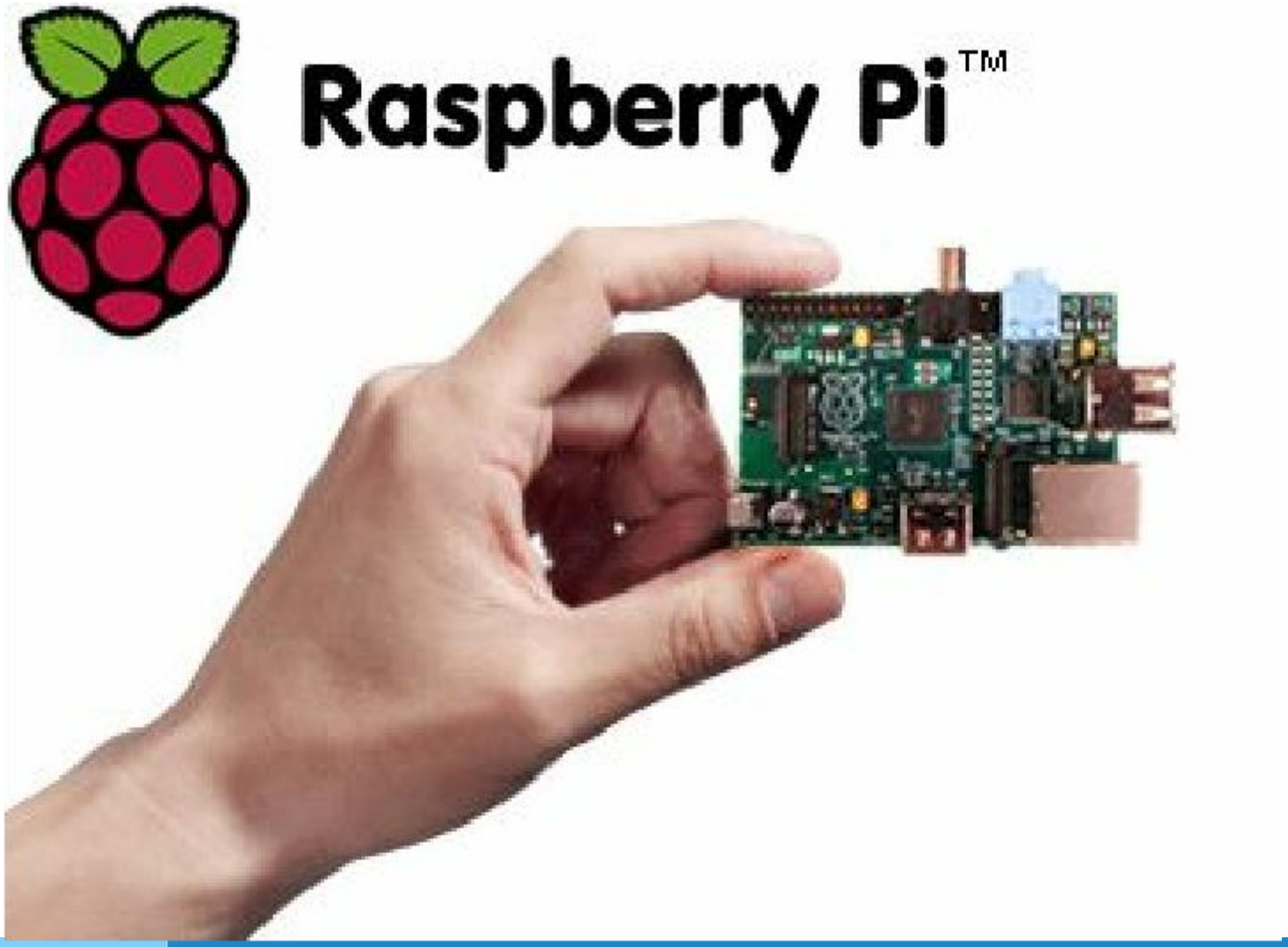
- Bonito é melhor que feio.
- Explícito é melhor que implícito.
- Simples é melhor que complexo.
- Complexo é melhor que complicado.
- Linear é melhor que aninhado.
- Esparso é melhor que denso.
- Legibilidade conta.
- Casos especiais não são especiais o bastante para quebrar as regras.
- Ainda que a preticidade vença a pureza.
- Erros nunca devem passar silenciosamente.
- A menos que sejam explicitamente silenciados.
- Diante da ambiguidade, recuse a tentação de adivinhar.
- Deveria haver um, e somente um, modo óbvio de fazer algo.
- Embora esse modo possa não ser óbvio a princípio a menos que você seja holandês.

O Zen do Python

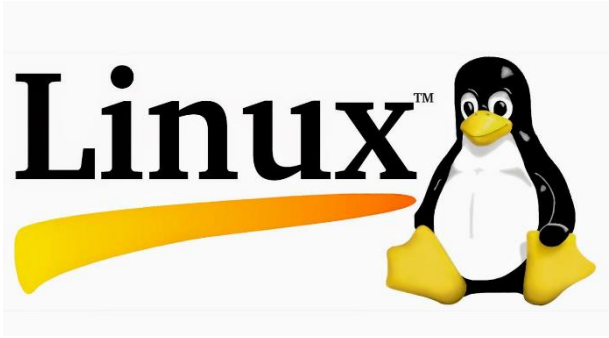
- Agora é melhor que nunca.
- Embora nunca frequentemente seja melhor que “já”.
- Se a implementação é difícil de explicar, é uma má ideia.
- Se a implementação é fácil de explicar, pode ser uma boa ideia.
- Namespaces são uma grande ideia, vamos ter mais dessas!

Usos do Python - Frameworks

Outros usos do python



Onde o Python é usado



Downloads

- Python: https://www.python.org/downloads/
  - Baixem a última versão
- Notepad++:
  https://notepad-plus-plus.org/download/v7.5.4.html
- Visual Studio Code:
  https://code.visualstudio.com/download
- Pycharm:
  https://www.jetbrains.com/pycharm/download/#section=windows

Material complementar

- Texto: PEP 8 - Guia de boas práticas para escrita de código em Python
  - http://pep8.org/
- Vídeo: Curso em Vídeo - Curso Python #02 - Para que serve Python?
  - https://www.youtube.com/watch?v=Mp0vhMDI7fA
- Livro: Python in Education
  - http://www.oreilly.com/programming/free/files/python-in-education.pdf
- Site: Python Brasil
  - http://python.org.br/

Constantes e Variáveis

Constantes

- São espaços reservados na memória do computador para armazenar uma valor fixo, que não será possível modificá-lo ao passar do tempo.
- Podem ser letras, nomes, números, entre outros tipos.
  - **Constantes numéricas:** números com valor fixo, podem ser inteiros ou reais(valores reais usam ponto decimal (.).
  - **Constantes string:** nomes ou caracteres com valor fixo, são delimitados por apóstrofo(‘) ou aspas(“).

```
>>> print(1)
1
>>> print(5.6)
5.6
>>> print('a')
a
>>> print("Olá mundo")
Olá mundo
>>>
```

Variáveis

- Posição na memória que através de um nome armazena valores, dados.
- Estes valores podem ser recuperados através dos nomes das variáveis.
- Conteúdo pode ser modificado através de um comando de atribuição (=).
- Em Python os tipos das variáveis não precisam ser declaradas, portanto, o tipo da variável pode variar durante a execução do programa.

```
>>> x = 4
>>> y = 20
>>> x = 12.5
```

```
>>> print(x)
4
>>> print(x)
12.5
>>> print(y)
20
```

Variáveis - Regras

- Devem começar com uma letra ou sublinhado _
- Não devem começar com números
- Podem conter **letras**, **números** ou **sublinhados**
- Python diferencia letras minúsculas de minúsculas, o que chamamos de *case sensitive*
- **Variáveis válidas:**
  - nome _nome nome2 sobre_nome
- **Variáveis inválidas:**
  - 1nome nome.1 #nome
- **Variáveis diferentes:**
  - nome Nome NOME

Variáveis - Boas Práticas

Como os nomes das variáveis são definidas pelo programador, procure seguir essas dicas:

- Usar nomes que expressam o significado da variável.
- Usar mneumônicos, palavras que facilitam a memoriização.
- Evitar usar caracteres especiais em nomes de variáveis (! $ % ?).
- Evitar usar acentuação
  - **Errado:** endereço = “Rua A”
  - Certo: endereco = “Rua A”
- Evitar nomear as variáveis apenas com letras,(i, j, x, y, z, a), exceto em contadores de laços de repetição e em coordenadas.
- Obedecer as regras vistas anteriormente citadas.

Variáveis - Legibilidade

exemplo.py

```
1  xyuasyausan = 20
2  shaibxaisbc = 200
3  hsaushaihs = xyuasyausan * shaibxaisbc
4  print (hsaushaihs)
```

exemplo.py

```
1  a = 20
2  b = 200
3  c = a * b
4  print (c)
```

O que cada código faz?

Qual é mais legível?

exemplo.py

```
1  valor_hora = 20
2  horas_trabalhadas = 200
3  salario = valor_hora * horas_trabalhadas
4  print (salario)
```

Variáveis - Palavras Reservadas

O Python possui palavras que não podem ser usadas como nome de variáveis, pois elas desempenham determinadas funções dentro da linguagem.

Estas são:

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

Fonte: https://www.programiz.com/python-programming/keyword-list

Comando de atribuição

Atribuindo valores

- Para atribuir um valor a uma variável é utilizado o comando de atribuição (=)
  - **variavel = valor** (Onde lê-se variável recebe valor)
- Não confundir o comando de atribuição (=) com o sinal de igualdade(==)
  - **variavel == valor** (Diferente do caso acima, aqui lê-se variável é igual a valor).
- Um comando de atribuição é formado por uma expressão do lado direito atribuída a uma variável do lado esquerdo, esta que armazena o resultado.
- Um comando de atribuição pode ser formado por:
  - Variável, constantes e operadores

x = 35 * + (2 - x)

Atribuição múltipla

Com a atribuição múltipla é possível definir vários valores para variáveis diferentes de uma só vez, como o exemplo abaixo:

```
>>> a, b, c = 1, 'a', "olá"
>>> print(a,b,c)
1 a olá
>>>
```

Entrada e Saída de Dados

Entrada de Dados

- Uso da função input( )
  - Maneira mais básica de entrada de dados em Python, ela recebe um string e armazena em uma variável:

    >>> nome = input(‘Digite seu nome: ’)

  - O comando acima irá gerar a seguinte mensagem:

    >>> Digite seu nome: _

  - O cursor _ está esperando que algo seja digitado

    >>> Digite seu nome: Antônio

  - Após digitado, o valor é armazenado na variável nome, então podemos imprimir o resultado

    >>> print(‘Olá’, nome)

  - E o resultado final será:

    >>> Olá Antônio

Entrada de Dados - Strings

- Ao usar a função input( ), o que foi digitado retorna uma String, mesmo que sejam digitados números.
  - Veja o exemplo:

    >>> Idade = input(‘Digite sua idade: ’)
  - Então digita-se a idade:

    Digite sua idade: 20
  - Podemod verificar o tipo de dado retornado através da função type( )

    >>> type(idade)
  - Logo, podemos verificar que ela é um tipo String

    <class 'str'>

Entrada de Dados - Números Inteiros

- Já que a função input( ) só retorna String, como eu poderia ler e armazenar um número inteiro?
  - É só transformar o valor em inteiro (int):

```
>>> valor = input( )
>>> valor_inteiro = int( valor)
```

- A variável valor continua sendo uma String, porém a variável valor_inteiro contém o inteiro correspondente a String da variável valor. Aqui está uma maneira resumida de se escrever o mesmo comando:

```
>>> valor_inteiro = int(input( ))
```

- Do mesmo modo podemos converter o valor retornado para um tipo float

```
>>> valor_float = float(input ( ))
```

Saída de dados

- Já temos todos os dados armazenados em variáveis, agora teremos que mostrar isto ao usuário, e é para este propósito que usaremos a função print( )
- Ela recebe um ou mais valores separados por vírgula e os imprime na tela.

```
>>> nome = input('Digite seu nome: ')
Digite seu nome: José
>>> cidade = input('Digite a cidade que você mora: ')
Digite a cidade que você mora: Aracaju
>>> print('Seu nome é ', nome, ' e mora em ', cidade, '.')
Seu nome é  José  e mora em  Aracaju .
>>>
```

Saída de dados

- Outra forma de imprimir usando a função print( ) e a função format( )

```
>>> nome = input('Digite seu nome: ')
Digite seu nome: José
>>> cidade = input('Digite a cidade que você mora: ')
Digite a cidade que você mora: Aracaju
>>> print('Seu nome é {} e mora em {}'.format(nome, cidade))
Seu nome é José e mora em Aracaju
>>>
```

Saída de dados

- Mas se você quisesse imprimir uma saída deste jeito:

  Dados do Usuário

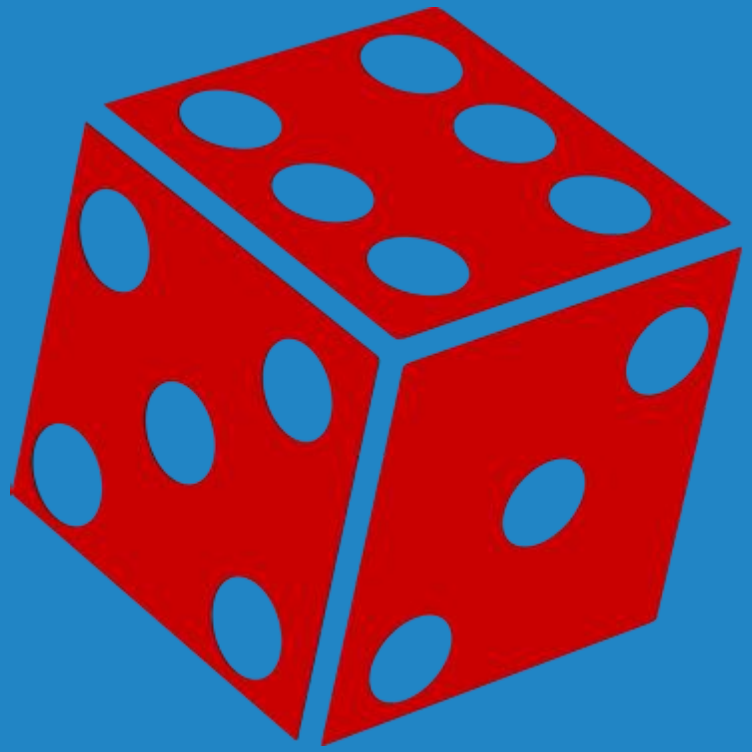
  Nome: José

  Cidade: Aracaju

Saída de dados

- Através do uso de aspas triplas “”” com a função format( ) é possível ter uma saída assim:

```
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Inte
l)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> nome = input('Digite seu nome: ')
Digite seu nome: José
>>> cidade = input('Digite a cidade que você mora: ')
Digite a cidade que você mora: Itabaiana
>>> print("""
Dados do Usuário
Nome: {}
Cidade: {}""".format(nome, cidade))

Dados do Usuário
Nome: José
Cidade: Itabaiana
>>>
```



Tipos de Dados

Tipos de Dados

- **Tipo** é uma maneira de classificar as coisas, sejam objetos ou informações.
- **Dado** é um valor em sua forma bruta.
- Logo, os tipos de dados em programação seriam como estariam classificados os valores, estes que podem ser armazenados em variáveis.

Tipos de dados - String

- É um conjunto de caracteres que formam palavras.
- Operações básicas com String:
  - Existem dois operadores que podem ser usados em Strings:
    - +
    - *
      - Ambos servem para concatenar .

```
>>> nome = 'João'
>>> sobrenome = ' Nascimento'
>>> nome_completo = nome + sobrenome
>>> print(nome_completo)
João Nascimento
>>>
```

```
>>> texto = 'hue'
>>> texto_multiplicado = texto * 6
>>> print(texto_multiplicado)
huehuehuehuehuehue
>>>
```

Tipos de dados - Números

- Existem 4 tipos numéricos em Python:
  - Inteiro (int)
  - Ponto Flutuante (float)
  - Booleano (bool)
  - Complexo (complex)*

*Neste curso não usaremos números complexos.

Tipos de dados - Números

- Operadores usados:

Operador	Operação
+	Soma
-	Subtração
*	Multiplicação
/	Divisão
//	Divisão Inteira
**	Exponenciação
%	Resto da Divisão

```
>>> print(1+1)
2
>>> print(3-1)
2
>>> print(2*4)
8
>>> print(6/2)
3
>>> print(5/2)
2
>>> print(5.0/2)
2.5
>>> print(5//2)
2
>>> print(5.0//2)
2.0
>>> print(2**4)
16
>>> print(10%3)
1
>>>
```

Tipos de dados - Booleanos

- É também um tipo de dado numérico, porém só armazena os valores 0 e 1.
- Utilizado para fazer comparações entre expressões, resultando em verdadeiro ou falso, onde:
  - 0: falso
  - 1: verdadeiro

```
>>> falso = False
>>> type(falso)
<type 'bool'>
>>> print(falso)
False
>>> verdadeiro = True
>>> type(verdadeiro)
<type 'bool'>
>>> print(verdadeiro)
True
>>>
```

Números Inteiros e Reais Misturados

- Quando é feita uma operação entre números inteiros(int) e reais (float) o valor sempre será um número real(float).

```
>>> operacao = 20.0 + 1
>>> type(operacao)
<type 'float'>
>>> print(operacao)
21.0
>>> operacao = ((20 + 4) * 3.5)/2
>>> type(operacao)
<type 'float'>
>>> print(operacao)
42.0
>>>
```

Precedência de Operadores

- Ao usar vários operadores juntos em um expressão, devemos saber qual tem prioridade, ou seja, qual será executado primeiro.
- Em Python essa é a ordem com que serão feitas as operações:

Parênteses

Exponenciação

Multiplicação/Divisão

Soma/Subtração

Esquerda → Direita

Precedência de Operadores

21 - 50 + (10 * 20) / 2**2

21 - 50 + 200 / 2**2

21 - 50 + 200 / 4

21 - 50 + 50

-29 + 50

21

```
>>> print(21 - 50 + (10 * 20) / 2**2)
21
>>>
```

Parênteses

Exponenciação

Multiplicação/Divisão

Soma/Subtração

Esquerda → Direita

Conversão de tipos - Números

- Através de funções built-in é possível fazer a conversão entre tipos.
  - float( ) - Transforma para float
  - int( ) - Transforma para inteiro

```
>>> numero_inteiro = 5
>>> type(numero_inteiro)
<type 'int'>
>>> numero_float = 5.0
>>> type(numero_float)
<type 'float'>
>>> numero_inteiro_convertido = float(numero_inteiro)
>>> type(numero_inteiro_convertido)
<type 'float'>
>>> numero_float_convertido = int(numero_float)
>>> type(numero_float_convertido)
<type 'int'>
```

Conversão de tipos - String

- Podemos também converter String em n[umeros com as funções int( ) e float( ), desde que a String só contenha números.

```
>>> valor_string = '123'
>>> type(valor_string)
<type 'str'>
>>> soma = valor_string + 1

Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    soma = valor_string + 1
TypeError: cannot concatenate 'str' and 'int' objects
>>> soma = int(valor_string) + 1
>>> print(soma)
124
>>>
```

Comandos Condicionais

Como tomar decisões?

- Programas de computador têm que tomar decisões o tempo todo, e é para isso que usamos comandos condicionais.
- São comandos que dependendo da entrada dos dados tomam decisões diferentes.
- E em Python usamos o comando if/else/elif para tomar decisões.

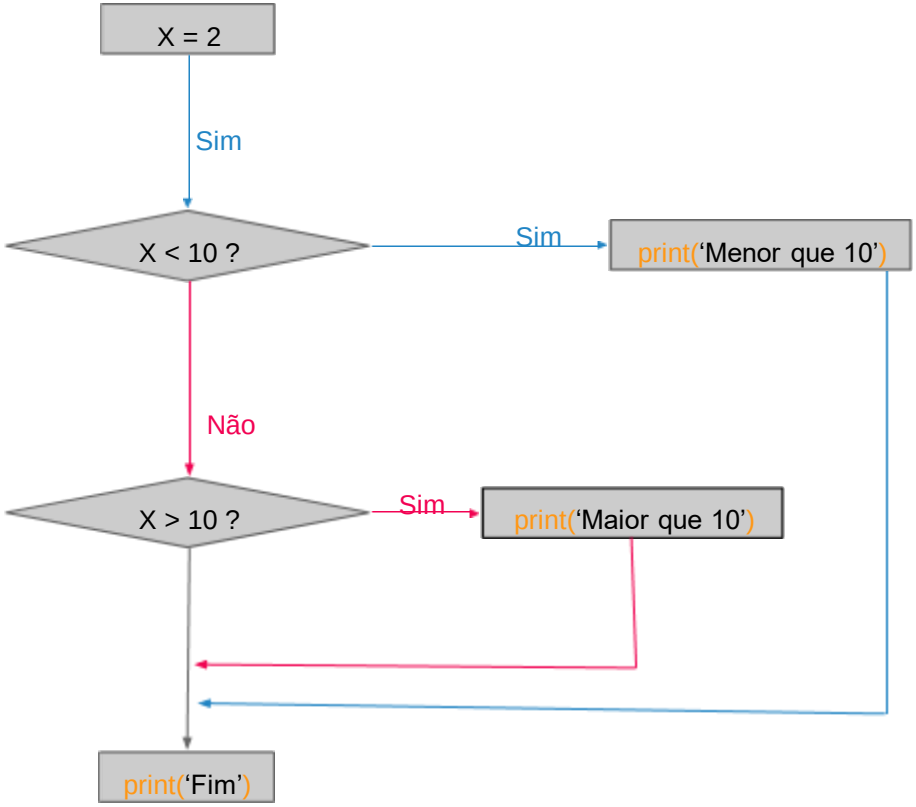
E se?

Entrada:

```
x = 2
if x < 10:
    print('Menor que 10')
if x > 10:
    print('Maior que 10')
print('Fim')
```

Saída:

```
Menor que 10
Fim
>>>
```



Blocos de códigos (Indentação)

- A linguagem Python usa indentação para saber onde um bloco de código começa e termina.
- Blocos de códigos são conjuntos de códigos dentro de um comando (if/for/while), método ou classe.

Blocos de códigos (Indentação)

- Para indentar seu código siga os seguintes passos:
  - Use ‘tab’ ou ‘espaço’ (Não use os dois, ou um ou outro!)
  - Após os comandos if/for/while aumente a identação
  - Mantenha a indentação para continuar dentro do escopo do comando
  - Para sair do escopo do comando reduza a indentação
  - Espaços com a tecla ‘enter’ e nem comentários não são considerados para a indentação
  - E mais uma vez, (não misture ‘tab’ com ‘espaço’!)

Blocos de códigos (Indentação)

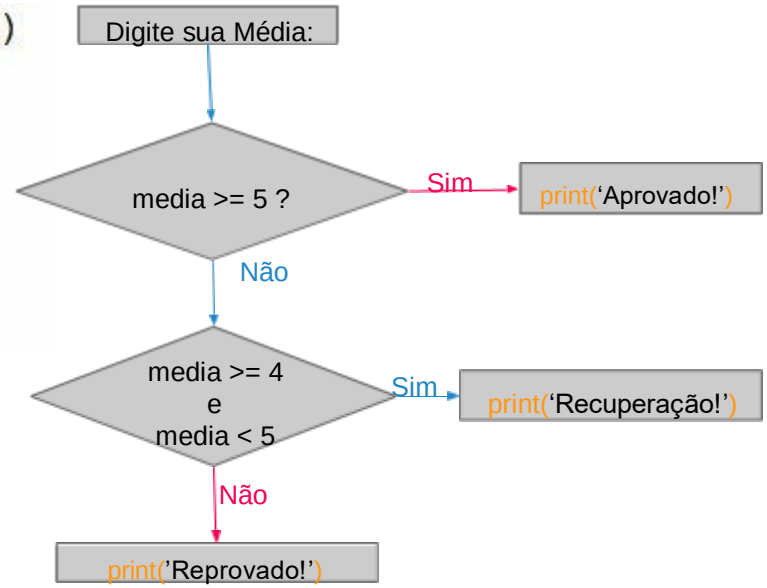
```
numero = 2
if numero % 2 == 0:
    print('O número é par')
    print('.........')
    print('O número continua sendo par')
print('Fim da parte 1 da aplicação')
print('--------------------')
```

```
for i in range(3):
    print(i)

    if i % 2 == 0 and i / 2 !=0:
        print('Número {} par'.format(i))
        print('---------------')

    print('Ainda está dentro do comando for..')
    print('-------------')

print('Fim do for')
print('Fim da aplicção')
```

Condições compostas

Entrada:

```
media = float(input('Digite sua Média:'))
if media >= 5:
    print('Aprovado!')
elif media >= 4 and media < 5:
    print('Recuperação!')
else:
    print('Reprovado!')
```



Saída:

```
Digite sua Média:4
Recuperação!
>>>
```

Comando de Repetição

while

Quando usar um laço de repetição?

```
numero = int(input('Digite o número: '))

print('Tabuada: ')
print(numero, ' x 1 = ', numero )
print(numero, ' x 2 = ', numero * 2)
print(numero, ' x 3 = ', numero * 3)
print(numero, ' x 4 = ', numero * 4)
print(numero, ' x 5 = ', numero * 5)
print(numero, ' x 6 = ', numero * 6)
print(numero, ' x 7 = ', numero * 7)
print(numero, ' x 8 = ', numero * 8)
print(numero, ' x 9 = ', numero * 9)
```

- O que acham deste código?
- Alguma coisa errada?
- Muito repetitivo?

Quando usar um laço de repetição?

```
numero = int(input('Digite o número: '))

print('Tabuada: ')
print(numero, ' x 1 = ', numero )
print(numero, ' x 2 = ', numero * 2)
print(numero, ' x 3 = ', numero * 3)
print(numero, ' x 4 = ', numero * 4)
print(numero, ' x 5 = ', numero * 5)
print(numero, ' x 6 = ', numero * 6)
print(numero, ' x 7 = ', numero * 7)
print(numero, ' x 8 = ', numero * 8)
print(numero, ' x 9 = ', numero * 9)
```

- Problemas:
  - Código repetitivo
  - Difícil de manutenção
  - Há poucas mudanças

Usando o while

- Para resolver este problema e melhorar nosso código podemos usar estruturas de repetição.
- Em Python existem 2 tipos destas estruturas: while e for.
- Veja como nosso algoritmo de tabuada ficaria usando o while:

```
numero = int(input('Digite o número: '))
n = 1
print('Tabuada: ')
while n <= 9:
    print(numero, ' x ', n, ' = ', numero * n )
    n = n + 1
```

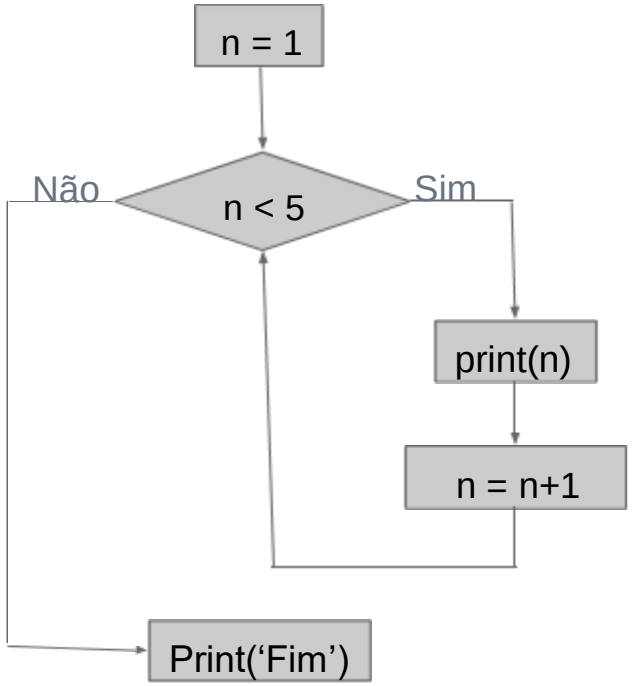
Fluxo do while

Entrada:

```
n = 1
while n < 5:
    print(n)
    n = n + 1
print('Fim')
```

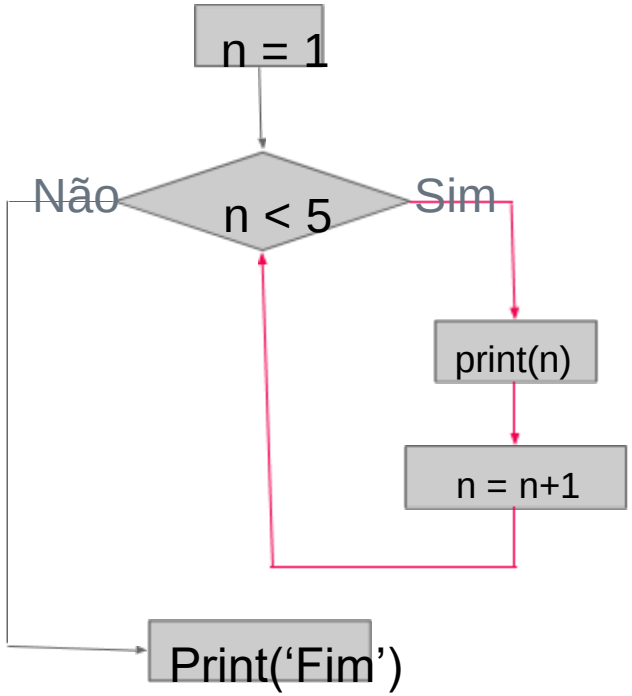
Saída:

```
1
2
3
4
Fim
>>>
```



Cuidados com laços de repetição

- Note que este algoritmo entra em um laço de repetição.
- O que aconteceria se tirassemos o incremento **n = n + 1**?
- Um laço infinito!



Entrada:

```
n = 1
while n < 5:
    print(n)
    #n = n + 1
print('Fim')
```

Saída:

```
1
1
1
1
1
1
1
1
.
.
.
```

Comando de Repetição

for

Quando usar o for?

- As vezes temos que listar uma gama específica de valores, ou seja, estes valores têm que estar em um determinado intervalo definido.
- Diferente do while, onde corremos o perigo de entrar em um laço infinito, podemos definir o número de vezes e como será feita a iteração para cada item do conjunto usando o for.
- Então podemos dizer que o for é um **laço definido**, pois ele é executado um número de vezes exata.
- Já o while seria um l**aço indefinido**, pois ele só é parado quando a condição de entrada é False.

Usando o for

- A maneira mais simples do uso do for é fazendo uma iteração em um intervalo definido dentro de uma lista.

Entrada:

```
for pessoa in ['Jõao', 'Maria', 'Carlos']:
    print('Olá {}.'.format(pessoa))
print('Até mais!')
```

Saída:

```
Olá Jõao.
Olá Maria.
Olá Carlos.
Até mais!
>>>
```

Usando o for

- Outro modo de usar o for é definindo um intervalo explícito através da função range.

Entrada:

```
for numero in range(0, 5):
    print(numero)
print('Fim')
```

Saída:

```
0
1
2
3
4
Fim
>>>
```

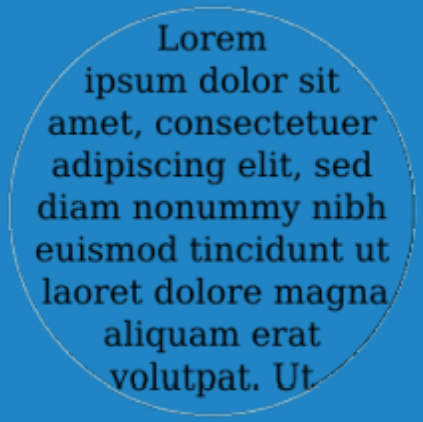
Função range

- Define uma lista com um intervalo inteiro para o laço de repetição.
- Permite controlar a quantidade de iterações em um laço.
- Oferece um contador que funciona como um iterador dentro do laço.

Função range

- Pode receber até 3 parâmetros:
  - **1 parâmetro**: gera uma lista começando do zero até o antecessor do que foi definido.
  - **2 parâmetros**: gera uma lista começando do primeiro parâmetro até o antecessor segundo parâmetro, sempre iterando de um em um.
  - **3 parâmetros**: gera uma lista igual a anterior com 2 parâmetros, porém o terceiro parâmetro define o valor do incremento da lista.

```
>>> range(5)
[0, 1, 2, 3, 4]
>>> range(1, 5)
[1, 2, 3, 4]
>>> range(1, 5, 2)
[1, 3]
>>> range(0, -6, -1)
[0, -1, -2, -3, -4, -5]
>>>
```

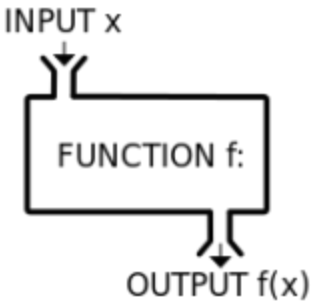

Lorem
ipsum dolor sit
amet, consectetuer
adipiscing elit, sed
diam nonummy nibh
euismod tincidunt ut
laoret dolore magna
aliquam erat
volutpat. Ut


python

Strings

Funções

Funções



- Uma função em Python é uma maneira de organizar sua aplicação, onde através de um conjunto de código reusável é possível receber um ou mais argumentos como entrada, computar algo, então retornar o resultado esperado.
- Existem dois tipos de funções em Python.
  - **Funções Built-in**, que são aquelas que já estão feitas e disponíveis para uso dentro da linguagem, ex: print( ), int( ), float ( )...
  - **Funções construídas pelo próprio programador**, que são aquelas definidas de acordo com a necessidade do mesmo.

Construindo funções

- Para definir uma função é usada a palavra chave reservada def seguida pelo nome da função e ( ).
- O corpo da função deve ser indentada após sua definição.
- O que está no corpo da função é o que será executado por ela.
- Após construir a função é preciso chamá-la para que a mesma seja executada.
- Uma vez definida é possível chamá-la quando quiser.

Definindo a função

```
>>> def ola_mundo():
        print('Ola Mundo')
```

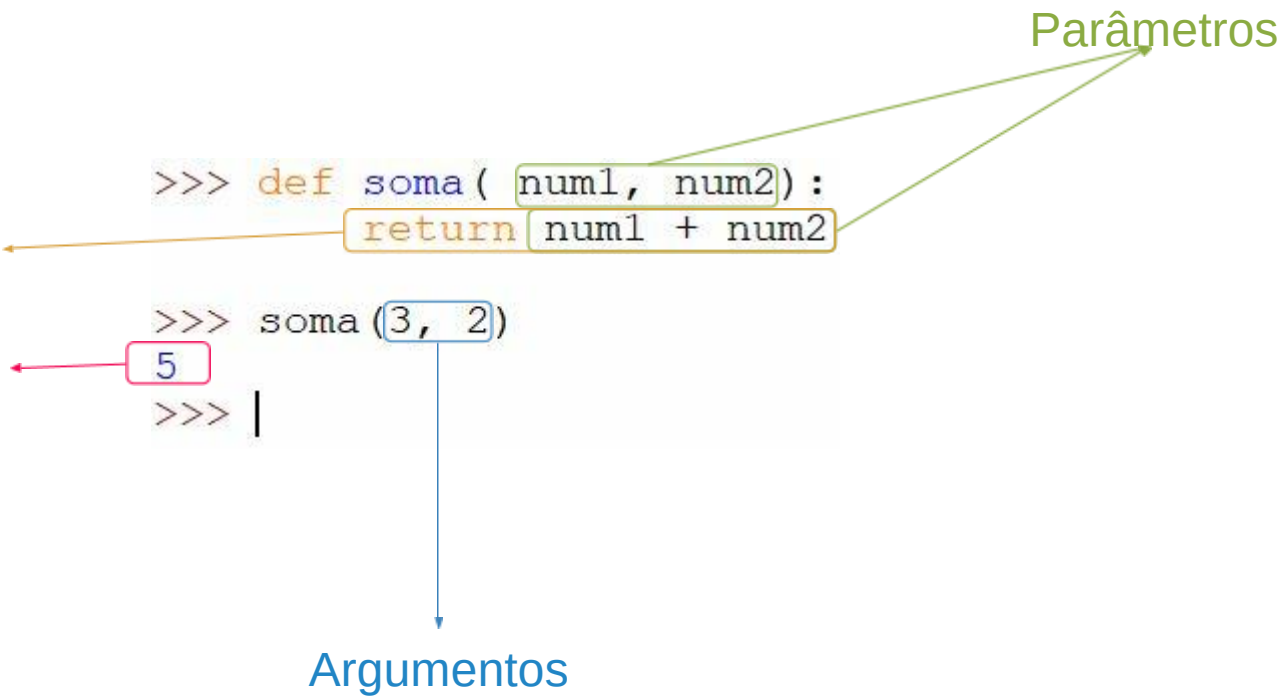
```
>>> ola_mundo()
Ola Mundo
>>>
```

Chamando a função

Esqueleto de uma função

- Uma função pode conter argumentos, parâmetros, retornar ou não algo.
- Argumento é um valor de entrada, que dependendo dele a mesma função pode executar tipos diferentes de trabalho.
- Parâmetro é uma variável que pode ser manipulada dentro da função através de um argumento passado pela mesma.
- Se a função tiver mais de um argumento ou parâmetro, eles são separados por **vírgulas**.
- Para retornar um valor pode-se usar a palavra reservada return.
- Funções que não retornam um valor são chamadas de ‘void’.

Esqueleto de uma função



Funções Built-in

- min( ): Recebe uma lista como argumento e retorna o menor valor dentro de uma lista.
- max( ) : Recebe uma lista como argumento e retorna o maior valor dentro de uma lista.
- sum( ): Recebe uma lista como argumento então soma todos os valores dentro dela.
- abs( ): Recebe um número como argumento e retorna seu módulo.

```
>>> lista = [2, 6, 8, 10]
>>> min(lista)
2
>>> max(lista)
10
>>> sum(lista)
26
>>> abs(-4)
4
>>>
```

Funções Recursivas

- Funções podem chamar outras funções dentro delas.
- Quando uma função chama ela mesma isto é chamado de recursividade.

Fatorial

n! = 1, se n = 0
n * (n - 1)!, se n > 0

```
>>> def fatorial(n):
        if n == 0:
            return 1
        else:
            return n * fatorial(n - 1)


>>> fatorial(0)
1
>>> fatorial(4)
24
>>>
```

Bibliotecas de funções em Python

- Biblioteca é um conjunto de funções que podem ser usadas pelo programador para facilitar seu trabalho.
- Em Python já existe uma série de bibliotecas pré definidas.
- Para usá-las é preciso usar o comando import, então especificar o nome, ou caminho da biblioteca.
- Se quisermos usar apenas uma função específica de uma biblioteca é usado o comando from

Biblioteca math

- Biblioteca com diversas funções matemáticas:
  - sin, cos, tan, asin, acos, atan
  - log, log2, log10
  - pow, sqrt
  - floor, ceil
  - factorial

```
>>> import math
>>> math.pow(2, 10)
1024.0
>>> math.sqrt(4)
2.0
>>> math.factorial(4)
24
>>>
```

```
>>> from math import pow
>>> pow(10, 2)
100.0
>>>
```

*Mais informações sobre a biblioteca math:

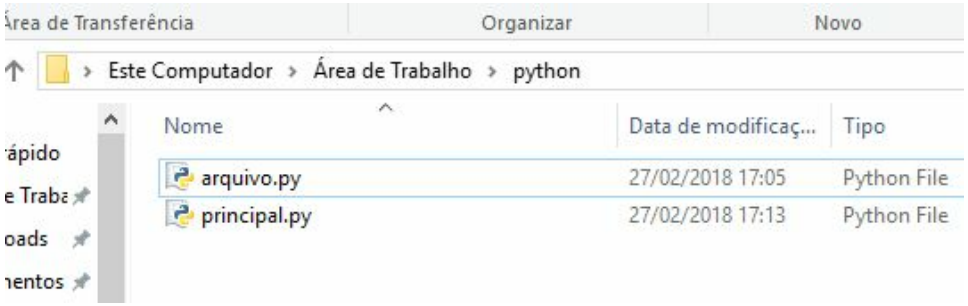
https://docs.python.org/3.4/library/math.html

Importando módulos próprios

- É possível também usar módulos próprios em Python, assim sua aplicação ficará mais enxuta, legível e com uma reusabilidade maior.

```
def minha_funcao():
    return 'Aqui está minha função!'
```

arquivo.py




```
import arquivo

print(arquivo.minha_funcao())
```

principal.py

```
Aqui está minha função!
>>>
```

Saída do principal.py

Importando bibliotecas

SorteandoOrdem.py

```python
# Sortear a ordem de apresentação de trabalhos dos alunos.
# Faça um programa que leia o nome dos quatro alunos e mostre a ordem sorteada.
import math
from random import shuffle
n1 = str(input('Primeiro aluno: '))
n2 = str(input('Segundo aluno: '))
n3 = str(input('Terceiro aluno: '))
n4 = str(input('Quarto aluno: '))
lista = [n1, n2, n3, n4]
shuffle(lista) # shuffle serve para embaralhar
print('A ordem de apresentação será: ')
print(lista)
```

Listas

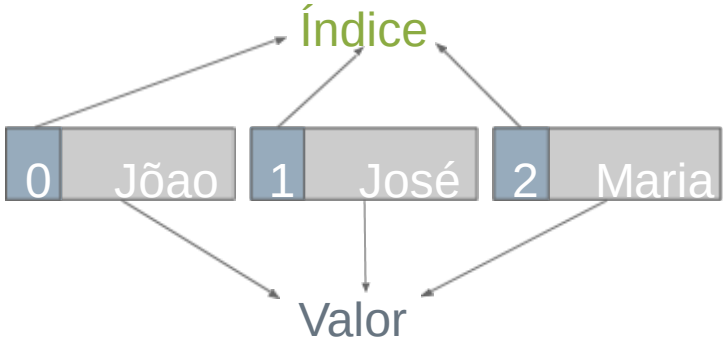
Listas

- Uma lista é uma coleção de alguma coisa.
- Em Python podemos guardar vários valores em uma única variável com o uso de listas.

```
numeros = [1, 4, 6, 7, 8]
amigos = ['João', 'Maria', 'José']
```

Listas

- A listas são delimitadas por colchetes []
- Podemos acessar um determinado elemento da lista através de seu índice, este dentro dos colchetes.
- Os índices sempre começam do zero.




```
>>> amigos = ['Jõao', 'José', 'Maria']
>>> print(amigos[1])
José
>>>
```

Percorrendo uma lista

- Podemos percorrer e listar todos os elementos de uma lista através do laço de repetição for.

Entrada:

```
amigos = ['José', 'Jõao', 'Maria']

for amigo in amigos:
    print('{} é meu amigo!'.format(amigo))

print('Até mais.')
```

Saída:

```
José é meu amigo!
Jõao é meu amigo!
Maria é meu amigo!
Até mais.
>>>
```

Operações de uma lista

- Dentre as operações que podemos utilizar nas listas estão as seguintes:
  - **Concatenação(+)** junta os elementos de duas ou mais listas.
  - **Concatenação Múltipla(*)** gera várias cópias, assim concatenando todos os elementos em uma nova lista (somente possível com a multiplicação por um número inteiro).
  - **Existência(in)** verifica se determinado elemento está contido na lista, então retorna uma valor booleano,True caso verdadeiro e False caso falso.

Operações de uma lista

```
>>> lista = ['Carro', 'Moto', 200, 100]
>>> lista2 = ['Ônibus', 'Bike']
>>> lista + lista2
['Carro', 'Moto', 200, 100, 'Ônibus', 'Bike']
>>> lista * 2
['Carro', 'Moto', 200, 100, 'Carro', 'Moto', 200, 100]
>>> 'Moto' in lista
True
>>>
```

Tamanho de uma lista

- Se quisermos saber o tamanho de uma lista é possível utilizar a função len( ).
- Ela recebe uma lista como parâmetro, então retorna o valor com o número de itens.
- Também é possível percorrer uma lista utilizando a função len( ).

```
>>> carros = ['Civic', 'Gol', 'Corsa']
>>> len(carros)
3
>>> for i in range(len(carros)):
        print(carros[i])


Civic
Gol
Corsa
>>>
```

Métodos de uma lista

- Métodos, ou funções, são um conjunto de código que desempenham uma determinada ação.
- Para auxiliar com determinados tipos de operações, as listas em Python possuem um conjunto de métodos já pré-definidos pela linguagem.
- Podemos verificar os métodos disponíveis em uma lista através da função dir( ).

```
>>> lista = [1, 2, 3]
>>> dir(lista)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__delslice
__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__
', '__getslice__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__it
er__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__'
, '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setit
em__', '__setslice__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'cou
nt', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
>>>
```

Métodos de uma lista

- count( )
  - Através do método count podemos saber a ocorrência de determinado valor dentro da lista.

```
>>> numeros = [1, 3, 4, 54, 2, 64, 2]
>>> numeros.count(3)
1
>>> numeros.count(2)
2
>>>
```

Métodos de uma lista

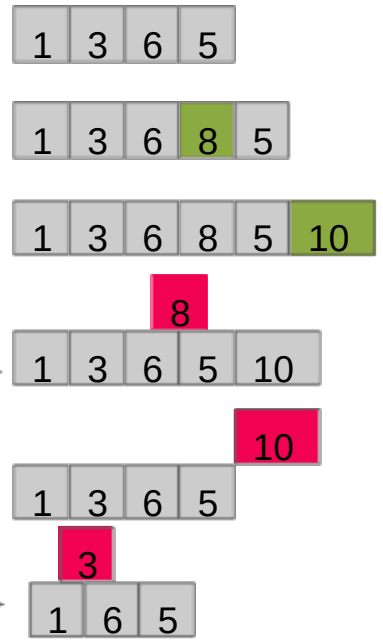
- index( )
  - Retorna a posição, índice, de um elemento em uma lista.
  - Caso se tenha elementos iguais, o método retorna a posição da primeira ocorrência do elemento na lista.

```
>>> numeros = [1, 3, 4, 54, 2, 64, 2]
>>> numeros.index(3)
1
>>> numeros.index(2)
4
>>>
```

Métodos de uma lista

- Inserindo e deletando elementos de uma lista

```
>>> numeros = [1, 3, 6, 5]
>>> print(numeros)
[1, 3, 6, 5]
>>> # insert(2, 8) - Insere na posição 2 o número 8
>>> numeros.insert(2, 8)
>>> print(numeros)
[1, 3, 8, 6, 5]
>>> # append(10) - Insere o número 10 ao final da lista
>>> numeros.append(10)
>>> print(numeros)
[1, 3, 8, 6, 5, 10]
>>> # pop(2) - Remove da lista o elemento da posição 2 e o retorna
>>> numeros.pop(2)
8
>>> print(numeros)
[1, 3, 6, 5, 10]
>>> # pop() - Remove o último elemento da lista e o retorna
>>> numeros.pop()
10
>>> print(numeros)
[1, 3, 6, 5]
>>> # remove(3) - Remove o elemento 3 da lista
>>> numeros.remove(3)
>>> print(numeros)
[1, 6, 5]
>>>
```



Métodos de uma lista

- Ordenação
  - sort( ) - Ordena de forma crescente.
  - sort(reverse = True) - Ordena de forma decrescente.
  - reverse() - Inverte a ordem dos valores.

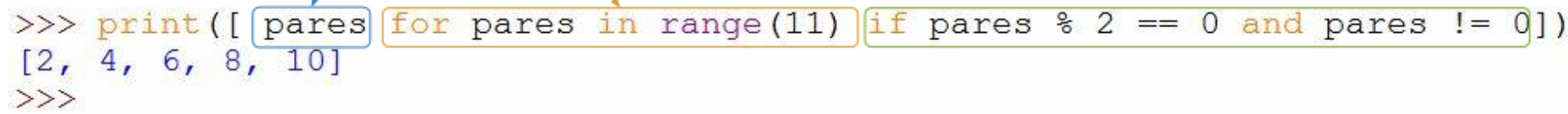
```
>>> numeros = [2, 5, 1, 6]
>>> print(numeros)
[2, 5, 1, 6]
>>> numeros.sort()
>>> print(numeros)
[1, 2, 5, 6]
>>> numeros.sort(reverse = True)
>>> print(numeros)
[6, 5, 2, 1]
>>> numeros.reverse()
>>> print(numeros)
[1, 2, 5, 6]
>>>
```

```
>>> amigos = ['Carlos', 'Maria', 'Alvaro']
>>> print(amigos)
['Carlos', 'Maria', 'Alvaro']
>>> amigos.sort()
>>> print(amigos)
['Alvaro', 'Carlos', 'Maria']
>>> amigos.sort(reverse = True)
>>> print(amigos)
['Maria', 'Carlos', 'Alvaro']
>>> amigos.reverse()
>>> print(amigos)
['Alvaro', 'Carlos', 'Maria']
>>>
```

```
>>> misto = [2, 4, 'c', 'a']
>>> print(misto)
[2, 4, 'c', 'a']
>>> misto.sort()
>>> print(misto)
[2, 4, 'a', 'c']
>>> misto.sort(reverse = True)
>>> print(misto)
['c', 'a', 4, 2]
>>> misto.reverse()
>>> print(misto)
[2, 4, 'a', 'c']
>>>
```

Compreensão de listas

- Maneira compactada de gerar uma lista que obedece a seguinte sintaxe:
  - [saída | laço de repetição | filtro]



Exercícios

1. Escreva uma programa que adicione em uma lista todos os números lidos até ser digitado 0. Ao final, exiba o conteúdo da listas, bem como os índices. Utilize a função enumerate para mostrar os indíces.

4. Construa um menu com opções de 1 - Inserir, 2 - Remover, 3 - Consultar, bem como suas funcionalidades.