

Introdução ao Python

Patrícia Novais



- » O que é Python e onde é utilizado
- » Anaconda
- » Variáveis
- » Operadores
- » Listas
- » Tuplas
- » Dicionários
- » Condicionais
- » Loops
- » Funções

- » Python é uma linguagem de alto nível, de código aberto, tipagem dinâmica e forte.
- » Foi criada por Guido Van Rossum, em 1991.
- » O nome é inspirado na série britânica Monty Python.



- » Aplicações web, desktop e mobile
- » Cálculos científicos
- » Computação gráfica
- » Automação de sistema
- » Mineração de dados
- » Big Data
- » Machine learning
- » Processamento de textos
- » Tratamento e reconhecimento de imagens
- » Animações 3D



Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one — and preferably only one — obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than right now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea — let's do more of those!

Bonito é melhor que feio.
Explícito é melhor que implícito.
Simples é melhor que complexo.
Complexo é melhor que complicado.
Linear é melhor do que aninhado.
Esparsa é melhor que densa.
Legibilidade conta.
Casos especiais não são especiais o bastante para quebrar as regras.
Ainda que praticidade vença a pureza.
Erros nunca devem passar silenciosamente.
A menos que sejam explicitamente silenciados.
Diante da ambigüidade, recuse a tentação de adivinhar.
Deveria haver um — e preferencialmente só um — modo óbvio para fazer algo.
Embora esse modo possa não ser óbvio a princípio a menos que você seja holandês.
Agora é melhor que nunca.
Embora nunca freqüentemente seja melhor que já.
Se a implementação é difícil de explicar, é uma má idéia.
Se a implementação é fácil de explicar, pode ser uma boa idéia.
Namespaces são uma grande idéia — vamos ter mais dessas!

* Dica: Pesquisar a [PEP8](#), guia de estilo em Python.

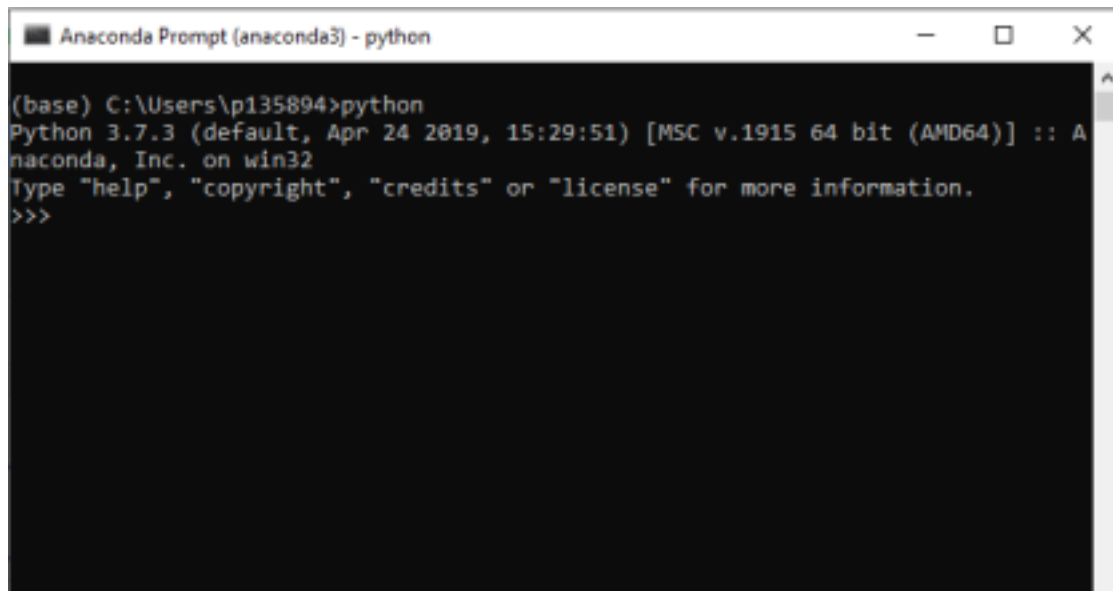


» Direto Ir até o [site oficial](#), fazer o download da versão escolhida e instalar de acordo com o sistema operacional de seu computador. A Python Brasil tem um tutorial bem explicado sobre como instalar Python no windows: [Clique aqui](#). Nessa opção, todas as bibliotecas adicionais precisam ser instaladas uma a uma.

» Outra opção é instalar através da plataforma [Anaconda](#), uma distribuição gratuita de Python (e R!) que já vem com os principais pacotes instalados. (Trabalharemos mais sobre essa plataforma na próxima aula)



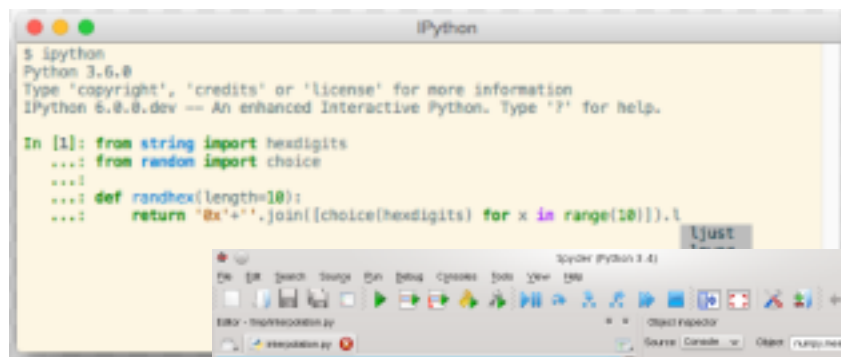
» Existem diversas maneiras de utilizar o Python. Uma delas é através do prompt de comando (terminal). Para isso, basta digitar python em um terminal.



```
Anaconda Prompt (anaconda3) - python

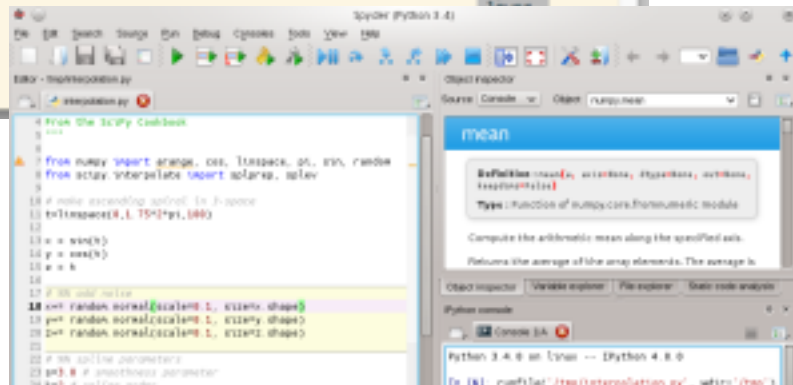
(base) C:\Users\p135894>python
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

» Há outras formas mais dinâmicas e visuais de utilizar o Python que serão abordadas nas próximas aulas.

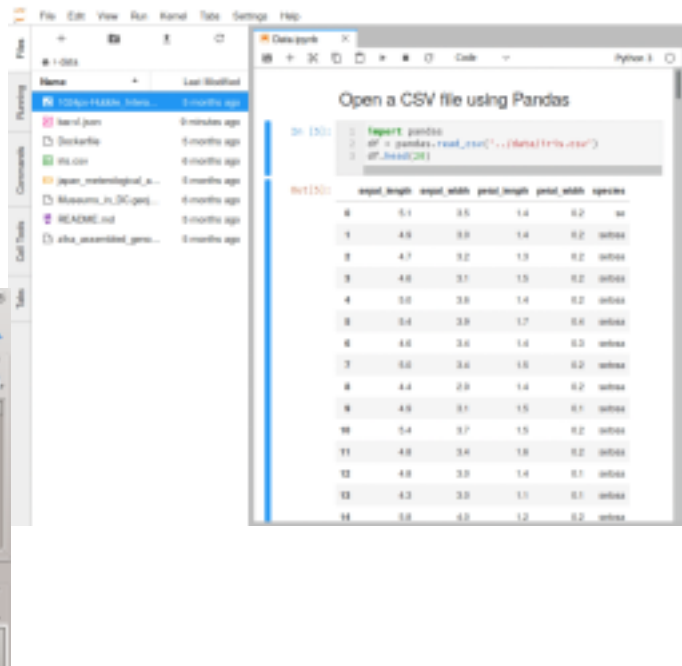


```
$ ipython
Python 3.6.0
Type 'copyright', 'credits' or 'license' for more information
IPython 6.0.0.dev -- An enhanced Interactive Python. Type '?' for help.

In [1]: from string import digits
...: from random import choice
...:
...: def randhex(length=10):
...:     return ''.join([choice(digits) for x in range(length)])
...: 
```



```
4 from the scipy cookbook
5
6
7 from numpy import arange, cos, linspace, pi, sin, random
8 from scipy.interpolate import splrep, splev
9
10 # note: expanding spline is 3-point
11 splrep(1, 1, 15*(pi, 100))
12
13 x = x_vals
14 y = sin(x)
15 n = 5
16
17 # fit cubic spline
18 co = random.normal(scale=1, size=y.shape)
19 pe = random.normal(scale=1, size=y.shape)
20 se = random.normal(scale=1, size=y.shape)
21
22 # fit spline parameters
23 p0, p1, p2 = spline parameters
24 k0, k1, k2 = spline order
```



File Edit View Run Kernel Help

Python 3.6.0

Open a CSV file using Pandas

```
In [10]: import pandas
...: df = pandas.read_csv('data/iris.csv')
...: df.head(20)
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa
6	4.6	3.6	1.6	0.3	setosa
7	5.0	3.6	1.6	0.2	setosa
8	4.4	2.9	1.4	0.2	setosa
9	4.9	3.1	1.5	0.1	setosa
10	5.4	3.7	1.5	0.2	setosa
11	4.8	3.4	1.6	0.2	setosa
12	4.8	3.0	1.4	0.1	setosa
13	4.3	3.0	1.1	0.1	setosa
14	5.0	4.0	1.2	0.2	setosa

» Python 2.*

- » Muitas aplicações foram criadas com essa versão
- » Muitas pessoas ainda utilizam essa versão
- » As atualizações foram descontinuadas desde o dia 01/01/2020.

» Python 3.*

- » Python 3 não é retrocompatível: aplicações em Python 2 podem não funcionar em Python 3
- » Veio para arrumar algumas falhas de design do Python 2
- » Já estamos na versão 3.8.*

```
(base) C:\Users\p135894>python --version  
Python 3.7.3  
(base) C:\Users\p135894>
```

Para verificar qual a versão de Python instalada.

» Variável, em programação, é o nome que damos a um valor ou expressão. Em Python, usamos o sinal de igual, '=', para atribuir um valor à uma variável.

```
a = 10  
b = 'banana'  
c = 0.5
```

» É sempre importante tentar utilizar nomes de variáveis que nos lembre qual a informação estamos armazenando nela.

```
fruta = 'banana'  
xpto = 'banana'
```

» Qual das variáveis acima são mais autoexplicativas?

Variáveis - Regras

- » Em Python, existem algumas regras para declararmos variáveis:
 - » podem ser usados algarismos, letras ou _
 - » nunca devem começar com um algarismo
 - » não podemos usar palavras-chave naturais
 - » no Python, por exemplo if, while, etc.
- » Para saber quais as palavras-chave reservadas:

```
import keyword  
print(keyword.kwlist)
```

```
(base) C:\Users\p135894>python  
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> import keyword  
>>> print(keyword.kwlist)  
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class',  
'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']  
>>>
```

» Numéricos:

» inteiros (int)

» `i = 2`

» `j = 5`

» ponto flutuante (float)

» `x = 0.5`

» `y = 2.3`

» `avogadro = 6E23`

» complexos (complex)

» `z = 2 + 1j`

» `z2 = 3 - 4j`

Observe que o decimal é dado pelo ponto, não pela vírgula.

» Literais:

» caracteres (string)

» `fruta = 'manga'`

» `nome = 'Patricia'`

» Lógicos:

» valores verdadeiros ou falsos (bool)

» `hoje_chove = True`

» `hoje_chove = False`

Python é case-sensitive!

» E como saber qual o tipo de uma variável?

Usamos a função type!

type(a)

type(x)

type(z)

```
>>> a = 5
>>> b = 3.2
>>> z = 'banana'
>>> type(a)
<class 'int'>
>>> type(b)
<class 'float'>
>>> type(z)
<class 'str'>
>>>
```

» Podemos atribuir diversos valores à diversas variáveis simultaneamente, utilizando a atribuições múltiplas!

- » `a, b, c = 2, 3, 4`
- » `a, x, z = 2, 0.5, 1+1j`
- » `hoje_chove, a, y = True, 3, 3.14`

```
>>> a, b, c = 2,3,4
>>> a
2
>>> b
3
>>> c
4
>>>
```

- » É possível converter uma variável de um tipo em outro, usando as funções `int()`, `float()`
 - » `a = 2`
 - » `b = float(a)`
 - » `b = int(b)`
- » Para converter para variável complexa, é necessário dois valores
 - » `z = complex(x,y)`

```
>>>  
>>> a = 2  
>>> type(a)  
<class 'int'  
>>> a = float(a)  
>>> a  
2.0  
>>> type(a)  
<class 'float'  
>>>
```

1. Abra um terminal. Digite python, para entrar no modo idle.
2. Atribua os seguintes valores à variáveis, um a um:
 - a. 347
 - b. 2.71
 - c. "347"
 - d. 2+3j
3. Quais os tipos das variáveis acima?
4. Faça uma atribuição múltipla das variáveis do exercício 2.
5. Declare a seguinte variável e verifique o que acontece:
 - a. teste = true
6. Transforme as variáveis do exercício 2 conforme segue:
 - a. para float
 - b. para inteiro
 - c. para float
 - d. para string
7. Crie uma variável complexa com os valores de 2.a e 2.b.

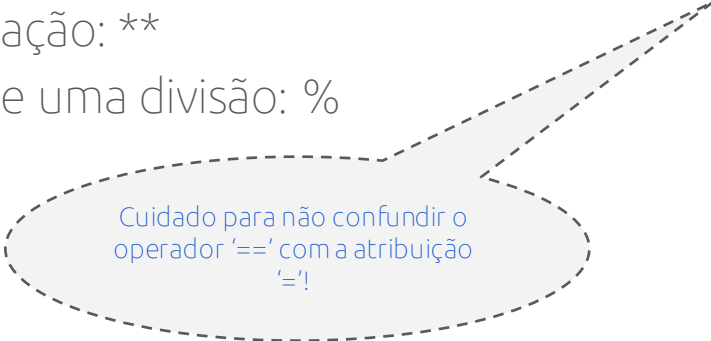
Temos dois tipos de operadores, os numéricos e os lógicos.

» Operadores numéricos

- » Adição: +
- » Subtração: -
- » Divisão: /
- » Multiplicação: *
- » Potenciação: **
- » Resto de uma divisão: %

» Operadores lógicos

- » Maior que: >
- » Menor que: <
- » Maior ou igual a: >=
- » Menor ou igual a: <=
- » Idêntico: ==
- » Diferente de: !=
- » Não: not
- » E: and
- » Ou: or



Cuidado para não confundir o operador '==' com a atribuição '='!

» Sejam $a = 5$ e $b = 3$:

- » Adição: $a + b$
- » Subtração: $a - b$
- » Divisão: $a/3$
- » Multiplicação: $a * b$
- » Potenciação: $a ** b$
- » Resto da divisão: $a \% b$

```
>>> a = 5
>>> b = 3
>>> a + b
8
>>> a - b
2
>>> a * b
15
>>> a ** b
125
```

Como em qualquer linguagem de programação, a ordem de execução das operações é

Parênteses > Exponencial > Multiplicação e/ou Divisão > Adição e/ou Subtração

Parentheses	Exponents	Multiply	Divide	Add	Subtract
()	a^2	X	\div	+	–

- » Sejam $a = 5$ e $b = 3$:
 - » Maior que: $a > b$
 - » Menor que: $a < b$
 - » Maior ou igual a: $a \geq 3$
 - » Menor ou igual a: $a \leq b$
 - » Idêntico: $a == b$
 - » Diferente de: $a != b$
 - » Não: $a \text{ not } b$
 - » E: $a \text{ and } b$
 - » Ou: $a \text{ or } b$

```
>>> a, b = 5,3
>>> a > b
True
>>> a < b
False
>>> a!=b
True
>>> a and b
```

Tabela Verdade

A	B	A and B	A or B	not(A)	not(B)
True	True	True	True	False	False
True	False	False	True	False	True
False	True	False	True	True	False
False	False	False	False	True	True

1. Abra um terminal. Digite python, para entrar no modo idle.
2. Defina as seguintes variáveis $x = 3$, $y = 4.0$, $z = 12$ e $t = \text{'banana'}$
3. Calcule:
 - a. $x + y$
 - b. $z - y$
 - c. $y ** x$
 - d. $x * z$
 - e. z / x
 - f. y / x
 - g. $y \% x$
4. Teste as seguintes relações
 - a. $x > y$
 - b. $x * y == z$
 - c. $t == y$
 - d. $x > y$ and $x < z$

» Existem diversas operações que podemos aplicar sobre as strings. Vamos definir a variável `fruta01 = 'BaNaNa'` e `fruta02 = 'AbaCaXi'` e testar algumas delas.

- » Lowercase: `fruta01.lower()`
- » Uppercase: `fruta02.upper()`
- » Capitalize: `fruta01.capitalize()`
- » Title: `fruta02.title()`
- » Concatenação: `fruta01 + fruta02`
- » Multiplicação de strings: `fruta01 * 3`
- » Começa com: `fruta01.startswith('B')`, `fruta02.startswith('B')`
- » Termina com: `fruta01.endswith('l')`, `fruta02.endswith('l')`
- » Tamanho da string: `len(fruta01)`

```
>>> fruta01 = 'Banana'
>>> fruta01.lower()
'banana'
>>> fruta01.upper()
'BANANA'
>>>
```

- » As strings em Python podem ser acessadas caracter a caracter, de acordo com seu índice.

Seja `salada = 'alface com tomate'`, temos:

- » `salada[0]`

`string[i]`

- » `salada[5]`

```
>>> salad = 'alface com tomate'
>>> salad[0]
'a'
>>>
```

- » Também é possível usar os índices para fatiar a string:

- » `salada[0:4]`

- » `salada[7:10]`

- » `salada[11:17]`

`string[i1:i2]`

```
>>> salad[7:10]
'com'
>>>
```

- » Ainda, é possível determinar qual o 'passo' utilizado no fatiamento da string:

- » `salada[0:10:2]`

- » `salada[::-3]`

`string[i]`

```
>>> salad[0:10:2]
'afc o'
>>>
```

- » Por último, é possível acessar os índices de trás pra frente!

- » `salada[-1]`

- » `salada[-10:-7]`

Hello					
0	1	2	3	4	
-5	-4	-3	-2	-1	

1. Abra um terminal. Digite python, para entrar no modo idle.
2. Crie uma variável com seu nome completo.
3. Escreva a variável em lowercase.
4. Escreva em uppercase.
5. Verifique se o nome começa com a letra P.
6. Verifique se o nome termina com a letra J.
7. Fatie a string para apenas o 1o nome.
8. Fatie a string de 2 em 2.
9. Considerando os espaços, qual o tamanho do seu nome?

- » Uma das principais estruturas do Python, a lista permite armazenar diversas variáveis de diversos tipos em apenas uma variável só.
- » Para declarar uma lista, as variáveis precisam estar entre colchetes, `[]`.
 - » `lista_compra = ['banana', 'pera', 'laranja']`
 - » `lista_coisas = ['Patricia', 47, 'abobrinha', 3.14, 1+1j]`
- » Os elementos de uma lista podem ser acessados através dos índices:
 - » `lista_compra[0]`
 - » `lista2 = lista_coisas[0:3]`
- » É possível, inclusive, criar uma lista de listas!
 - » `lista_geral = [lista_compra, lista_coisas, lista2]`

```
>>> lista_compras = ['banana', 'pera', 'laranja']  
>>> lista_compras[0]  
'banana'  
>>>
```

- » Para adicionar elementos à uma lista já existente, usamos o comando `append()`:
 - » `lista_compra.append('batata')`
- » Já para criar uma string com todos os elementos de uma lista, podemos utilizar o comando `join()`:
 - » `supermercado = ' e '.join(lista_compra)`
- » Ainda com strings, podemos criar uma lista a partir de uma string com o comando `split()`:
 - » `feira = 'batata, cebola, tomate, pepino'`
 - » `lista_feira = feira.split(',')`

```
>>> lista_compra = ['banana', 'pera', 'laranja']
>>> lista_compra
['banana', 'pera', 'laranja']
>>> lista_compra.append('batata')
>>> lista_compra
['banana', 'pera', 'laranja', 'batata']
>>>
```

1. Abra um terminal. Digite python, para entrar no modo idle.
2. Crie uma lista com nomes de 4 times de futebol.
3. Acesse o time que está na 3a posição.
4. Crie uma nova lista com duas listas de 3 times de futebol, cada uma de uma divisão diferente.
5. Crie uma lista com 3 diferentes moedas. Acrescente mais 2 outras moedas à essa mesma lista.
6. Crie uma string com a lista do exercício anterior.
7. Agora utilize a string do exercício 6 para recriar uma lista.

- » Diferente das listas, as Tuplas são objetos imutáveis, objetos que não permitem adição ou exclusão de elementos.
- » Uma tupla é declarada com parênteses ou por elementos separados apenas por vírgulas
 - » `tupla1 = (1, 3.14, 'abacate')`
 - » `tupla2 = 1, 3.14, 'abacate'`

```
>>> tupla1 = (1, 3.14, 'abacate')
>>> tupla1
(1, 3.14, 'abacate')
>>> tupla2 = 1, 3.14, 'abacate'
>>> tupla1 == tupla2
True
>>>
```

- » Dicionário é uma coleção não ordenada em que há um mapeamento de chave:valor.
- » Cada elemento de um dicionário é chamado por uma chave imutável.
- » Para se declarar um Dicionário, usamos os { } e cada elemento chave:valor é separado por vírgula.

```
>>> cadastro = {'nome': "Tania", 'idade': 35, 'fruta': 'uva', 'cor': 'roxa', 'musica': 'pop'}
>>> cadastro['nome']
'Tania'
>>> cadastro['cor']
'roxa'
>>>
```

» Podemos criar um dicionário a partir de uma lista de tuplas usando a função `dict()`.

```
>>> tupla_1 = ('nome', 'Tania')
>>> tupla_2 = ('idade', 35)
>>> tupla_3 = ('fruta', 'uva')
>>> tupla_4 = ('cor', 'roxa')
>>> tupla_5 = ('musica', 'pop')
>>> lista_tuplas = [tupla_1, tupla_2, tupla_3, tupla_4, tupla_5]
>>> cadastro = dict(lista_tuplas)
>>> cadastro
{'nome': 'Tania', 'idade': 35, 'fruta': 'uva', 'cor': 'roxa', 'musica': 'pop'}
>>>
```

» Podemos adicionar ou editar um valor em um dicionário:

Editando um valor

```
>>> cadastro['fruta'] = 'pera'
>>> cadastro
{'nome': 'Tania', 'idade': 35, 'fruta': 'pera', 'cor': 'roxa', 'musica': 'pop'}
```

Adicionando novo chave:valor

```
>>> cadastro['sobrenome'] = 'Silva'
>>> cadastro
{'nome': 'Tania', 'idade': 35, 'fruta': 'pera', 'cor': 'roxa', 'musica': 'pop', 'sobrenome': 'Silva'}
```


» Podemos deletar um elemento:

```
>>> del cadastro['musica']  
>>> cadastro  
{'nome': 'Tania', 'idade': 35, 'fruta': 'pera', 'cor': 'roxa', 'sobrenome': 'Silva'}  
>>>
```

» Podemos atualizar mais de um valor usando a função update():

```
>>> cadastro.update({'nome': 'Ana', 'idade': 27, 'fruta': 'pera'})  
>>> cadastro  
{'nome': 'Ana', 'idade': 27, 'fruta': 'pera', 'cor': 'roxa', 'sobrenome': 'Silva'}  
>>>
```

1. Abra um terminal. Digite python, para entrar no modo idle.
2. Crie um dicionário chamado cardapio em que as chaves são os dias da semana e os respectivos valores sejam os pratos do dia.
3. Crie um dicionário chamado hemograma que contenha as seguintes chave:valor:
 - hemacias: 4.71
 - hemoglobina: 14.1
 - hematocrito: 41.2
 - linfocitos: 38
 - monocitos: 7
 - resultado: saudável
4. Corrija o dicionário acima com monocitos:12.

» O Python permite entrada de dados tanto por arquivo quanto direto pelo IDLE. A função `input()` é a função utilizada para leitura de informação digitadas fora do código.

```
>>> nome = input('Escreva seu nome: ')  
Escreva seu nome:
```

Abre-se um campo para digitar a
informação

» A função `input()` sempre gera uma string. Quando necessário, devemos usar uma função para transformar em outro tipo de dado.

```
>>> idade = int(input('Qual sua idade: '))  
Qual sua idade: 33  
>>> type(idade)  
<class 'int'>  
>>> altura = float(input('Qual sua altura: '))  
Qual sua altura: 1.55  
>>> type(altura)  
<class 'float'>  
>>>
```

Transformando em valor inteiro

Transformação em valor float.

» Já para imprimir valores ao longo do código, podemos usar a função `print()` ou ainda a função `display()` em alguns idles.

```
>>> a = 3
>>> b = 5
>>> c = a * b
>>> print(c)
15
```

» Existem algumas maneiras de referenciar uma variável dentro da função `print()`, dentre elas é usar o `%`.

```
>>> nome = input('Escreva seu nome: ')
Escreva seu nome: Patricia
>>> print('Seu nome é %s.' % nome)
Seu nome é Patricia.
>>>
```

» Para referenciar mais de uma variável, usamos como argumento da função `print()` uma tupla com as variáveis.

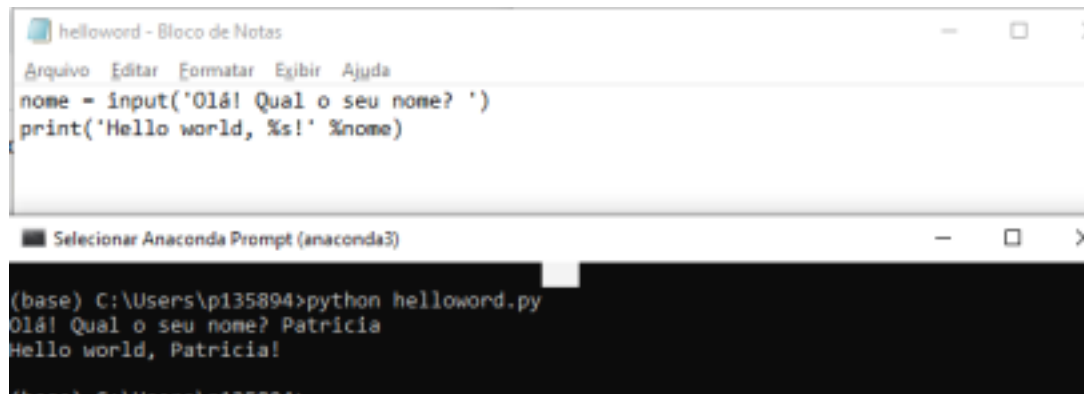
```
>>> a = 3
>>> b = 5
>>> c = a * b
>>> print('A multiplicação de %d por %d é: %f' %(a,b,c))
A multiplicação de 3 por 5 é: 15.000000
>>>
```

» Os tipos utilizados para referenciar corretamente as variáveis são:

%s	strings
%d	inteiros
%f	float
%%	%

» Para mais formatos de escrita: <https://realpython.com/python-print/>

- » Quando temos um programa com muitos comandos é inviável ter que escrevê-los direto no terminal.
- » Para essas situações, podemos escrever todos os nossos códigos em um arquivo e salvá-lo com o formato .py.
- » Para rodar o código basta, no terminal mas fora do idle Python, digitar *python nome_do_arquivo.py*



The screenshot displays two windows. The top window, titled 'helloworld - Bloco de Notas', contains a Python script with two lines: `nome = input('Olá! Qual o seu nome? ')` and `print('Hello world, %s!' %nome)`. The bottom window, titled 'Selecionar Anaconda Prompt (anaconda3)', shows the command prompt where the script is executed. The prompt shows the command `(base) C:\Users\p135894>python helloworld.py`, followed by the user input `Olá! Qual o seu nome? Patricia` and the program output `Hello world, Patricia!`.

```
helloworld - Bloco de Notas
Arquivo  Editar  Formatar  Exibir  Ajuda
nome = input('Olá! Qual o seu nome? ')
print('Hello world, %s!' %nome)

Selecionar Anaconda Prompt (anaconda3)
(base) C:\Users\p135894>python helloworld.py
Olá! Qual o seu nome? Patricia
Hello world, Patricia!
```

1. Abra um bloco de notas e crie um código chamado `imc.py` que:
 - a. Pergunte o nome da pessoa e atribua na variável `nome`.
 - b. Pergunte a altura (em metros) e atribua na variável `alt`. Não esqueça de que a variável deve ser do tipo `float`.
 - c. Pergunte o peso (em quilos) e atribua na variável `kg`. Não esqueça de que a variável deve ser do tipo `float`.
 - d. Calcule o IMC através da fórmula $IMC = peso / (alt * alt)$
 - e. Escreva o resultado do cálculo do IMC como “Olá <Fulano>, seu IMC é <xx>”, em que <Fulano> seja o nome da pessoa e <xx> seja o valor do IMC.

» Como qualquer linguagem de programação, as estruturas condicionais são partes importantes de um algoritmo.

» if:

```
>>> temperatura = 35
>>> if(temperatura > 30):
...     print('Hoje está muito quente!')
...
Hoje está muito quente!
>>>
```

Atenção à
indentação!

```
>>> temperatura = 25
>>> if(temperatura > 30):
...     print('Hoje está muito quente!')
...
>>> print('Fim')
Fim
>>>
```

A condição fica entre os parênteses e os ':' indicam o início dos comandos a serem realizados caso a condição seja satisfeita.

» if-else:

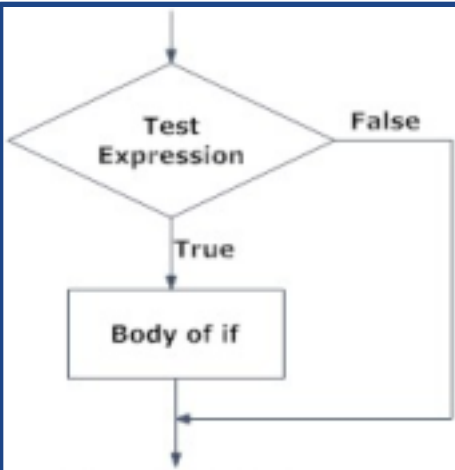
```
>>> temperatura = 28
>>> if(temperatura > 30):
...     print('Hoje está muito quente!')
... else:
...     print('A temperatura hoje está agradável!')
...
A temperatura hoje está agradável!
>>>
```

Uma vez que a 1a condição não foi satisfeita, passou-se para a 2a condição.

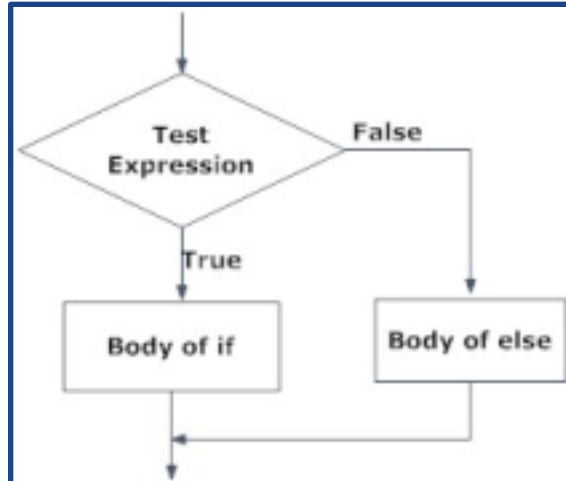
» if-elif: em Python, para evitar o aninhamento de diversos else's, existem a condicional elif, que seria uma espécie de else if.

```
>>> temperatura = 18
>>> if(temperatura > 30):
...     print('Hoje está muito quente!')
... elif(temperatura > 25):
...     print('A temperatura hoje está agradável!')
... elif(temperatura > 15):
...     print('O tempo está mudando!')
... else:
...     print('Que friaca!')
...
O tempo está mudando!
>>>
```

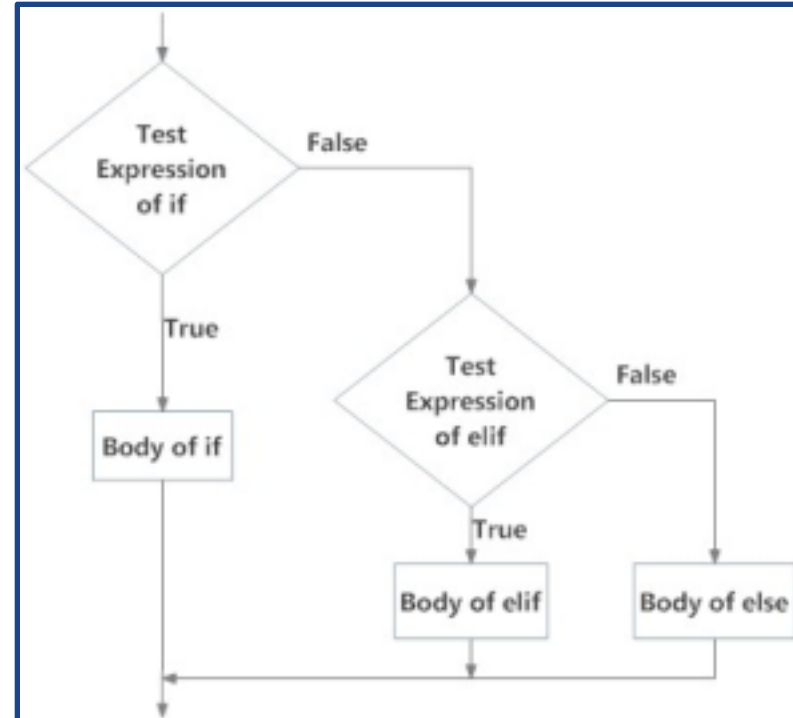
Usa-se quantos elif's forem necessários.



Comando if



Comando if-else



Comando if-elif

1. Atualize o programa `imc.py` feito anteriormente para que o resultado exibido seja

“Olá <Fulano>, seu IMC é <xx>, logo você está <situação>.”

em que a <situação> segue as condições abaixo:

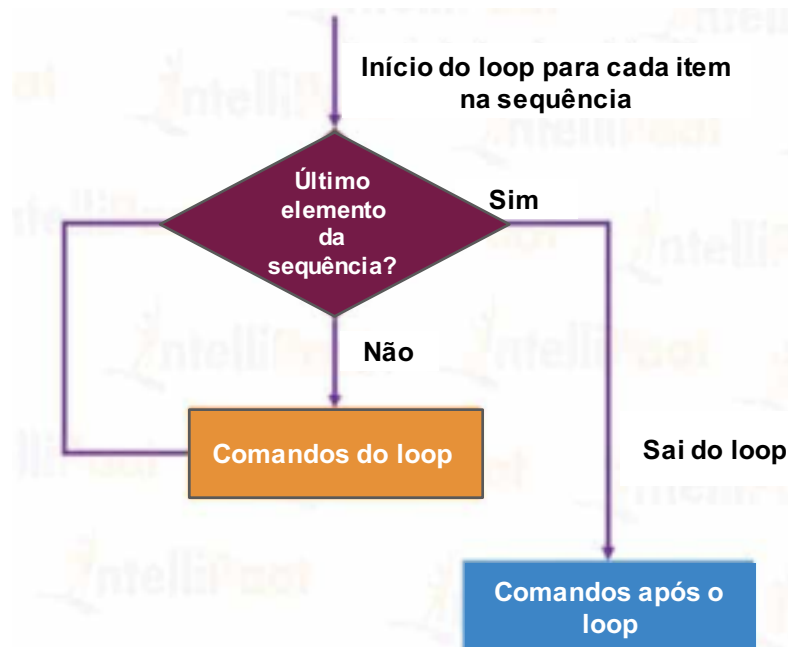
Resultado	Situação
Abaixo de 17	Muito abaixo do peso
Entre 17 e 18,49	Abaixo do peso
Entre 18,50 e 24,99	Peso normal
Entre 25 e 29,99	Acima do peso
Entre 30 e 34,99	Obesidade I
Entre 35 e 39,99	Obesidade II (severa)
Acima de 40	Obesidade III (mórbida)

» Outra estrutura importante de um algoritmo são as operações em laço (loop), onde uma determinada ação é feita repetidas vezes.

» Em Python, temos os loops for e while.

- for: opera sobre os itens de qualquer tipo de sequência (lista ou string) na ordem em que aparecem.
- while: usado quando precisamos repetir uma ação algumas vezes ou fazer uma iteração até que uma condição seja satisfeita.

» for: opera sobre os itens de qualquer tipo de sequência (lista ou string) na ordem em que aparecem.



» Exemplos:

```
>>> frutas = ['abacate', 'pera', 'abacaxi']  
>>> for fruta in frutas:  
...     print(fruta)  
...  
abacate  
pera  
abacaxi
```

```
>>> frutas = ['abacate', 'pera', 'abacaxi']  
>>> for fruta in frutas:  
...     if fruta.startswith('a'):  
...         print(fruta)  
...  
abacate  
abacaxi
```

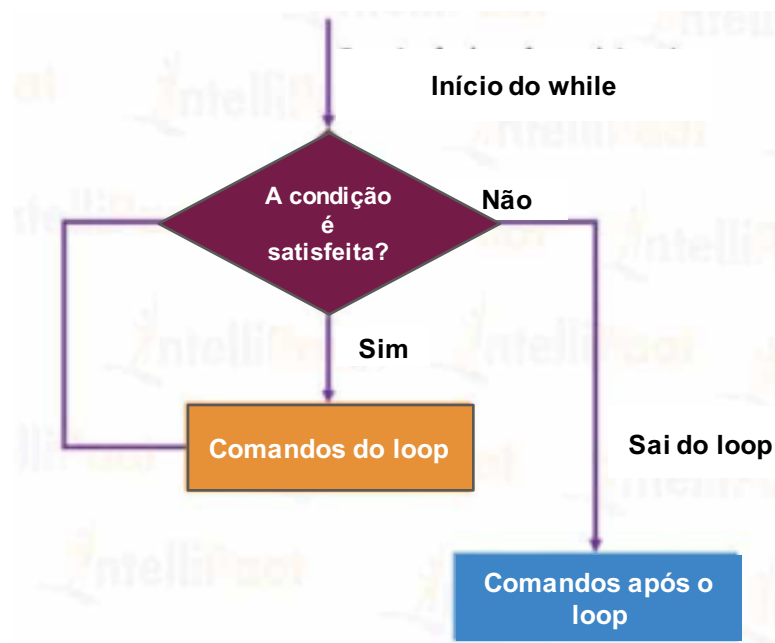
**Repetição e
Condicional!**

```
>>> lista_num = [1,2,3,4,5]  
>>> soma = 0  
>>> for numero in lista_num:  
...     soma = soma + numero  
...  
>>> print(soma)  
15
```

Acumulador.

1. Abra um bloco de notas e crie um programa chamado tabuada.py que faça:
 - a. Declare uma lista `multiplos = [1,2,3,4,5,6,7,8,9,10]`
 - b. Peça ao usuário um número inteiro de 1 a 10 e atribua na variável `number`.
 - c. Faça um laço `for` que imprima os valores da tabuada de `number`.

» while: usado quando precisamos repetir uma ação algumas vezes ou fazer uma iteração até que uma condição seja satisfeita.



Compara os dois esquemas.
Qual a diferença entre o for e o while?

- Exemplos:

```
>>> condicao = 1
>>> soma = 0
>>> while condicao <= 5:
...     soma = soma + condicao
...     condicao = condicao + 1
...
>>> print(soma)
15
>>>
```

Acumulador

Contador

```
>>> condicao = 5
>>> soma = 0
>>> while condicao >= 0:
...     soma = soma + condicao
...     condicao = condicao - 1
...
>>> print(soma)
15
>>>
```

Acumulador

Contador

1. Abra um bloco de notas e crie um programa chamado fatorial.py que:
 - a. Peça um número inteiro entre 2 e 15 ao usuário e atribua na variável valor.
 - b. Crie uma variável fat e atribua um valor inicial igual a zero.
 - c. Crie um contador cont e atribua um valor inicial igual a zero.
 - d. Usando um loop while, enquanto cont for menor que valor, atualize fat como $\text{fat} = \text{fat} \times \text{valor}$
 - e. Quando o loop terminar, imprima “O fatorial de <valor> é <fat>”, em que <valor> é o número dado pelo usuário e <fat> seja o resultado do loop.