



Prof. Luciano Nunes
2020/2
lnunes at gmail

Objetivos



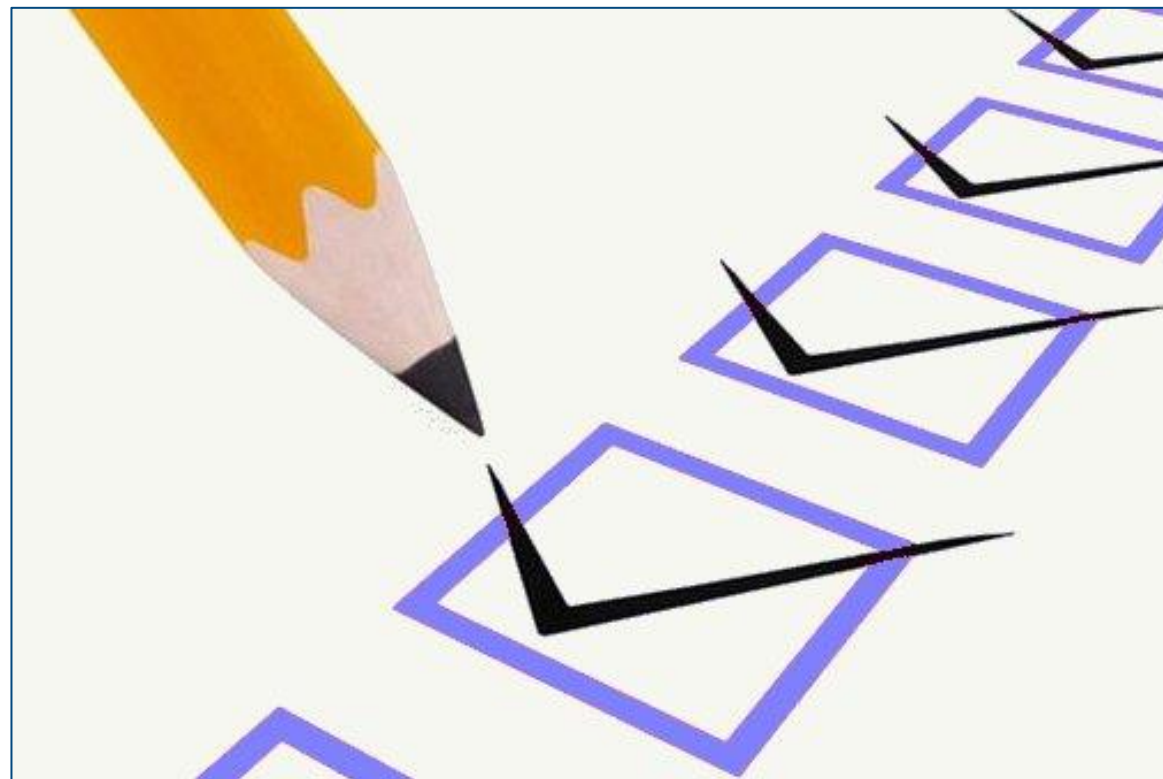
Abordagem

- Desenvolver raciocínio lógico e capacidade de escrever algoritmos utilizando a linguagem de programação Python
- Conhecer estruturas de dados disponibilizadas pela linguagem e escolher as mais convenientes para solução de problemas diversos
- Aprender a ler e escrever em arquivos e utilizar expressões e programação multithread quando necessário.

- Apresentação da teoria de forma gradual e ordenada passando pelos tipos, estruturas de dados, controle de fluxo, laços, arquivos e tratamento de exceções
- Demonstração com exemplos práticos de cada tópico apresentado
- Aplicação prática na forma de exercícios a serem resolvidos, exigindo o emprego de raciocínio lógico e conhecimentos adquiridos da linguagem

Critérios de Avaliação

- Ao final de cada aula, será disponibilizada uma lista de exercícios práticos
- As listas deverão ser entregues até o prazo determinado, através da plataforma em uso
- Listas entregues após o prazo determinado, sofrerão decréscimo do valor da nota em 50%.
- A nota final do módulo será a somatória da pontuação de cada lista de exercícios
- Siga atentamente as instruções dos exercícios propostos.



Motive-se

- Defina objetivos (metas e prazos) e periodicamente verifique os resultados. Faça ajustes quando necessário
- Aplique os conhecimentos adquiridos neste curso no seu cotidiano
- Se desafie, tente algo além do proposto
- Trabalhe de forma colaborativa, isso o ajudará a melhorar seu desempenho
- Pratique. Quando terminar, pratique mais.
- As oportunidades normalmente se apresentam disfarçadas de trabalho árduo e é por isso que muitos não as reconhecem (Bernardinho)
- E por último e não menos importante, divirta-se sempre!



DID YOU
KNOW?

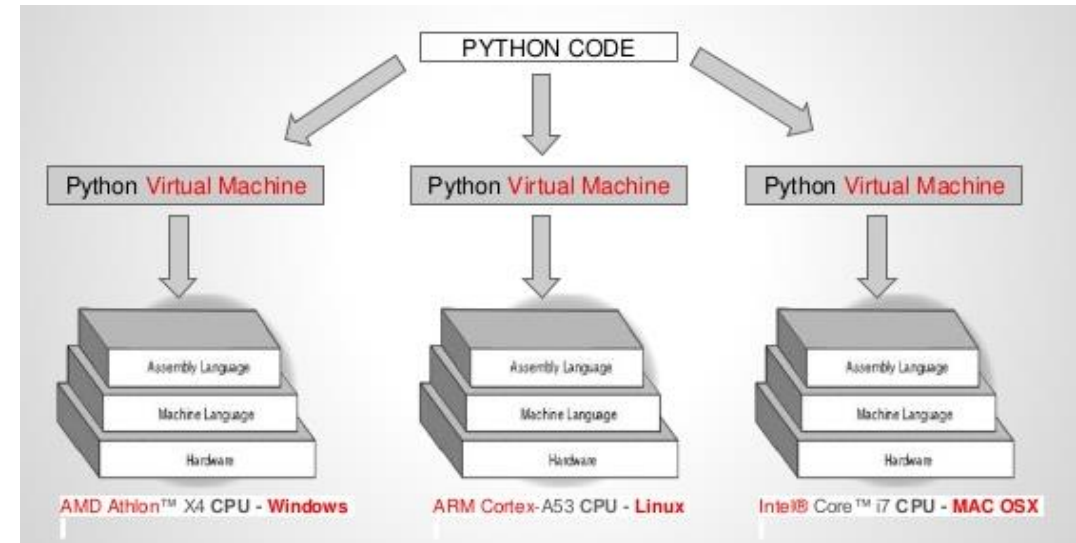


Linguagens de Programação

- Baixo Nível
 - Código Binário ou linguagem de máquina
 - Assembly
- Alto Nível
 - Compiladas
 - C / C++
 - Pascal
 - Delphi
 - Interpretadas
 - Java
 - C#
 - PHP
 - Javascript
 - Python

- Código Intermediário

- Bytecode
- JIT Compilation
- Virtual Machine



Boas Práticas de Programação

- Code Conventions
 - <https://www.python.org/dev/peps/pep-0008/>
 - Clean Code
 - Fácil de entender
 - Direto ao ponto
 - Eficiente
 - Elegante
 - Nomes Significativos (substantivos e verbos)
 - Documentação
 - Comentários
 - Testes Unitários
 - Code Review
- Dicas
 - Antes de iniciar a programação tenha certeza do entendimento do problema
 - Adquira o hábito de comentar seu código no momento em que estiver programando, preferencialmente antes de escrever a instrução
 - Seja breve e direto nos comentários, evitando escrever o que não é relevante
 - Dê preferência aos comentários em inglês, já imaginou ler um código com comentários em Latin? 😊
 - Não exagere, pois comentários em excesso podem atrapalhar ao invés de ajudar, procure comentar trechos de código

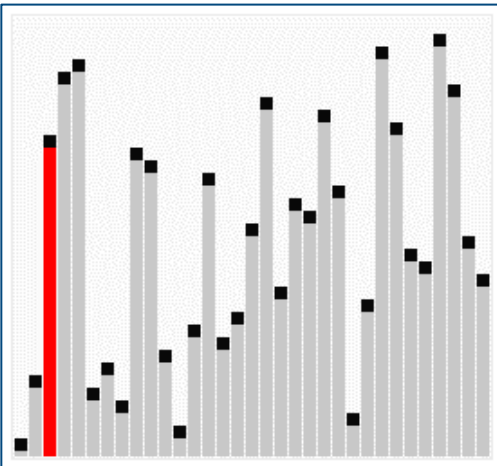
O que é um Algoritmo?

- Algoritmo é uma sequência finita de instruções bem definidas e não ambíguas, cada uma das quais devendo ser executadas mecânica ou eletronicamente em um intervalo de tempo finito e com uma quantidade de esforço finita. (Wikipedia)
- Simplificadamente um algoritmo é uma receita, um conjunto de instruções bem definidas para solucionar um problema conhecido.
- Dica: Algoritmo não se aprende copiando ou estudando algoritmos prontos e sim construindo e testando seus próprios algoritmos



Algoritmo BubbleSort

- Percorra o vetor inteiro comparando elementos adjacentes (dois a dois)
- Troque as posições dos elements se eles estiverem fora de ordem
- Repita os dois passos acima com os primeiros $n-1$ itens, depois com os primeiros $n-2$ itens, até que reste apenas um item



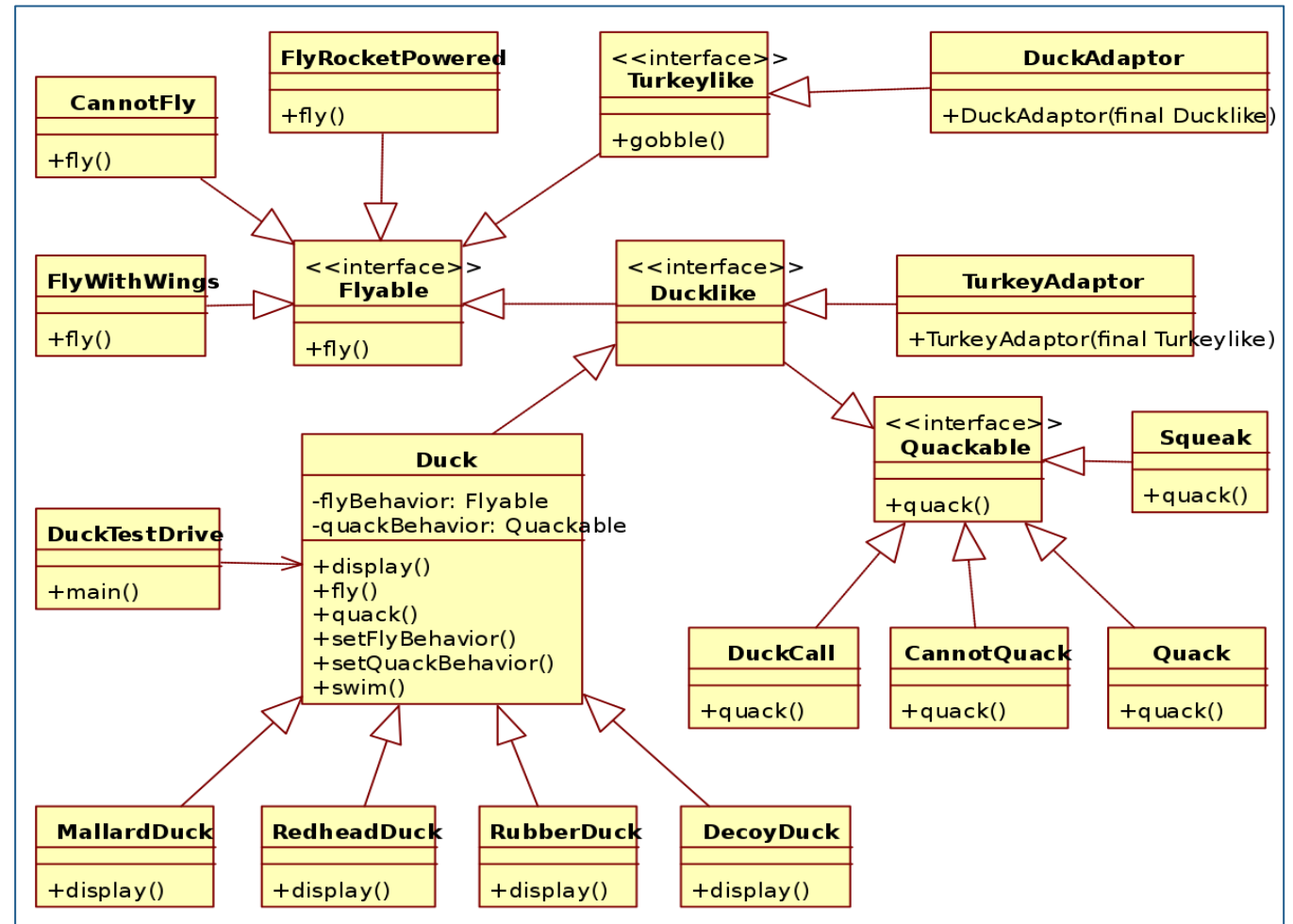
```
BubbleSort.py x
1  def bubbleSort(alist):
2      for passnum in range(len(alist) - 1, 0, -1):
3          for i in range(passnum):
4              if alist[i] > alist[i + 1]:
5                  temp = alist[i]
6                  alist[i] = alist[i + 1]
7                  alist[i + 1] = temp
8
9      alist = [111, 245, 54, 26, 93, 17, 77, 31, 44, 55, 20]
10     bubbleSort (alist)
11     print (alist)
12
```

```
BubbleSort x
C:\dev\workspace_git\DataScienceFacens\HelloWorldProje
[17, 20, 26, 31, 44, 54, 55, 77, 93, 111, 245]

Process finished with exit code 0
```

Orientação à Objetos

- Abstração
- Polimorfismo
- Herança
- Acoplamento
- Coesão
- Reutilização
- UML
- Saber construir e interpretar diagramas (Casos de Uso, Classes, Sequencia)





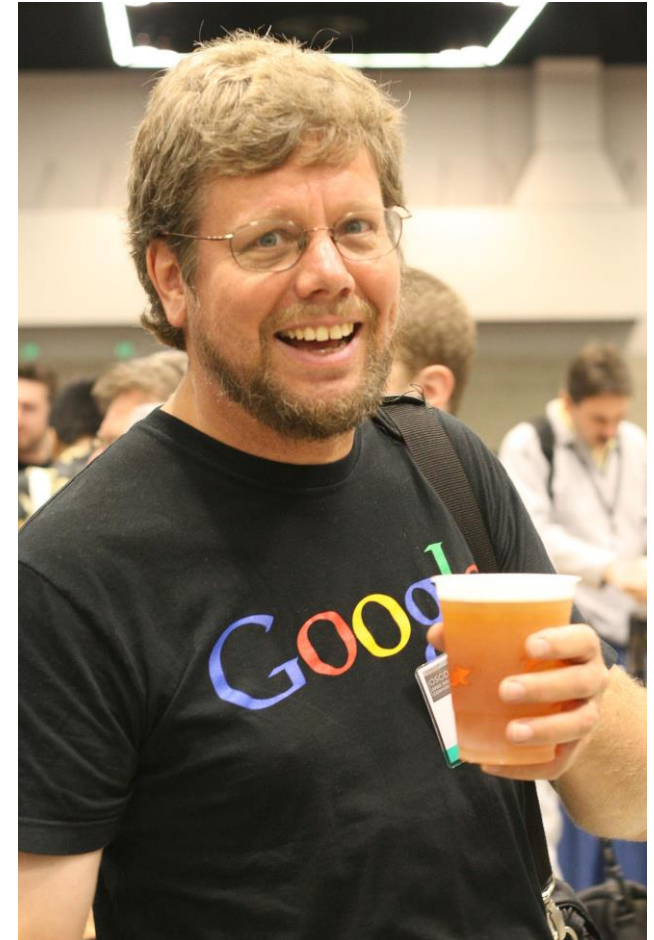
PRACTICE
MAKES
PERFECT

A stack of several colorful sticky notes (yellow, orange, and teal) is piled on a light-colored, textured surface. The topmost note is bright green and features the phrase "PRACTICE MAKES PERFECT" written in bold, black, hand-drawn capital letters.



Uma Breve Introdução

- Python é uma linguagem de programação desenvolvida por Guido van Rossum no final da década de 1980 com o objetivo de ser fácil e intuitiva, porém poderosa.
- Suas principais características são:
 - Linguagem de programação de alto nível
 - Interpretada e de código-fonte aberto
 - Interativa
 - Multi-plataforma e Multi-paradigma
 - Sintaxe simples, fácil de aprender e de manter
 - Tipagem forte e dinâmica
 - Tudo em Python é um objeto: variáveis, funções, etc. Cada objeto tem um ID, tipo e valor
- Curiosamente o nome não tem nenhuma relação com o anfíbio de mesmo nome e sim uma homenagem ao grupo de comédia britânico Monty Python!



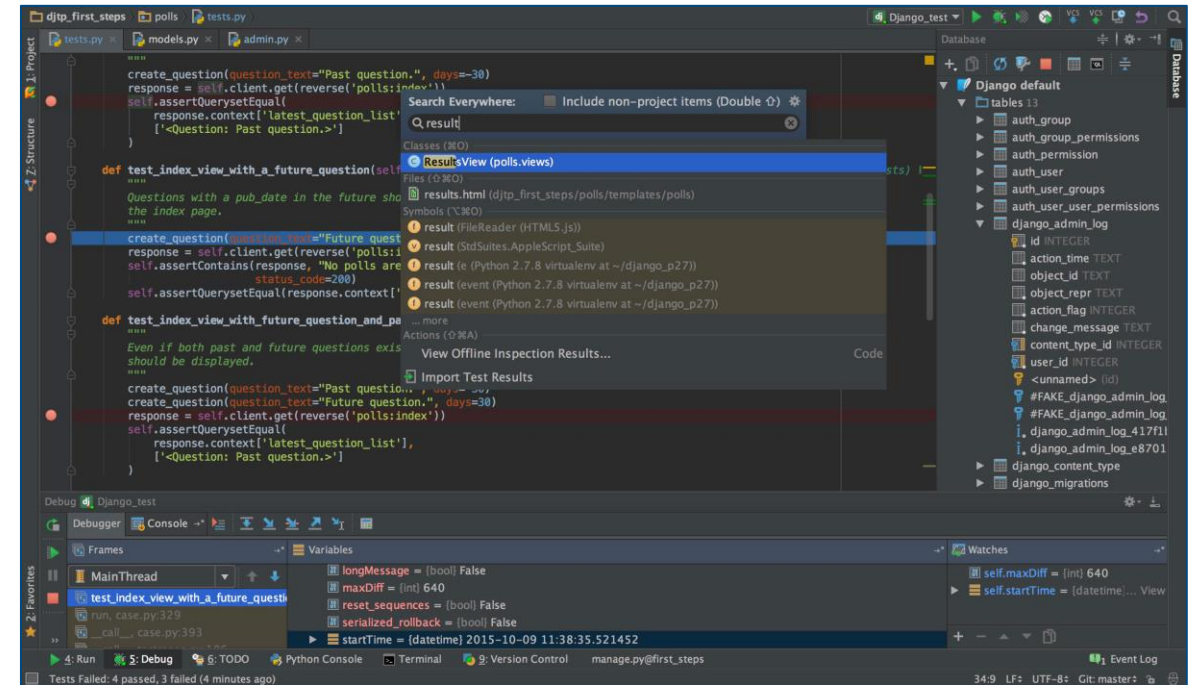
Uma Breve Introdução



- Curiosamente o nome não tem nenhuma relação com o anfíbio de mesmo nome e sim uma homenagem ao grupo de comédia britânico [Monty Python!](#)

Ambiente de Desenvolvimento

- IDE Versus Ambiente de desenvolvimento
- Aspectos a serem considerados
 - Estrutura e organização dos códigos fontes
 - Compilador e executor integrado
 - Ferramenta para depuração (debug)
 - Ferramenta para inspeção (variable inspect)
 - Gerenciador de pacotes
 - Consumo de recursos
- Ambientes de Desenvolvimento para Python
 - PyCharm
 - IDLE
 - Anaconda
- IDE
 - Visual Studio Code
 - Atom
- Ambientes Interativos
 - IPython
 - Jupyter Notebook



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

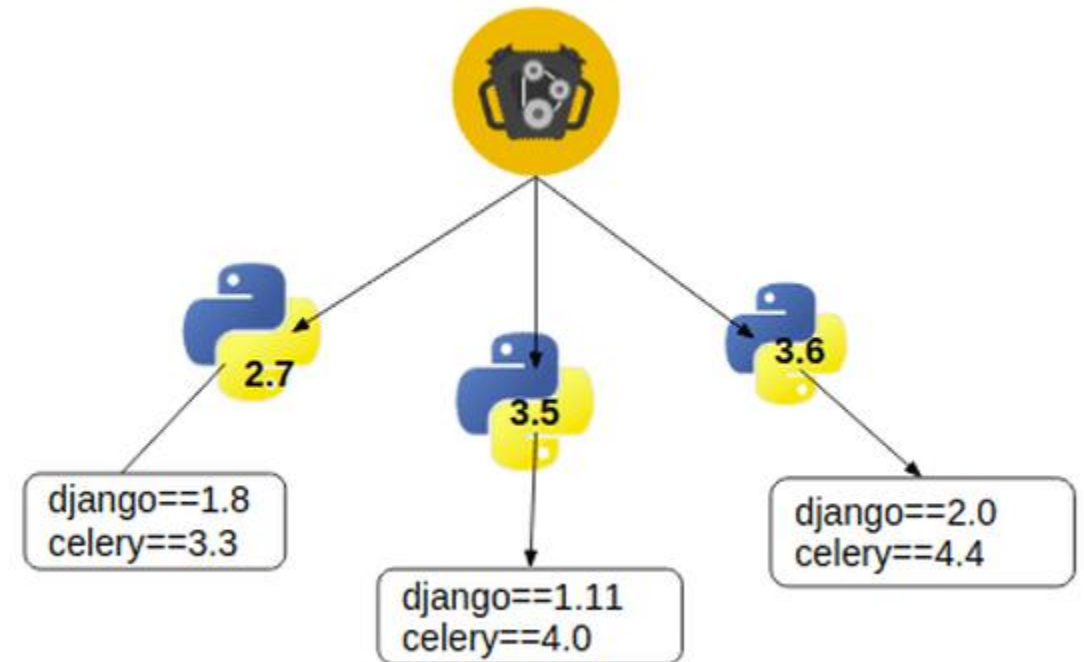
Python pip

- Sistema de instalação e gerenciamento de pacotes do Python
- Utiliza o Python Package Index ou **PyPI** como repositório
- <https://pypi.org/>
- A grande vantagem da utilização do **pip** é a facilidade de sua execução através de linha de comando:

```
pip install algum-nome-de-pacote
```

```
pip uninstall algum-nome-de-pacote
```





Virtualenv

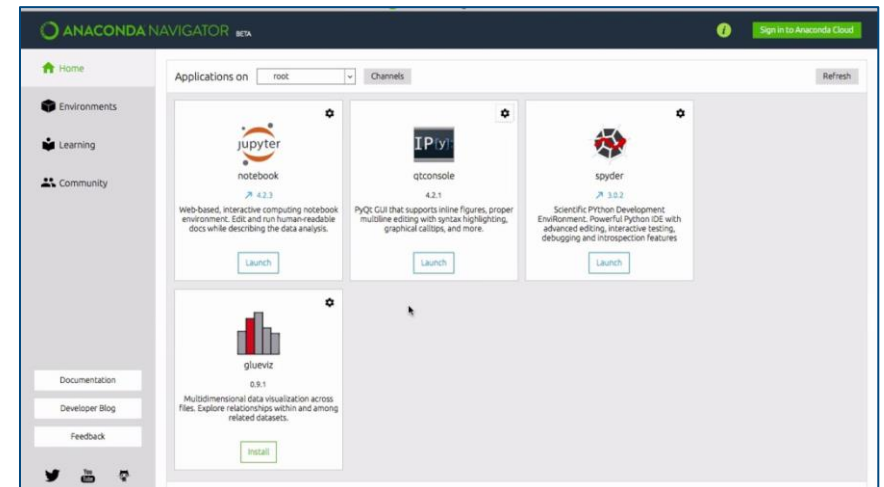
- Um virtualenv é uma ferramenta que facilita o agrupamento de projetos, dependências e bibliotecas em um único lugar. Esse ambiente é específico para os projetos contidos nele e não interfere as dependências de outros projetos
- Com eles é possível a existência de projeto X que usa a versão 1.0 da biblioteca Z e também manter o projeto Y usando a versão 2.0 da mesma biblioteca Z
- Quando um ambiente virtual é criado, os executáveis do Python, *pip*, etc são copiados para esse ambiente e todas as bibliotecas instaladas através do *pip* serão mantidas apenas nesse ambiente, não afetando outros ambientes
- Dessa forma é possível administrar ambientes completamente diferentes, sem que um interfira no outro, por exemplo um ambiente usa Python na versão 2.7 e a biblioteca NumPy na versão 1.13 e outro ambiente com Python na versão 3.6 e a biblioteca NumPy na versão 1.15, cada um desses ambientes com as mesmas bibliotecas em versões diferentes
- Esse recurso é bastante útil, quando se pretende migrar uma ou mais bibliotecas para versões mais recentes, sem quebrar a compatibilidade dos seus projetos com as versões anteriores dessas bibliotecas



ANACONDA®

Anaconda

- O Anaconda é uma plataforma open source para Data Science, muito popular entre os cientistas de dados, estatísticos, cientistas da computação, entre outros
- Multiplataforma
- Possui um gerenciador de ambientes virtuais (virtualenv)
- Instala e se integra com o **pip**
- Disponibiliza um repositório com mais de 1000 bibliotecas open source para os mais diversos fins (<https://docs.anaconda.com/anaconda/packages/pkg-docs>)
- Entre os pacotes disponibilizados na instalação está o Jupyter
- <https://www.anaconda.com/download/>





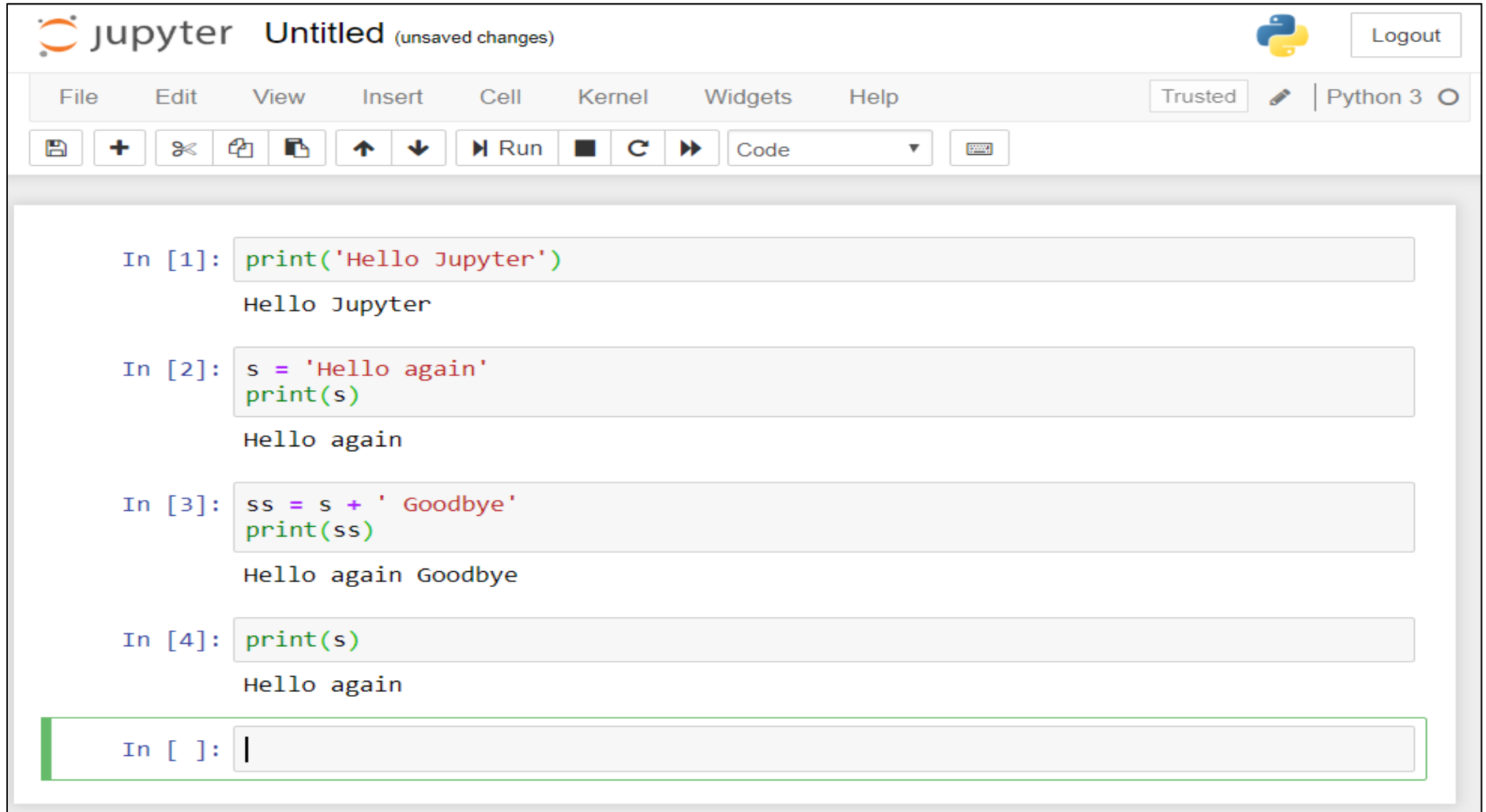
jupyter


Jupyter

- Aplicação open source muito utilizada para ensinar linguagens de programação (kernels)
- Pode ser executada através de um navegador
- Permite programar de forma iterativa, criar e compartilhar documentos que contenham trechos de código, equações, adicionar notas e visualizar suas ações de forma muito intuitiva e rápida
- Possui integração nativa com ferramentas específicas de big data como Apache Spark por exemplo
- Inclui ferramentas para limpeza de dados (data cleaning), transformação (transforming), simulação numérica, modelagem estatística, visualização de dados, aprendizado de máquinas, entre outras
- A forma mais simples de se instalar o Jupyter é através do Anaconda, porém pode ser instalada através do ***pip*** (pip install jupyter)
- www.jupyter.org

Jupyter

.ipynb



jupyter Untitled (unsaved changes)  Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Save Add Delete Copy Paste Undo Redo Run Stop Restart Code

```
In [1]: print('Hello Jupyter')
Hello Jupyter
```

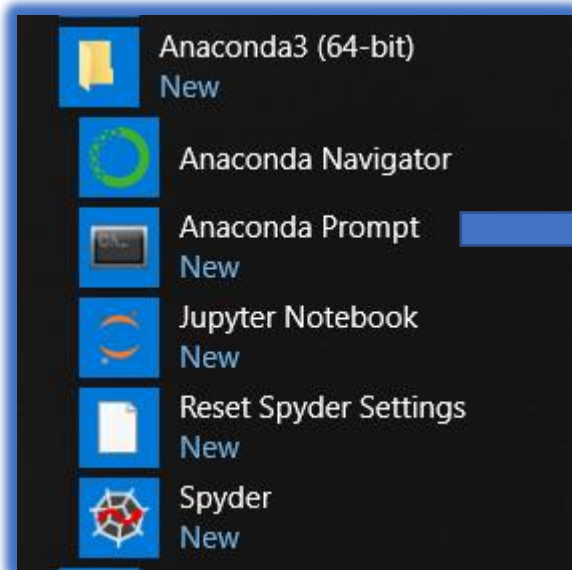
```
In [2]: s = 'Hello again'
print(s)
Hello again
```

```
In [3]: ss = s + ' Goodbye'
print(ss)
Hello again Goodbye
```

```
In [4]: print(s)
Hello again
```

```
In [ ]: |
```

Jupyter



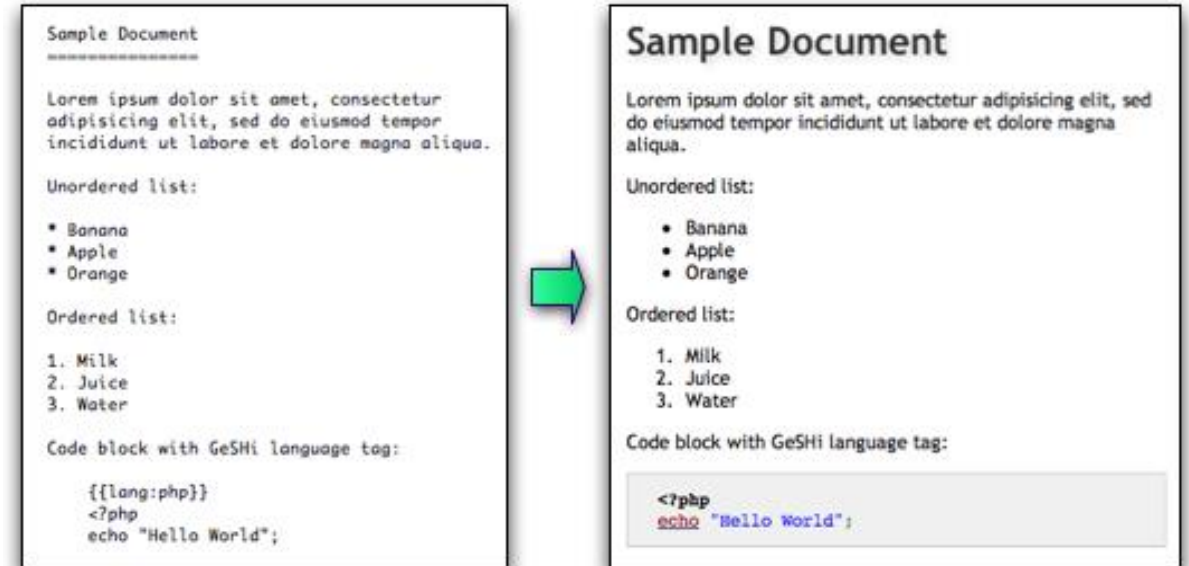
```
Anaconda Prompt - jupyter notebook

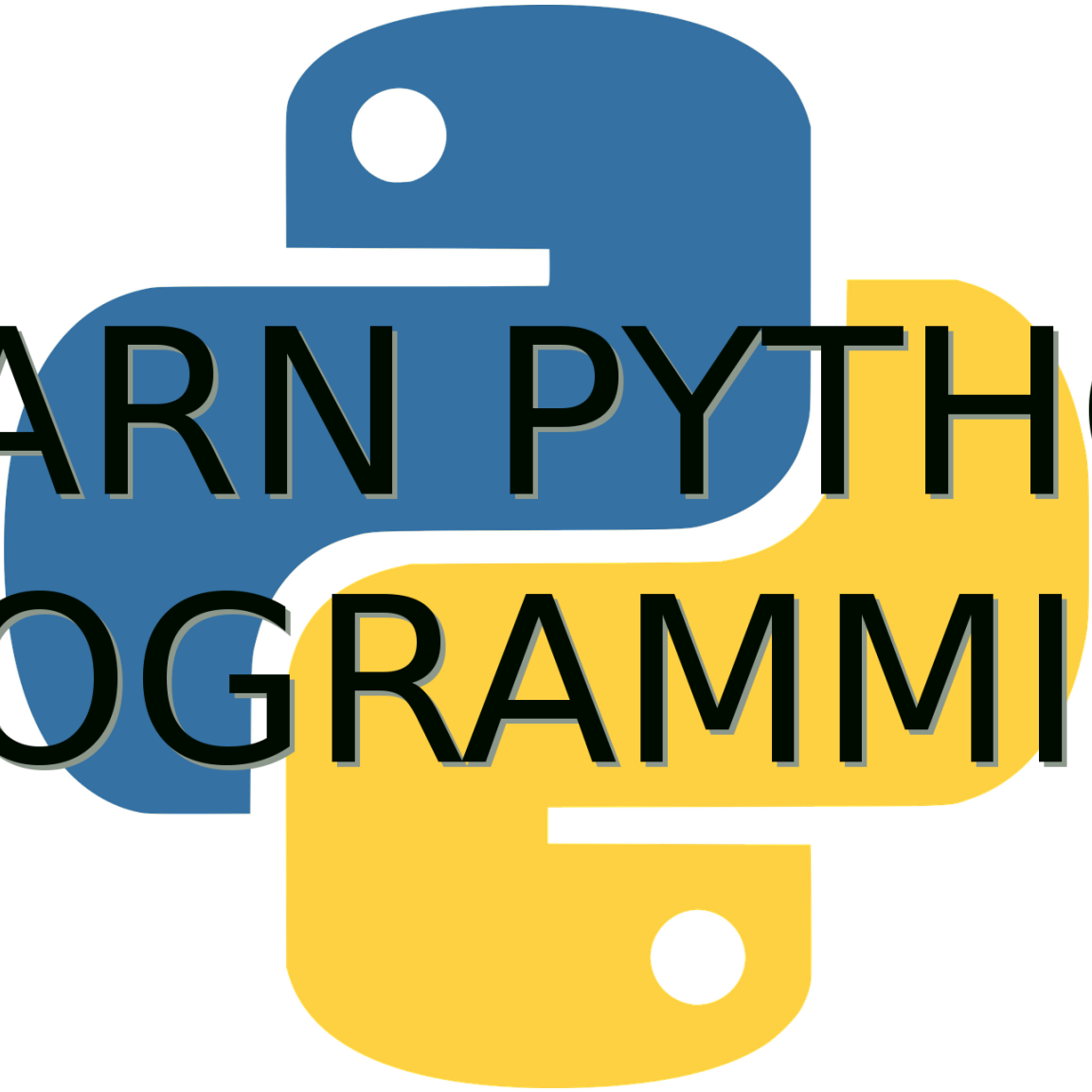
(base) C:\Users\sao\lsan>jupyter notebook
[I 22:52:25.485 NotebookApp] The port 8888 is already in use, trying another port.
[I 22:52:25.532 NotebookApp] JupyterLab beta preview extension loaded from C:\dev\python\anaconda3\lib\site-packages\jupyterlab
[I 22:52:25.533 NotebookApp] JupyterLab application directory is C:\dev\python\anaconda3\share\jupyter\lab
[I 22:52:25.686 NotebookApp] Serving notebooks from local directory: C:\Users\sao\lsan
[I 22:52:25.686 NotebookApp] 0 active kernels
[I 22:52:25.686 NotebookApp] The Jupyter Notebook is running at:
[I 22:52:25.686 NotebookApp] http://localhost:8889/?token=844164a39598434eb6a281e6e0bc15bbeb483aeaea5f92c7
[I 22:52:25.686 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 22:52:25.689 NotebookApp]

Copy/paste this URL into your browser when you connect for the first time,
to login with a token:
http://localhost:8889/?token=844164a39598434eb6a281e6e0bc15bbeb483aeaea5f92c7&token=844164a39598434eb6a281e6e0bc15bbeb483aeaea5f92c7
[I 22:52:25.845 NotebookApp] Accepting one-time-token-authenticated connection from ::1
```

Jupyter - Markdown

- MD é um sistema simples de formatação de textos que busca tornar simplificar a escrita e leitura de textos técnicos
- Com um conjunto razoavelmente pequeno de códigos, fornece um conjunto de elementos de formatação (símbolos) que são automaticamente convertidos para HTML
- Com ele é possível formatar elementos em itálico, negrito, criar citações, listas ordenadas e não ordenadas, tabelas, links e trechos de código nativo de várias linguagens
- O Jupyter fornece suporte a escrita de trechos de código no formato MD o que facilita muito o aprendizado e é uma forma muito eficiente de documentar seus programas

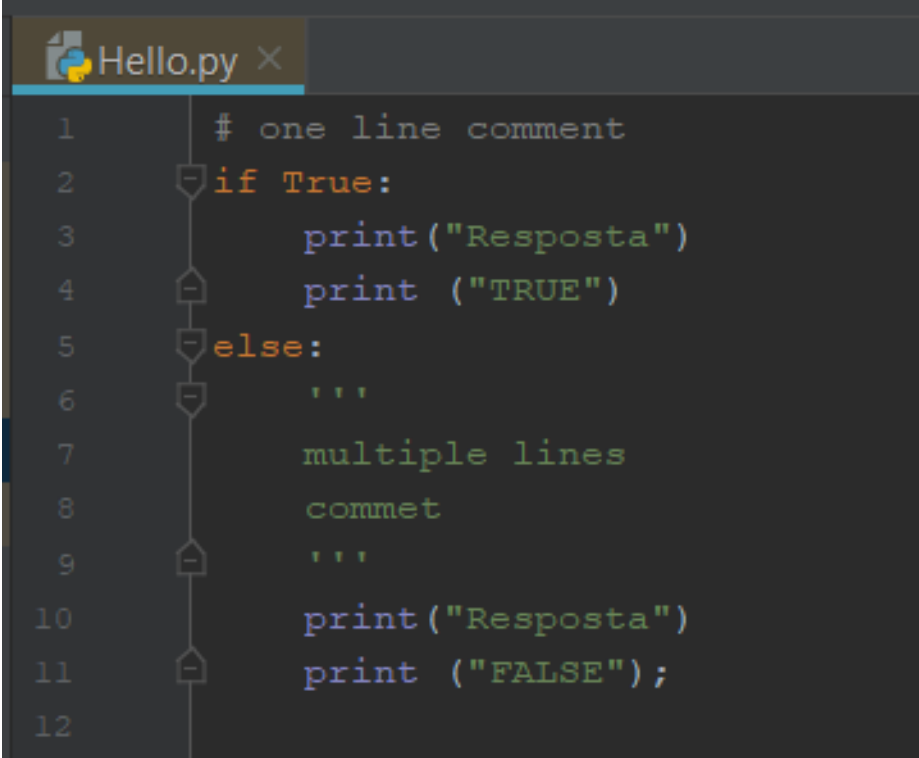




LEARN PYTHON PROGRAMMING

Sintaxe Básica

- A indentação é usada em Python para delimitar blocos de código. O número de espaços pode variar, porém todos os *statements* (instruções) dentro de um mesmo bloco, devem possuir a mesma quantidade de espaços
- A primeira linha que compõe *statements* como *if*, *else*, *while*, *def*, e *class* deve ser finalizada com dois pontos :
- O ponto e vírgula é opcional ao final de um *statement*
- Comentários de apenas uma linha começam por # e de múltiplas linhas por '''
- Arquivos de programas Python tem extensão .py



```
1      # one line comment
2      if True:
3          print("Resposta")
4          print ("TRUE")
5      else:
6          '''
7              multiple lines
8              comment
9          '''
10         print("Resposta")
11         print ("FALSE");
12
```

Variáveis

- Python é dinamicamente tipado, você não precisa declarar o tipo das variáveis
- A declaração acontece automaticamente no momento em que um valor é atribuído à variável
- Variáveis podem mudar de tipo, simplesmente com uma nova atribuição de um tipo diferente
- Python permite atribuir um único valor à várias variáveis ao mesmo tempo
- Assim como permite atribuir múltiplos objetos à múltiplas variáveis

```
max = 10           # integer
value = 150.0      # float point
name = "Python"    # string
nothing = None     # null value
```

```
x = 1
x = "String value"
```

```
a = b = c = 1
```

```
a, b, c = 1, 2, "Python"
```

Tipos de Dados

Boolean

True / False

```
if (number % 2) = 0:  
    even = True  
else:  
    even = False
```

Numbers

Integers, Floats, Fractions and
Complex Numbers

```
a = 5  
b = 7.3  
c = 2 + 3j
```

Strings

Sequences of Unicode
Characters

```
s = "This is a string"
```

Lists

Ordered sequences of values

```
a = [ 1, 2.2, "Python"]
```

Tuples

Ordered immutable
sequences of values

```
t = (2, "Tuple", "95")
```

Sets

Unordered bags
of values

```
week = {'Mon', 'Tue',  
        'Wed', 'Thu', 'Fri', 'Sat',  
        'Sun'}
```

Dictionaries

Unordered bags of
key-value pairs

```
d = {'value':5, 'key':125}
```

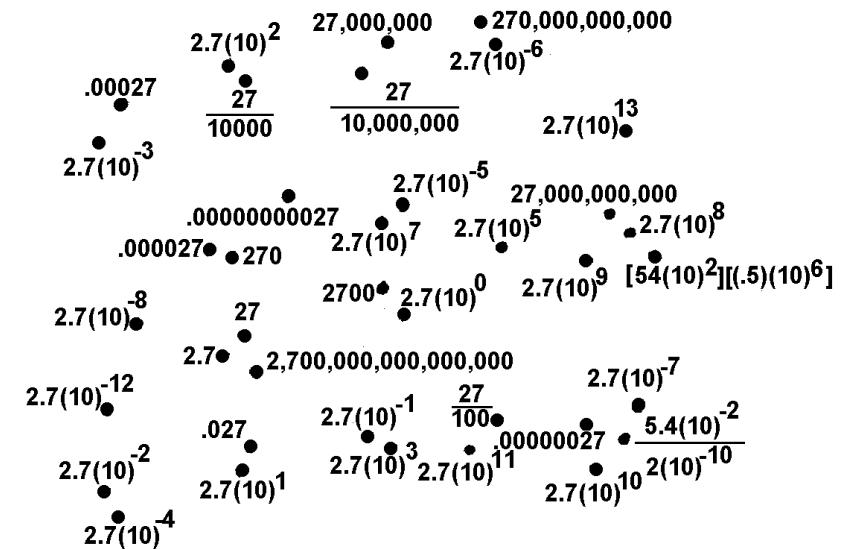
Tipos Booleanos

- Em Python, o tipo booleano (bool) é uma especialização do tipo inteiro (int)
- O verdadeiro é chamado de True e é igual a 1.
- O falso é chamado de False e é igual a 0.
- Os valores abaixo são considerados falsos:
 - False (falso)
 - None (nulo)
 - O objeto None, que é do tipo NoneType do Python, representa o nulo e é avaliado como falso pelo interpretador.
 - 0 (zero)
 - "" (string vazia)
 - [] (lista vazia)
 - () (tupla vazia)
 - {} (dicionário vazio)
 - Outras estruturas com o tamanho igual a zero



Tipos Numericos

- Números são objetos imutáveis em Python, isso significa que seus valores não podem ser alterados
- Existem 3 tipos de dados para números em Python
 - int (signed integer)
 - Também chamado apenas de **integer**, comporta números positivos e negativos sem casas decimais
 - float (floating point)
 - Representa números reais e são escritos com ponto decimal, dividindo um inteiro em partes fracionárias, também podendo ser escrito em notação científica com “e” indicando potencia de 10 (ex: 2.5e2)
 - complex
 - São escritos por dois valores reais, a parte real e a parte imaginária na forma (real + IMAG J). Neste caso o número i ($\sqrt{-1}$) é designado pela letra j. (Não são muito utilizados)



Funções mais comuns com tipos numéricos

Função	Descrição
int (x)	Converte x em um inteiro
float (x)	Converte x em um ponto-flutuante
abs (x)	Retorna o valor absoluto de x
exp (x)	Retorna o exponencial de x (e^x)
log (x)	Retorna o logaritmo natural de x (inverso da função exponencial)
pow (x, y)	Retorna o valor de x elevado à potencia y
sqrt (x)	Retorna a raiz quadrada de x
round (x, y)	Retorna x arredondado em y casas decimais

Strings

- String, assim como numbers, são objetos imutáveis em Python. Dessa forma um update só pode ser possível com a alocação de um novo objeto com o novo conteúdo
- Python não suporta tipos “char”. Um char em Python é considerado uma String de tamanho 1
- Uma String pode ser atribuída de várias formas diferentes
 - ‘Uma String’
 - “Outro exemplo de String”
 - """String com
múltiplas linhas"""
- Strings iniciam sempre com índice 0
- Pode se usar operações como slicing ([], [:]), concatenação (+), repetição (*) e *membership* (in)



Funções e métodos mais comuns com tipo String

Função	Descrição
<code>str (num)</code>	Converte um número em String
<code>len (str)</code>	Retorna o tamanho de uma String
<code>str_value.count (s)</code>	Retorna a quantidade de conjuntos <code>s</code> presentes na string
<code>str_value.isalpha ()</code>	Retorna False se a string contiver algum caracter que não seja letras
<code>str_value.isdigit ()</code>	Retorna False se a string contiver algum caracter que não seja número
<code>str_value.lower ()</code>	Retorna a string transformada em minúsculos
<code>str_value.upper ()</code>	Retorna a string transformada em maiúsculos
<code>str_value.replace (old, new)</code>	Substitui uma porção da string por outro conteúdo
<code>str_value.strip ()</code>	Retira espaços em branco no começo e no fim da string
<code>str_value.title ()</code>	Retorna a string capitalizada (iniciais em maiúscula)
<code>str_value.split (delimiter)</code>	Separa uma string conforme um delimitador. É o inverso do <code>join()</code>
<code>str_value.join (sequence)</code>	Junta cada item da string com um delimitador especificado. É o inverso do <code>split()</code> .

Listas

- Listas são grupos de itens ou elementos indexados, não necessariamente do mesmo tipo
- Listas são mutáveis, isso significa que seu conteúdo pode ser alterado sem que um novo objeto seja criado
- Para atribuir elementos a uma lista é necessário que estejam entre colchetes e separados por vírgula, ex:

`l = [item1, item2, ... Itemn]`

- Os índices em uma lista sempre se iniciam em 0 (zero)
- Permitem duplicações
- Listas podem conter sublistas que podem conter sublistas e assim por diante, tornando esse tipo de dado extremamente versátil
- Pode se usar operações como slicing (`[]`, `[:]`), concatenação (+), repetição (*) e *membership* (`in`)

Grocery List

<input type="checkbox"/> Milk	<input type="checkbox"/> Orange Juice
<input type="checkbox"/> Eggs	<input type="checkbox"/> Flour
<input type="checkbox"/> Bread	
<input type="checkbox"/> Intel Core i7-3930k CPU	

Funções e métodos mais comuns com tipo Lista

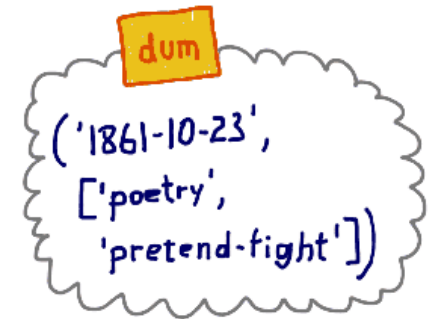
Função	Descrição
<code>len(lista)</code>	Retorna o tamanho total da lista
<code>max(lista)</code>	Retorna o maior elemento da lista
<code>min(lista)</code>	Retorna o menor elemento da lista
<code>list(tupla)</code>	Converte uma tupla em uma lista
<code>lista.append(obj)</code>	Adiciona um objeto à lista
<code>lista.insert(index, obj)</code>	Insere um objeto à lista em determinada posição
<code>lista.count(obj)</code>	Retorna a quantidade de vezes que um determinado objeto ocorre na lista
<code>lista.index(obj)</code>	Retorna o primeiro índice de ocorrência de um determinado objeto
<code>lista.remove(obj)</code>	Remove um objeto da lista
<code>lista.reverse()</code>	Reverte o ordenamento da lista
<code>lista.sort()</code>	Ordena a lista

Tuplas

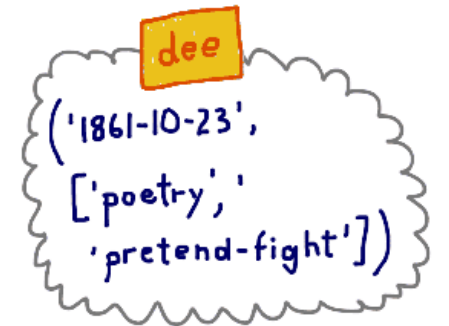
- Tuplas são tipos muito semelhantes às listas, porém imutáveis
- Por esse motivo, tuplas são mais eficientes que as listas, dessa forma quando não existe a necessidade de alteração de seu conteúdo, devem ser escolhidas preferencialmente
- Como são imutáveis, protegem os dados contidos nela de alterações acidentais
- Permitem duplicações
- Para atribuir elementos a uma tupla é necessário que estejam entre parêntesis e separados por vírgula, ex:

```
t = (item1, item2, ... Itemn)
```
- Para tuplas de um elemento apenas é necessário incluir a vírgula após o elemento

```
t = (item1,)
```
- Índices funcionam da mesma forma das listas, assim como os métodos e funções



dum
('1861-10-23',
 ['poetry',
 'pretend-fight'])



dee
('1861-10-23',
 ['poetry',
 'pretend-fight'])

Métodos das tuplas

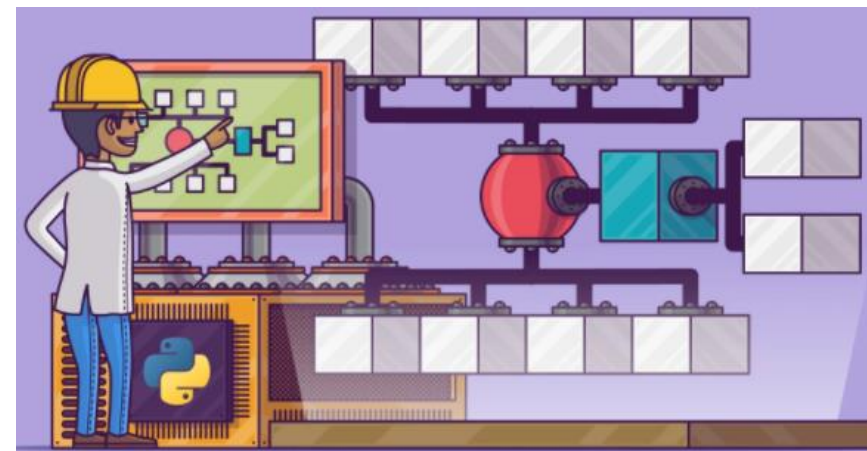
Função	Descrição
t.count(obj)	Retorna a quantidade de vezes que um determinado objeto ocorre na lista
t.index(obj)	Retorna o primeiro índice de ocorrência de um determinado objeto

Sets

- Sets são estruturas disponíveis como builtins do Python, utilizadas para representar coleções desordenadas de elementos únicos.
- Os elementos não são armazenados em uma ordem específica e confiável;
- Sets são mutáveis, isso significa que seu conteúdo pode ser alterado sem que um novo objeto seja criado
- Para atribuir elementos a um set é necessário que estejam entre chaves e separados por vírgula, ex:

`s = {item1, item2, ... Itemn}`

- Elementos em um Set não podem ser acessados através de índices ou chaves mas podem ser iterados por um loop
- Não permitem duplicações
- Pode se usar operações de *membership* (in)



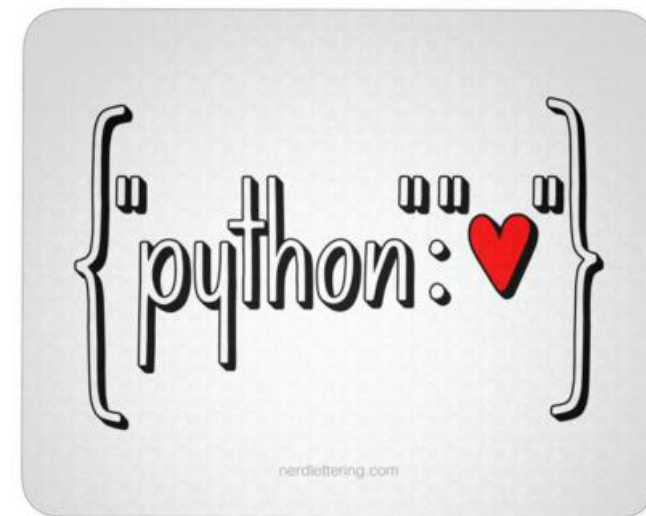
Funções e métodos mais comuns com tipo Set

Função	Descrição
<code>len(set)</code>	Retorna o tamanho total do set
<code>max(set)</code>	Retorna o maior elemento do set
<code>min(set)</code>	Retorna o menor elemento do set
<code>set(tupla)</code>	Converte uma tupla em um set
<code>set_obj.add(obj)</code>	Adiciona um objeto ao set
<code>set_obj.clear()</code>	Remove todos os elementos do set
<code>set_obj.copy()</code>	Retorna uma cópia do set
<code>set_obj.difference(other_set)</code>	Retorna apenas os itens que não existem no set parâmetro
<code>set_obj.remove(obj)</code>	Remove um objeto do set

Dicionários

- Dicionários são uma implementação de hash table que consiste em uma lista de pares de chave-valor, sem ordenação
 - Chaves, devem ser tipos imutáveis e usualmente são utilizadas strings ou números
 - Valores podem ser qualquer tipo de objeto Python
 - Para atribuir elementos a um dicionário é necessário que estejam entre chaves, separados por vírgula, e com os pares chave, valor separados por : (dois pontos)
- ```
dic = {'k1':1, 'k2':10, ... 'kn':n}
```
- Os elementos de um dicionário podem ser acessados ou alterados através de sua chave entre colchetes

```
dic['k1'] = 2
```

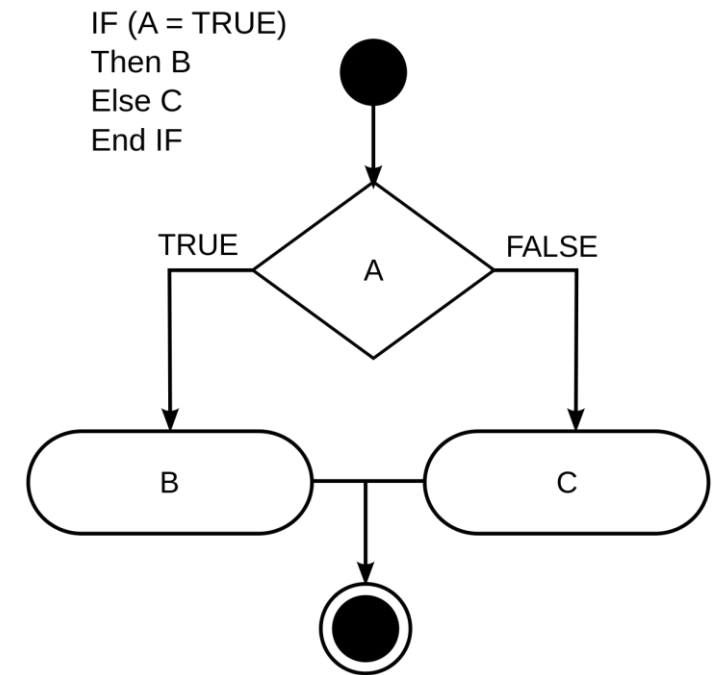


## Funções métodos mais comuns com tipo Dicionário

| Função                                   | Descrição                                                                 |
|------------------------------------------|---------------------------------------------------------------------------|
| <code>len(dict)</code>                   | Retorna o número de elementos do dicionário                               |
| <code>str(dict)</code>                   | Retorna a representação em formato string do dicionário                   |
| <code>dict.keys()</code>                 | Retorna a lista de chaves do dicionário                                   |
| <code>dict.values()</code>               | Retorna a lista de valores do dicionário                                  |
| <code>dict.items()</code>                | Retorna a lista de chave, valor do dicionário                             |
| <code>dict.get(key, default=None)</code> | Retorna o valor de uma chave ou um valor default caso não seja encontrada |
| <code>dict.update(dict2)</code>          | Insere um elemento (chave, valor) no dicionário                           |
| <code>dict.clear()</code>                | Remove todos os elementos do dicionário                                   |

## Estruturas de Controle de Fluxo

- Podemos dividir as estruturas de controle de fluxo em 2 grupos
  - Estruturas de Seleção
  - Estruturas de Repetição
- Estrutura de Seleção são utilizadas para decidir qual fluxo de execução deverá ser tomado pelo programa, dada uma determinada condição ou um conjunto de condições
- São representadas em Python, assim como em outras linguagens pelas instruções:
  - if
  - elif
  - else



## Estruturas de Controle de Fluxo

- Estruturas de Seleção sempre utilizam *expressões booleanas* para representar uma determinada condição
- True e False são Objetos imutáveis da classe *bool* e assume-se que qualquer valor diferente de 0 (zero) E diferente de null são considerados True, todos os outros valores considerados False
- Estruturas de Seleção podem ser aninhadas e combinadas juntamente com operadores (lógicos, aritméticos e relacionais), podendo tornar-se complexas estruturas de decisão

```
Selection.py x
1 var1 = 100
2 if var1:
3 print(var1)
4
```

```
Selection x
C:\dev\workspace_git\DataScienceFac
100
Process finished with exit code 0
```

```
Selection.py x
1 var1 = 100
2 if var1:
3 print('OK')
4 else:
5 print('NOK')
```

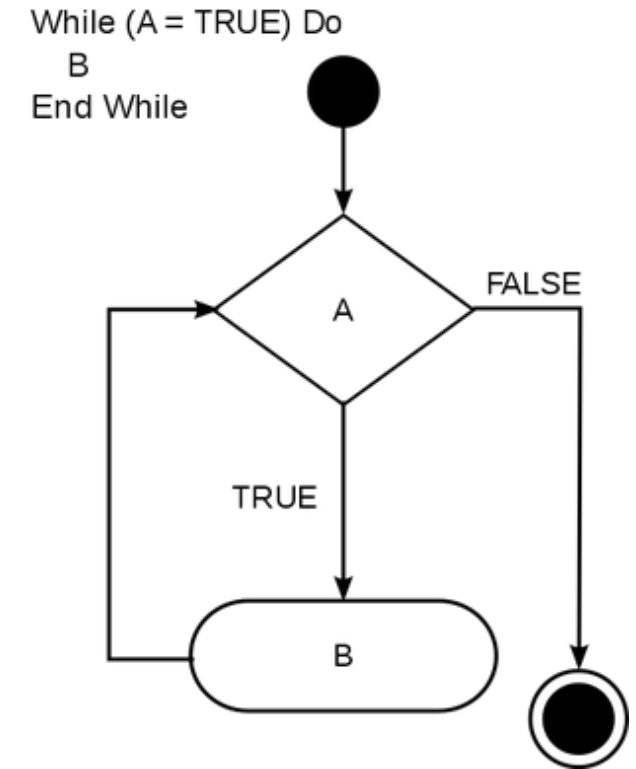
```
Selection x
C:\dev\workspace_git\DataScienceFac
OK
Process finished with exit code 0
```

```
Selection.py x
1 var = 100
2 if var < 150:
3 if var == 150:
4 print("150")
5 elif var == 100:
6 print("100")
7 elif var <= 50:
8 print("<= 50")
9 else:
10 print(">= 150")
```

```
Selection x
C:\dev\workspace_git\DataScienceFac
100
Process finished with exit code 0
```

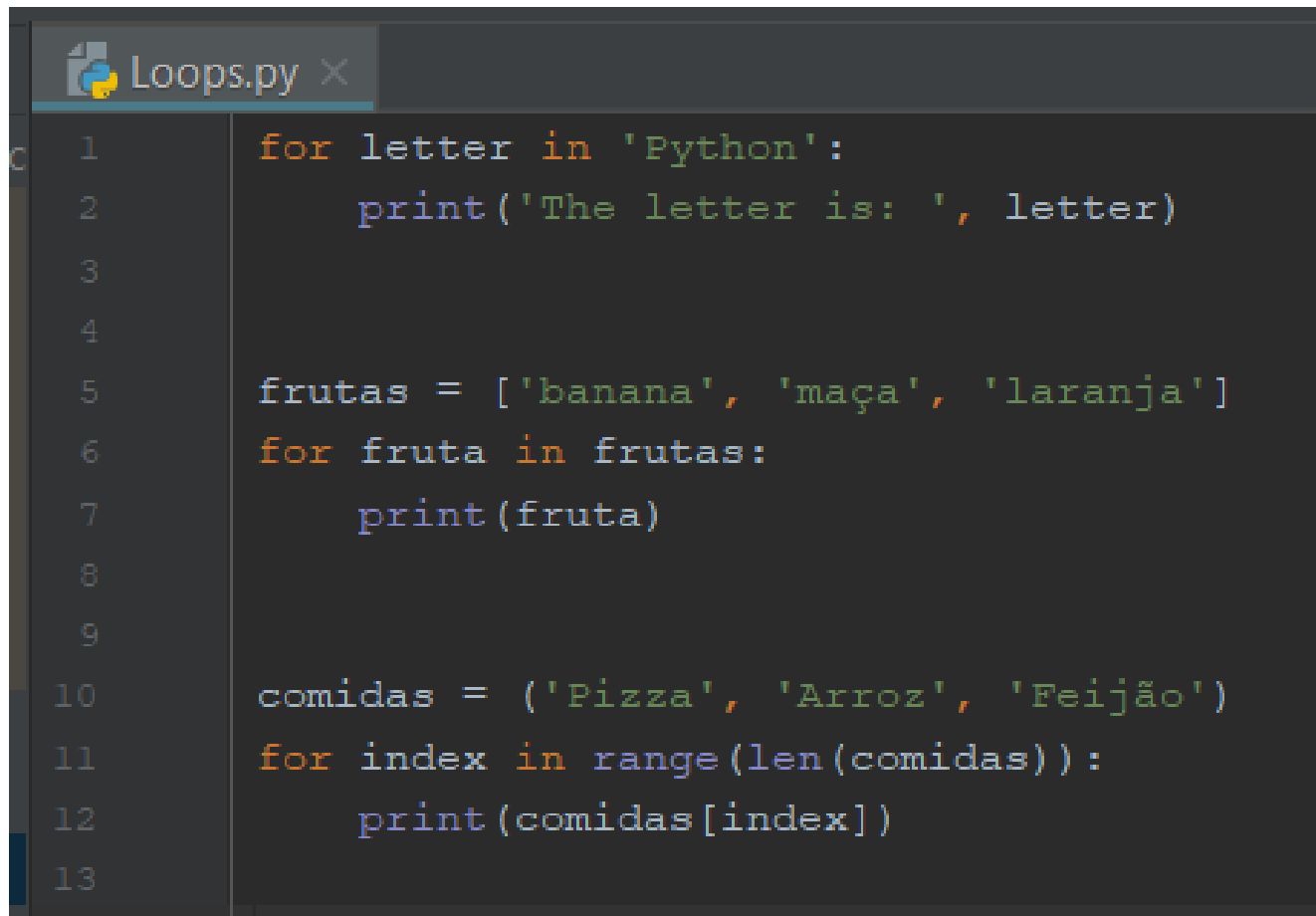
## Estruturas de Controle de Fluxo

- Estruturas de Repetição são utilizadas para instruir o programa a repetir um determinado conjunto de código enquanto uma determinada condição seja satisfatória
- São utilizadas também para iterar em coleções de dados ou listas
- Em Python, existem 2 tipos de estruturas de repetição:
  - for
  - while
- O for é mais utilizada na iteração de listas ou quando há um número conhecido de iterações a serem executadas
- O while é mais recomendado quando uma determinada condição seja satisfatória para a execução do trecho de código

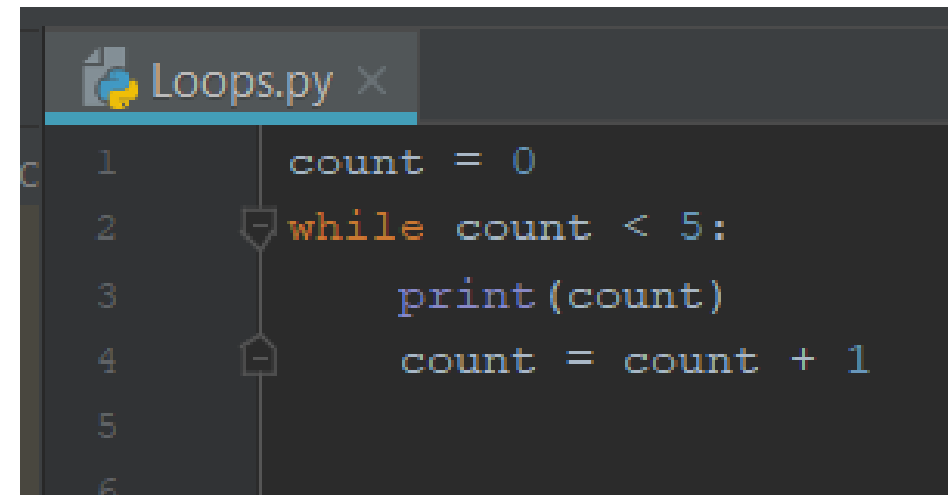




## Estruturas de Controle de Fluxo



```
Loops.py x
1 for letter in 'Python':
2 print('The letter is: ', letter)
3
4
5 frutas = ['banana', 'maça', 'laranja']
6 for fruta in frutas:
7 print(fruta)
8
9
10 comidas = ('Pizza', 'Arroz', 'Feijão')
11 for index in range(len(comidas)):
12 print(comidas[index])
13
```



```
Loops.py x
1 count = 0
2 while count < 5:
3 print(count)
4 count = count + 1
5
6
```

## Estruturas de Controle de Fluxo

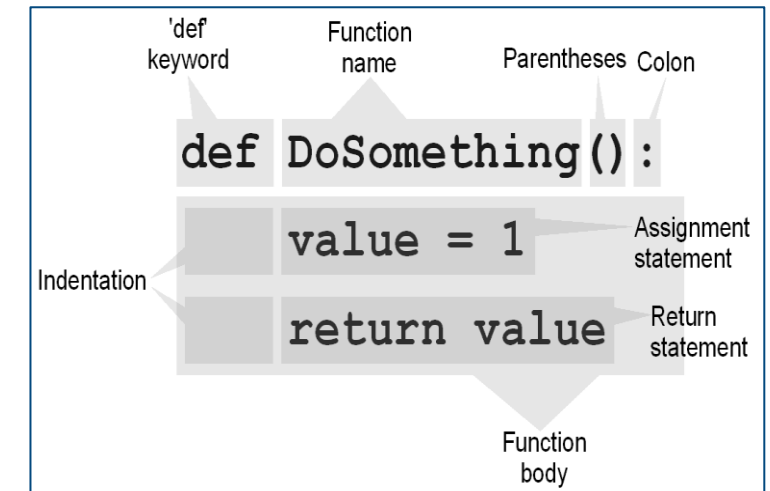
- Estruturas de Repetição oferecem instruções de controle em determinadas situações:
- **break** – permite a saída do fluxo antes da condição de finalização do laço ser atingida
- **continue** – Instrui o interpretador a passar para a próxima iteração imediatamente, ignorando os comandos subsequentes
- **pass** – permite que um determinado trecho de código seja escrito sintaticamente correto, porém em sua execução não será executada nenhuma instrução naquele momento

## Funções

- Simplificadamente, uma função é um conjunto de instruções do qual pode-se dar um nome
- Funções são uma forma muito útil de organizar o código
- Promovem o reaproveitamento, uma vez que um podem ser invocadas em diversos pontos do mesmo programa, sem a necessidade reescrita desse trecho de código
- A sintaxe de definição de uma função é a seguinte:

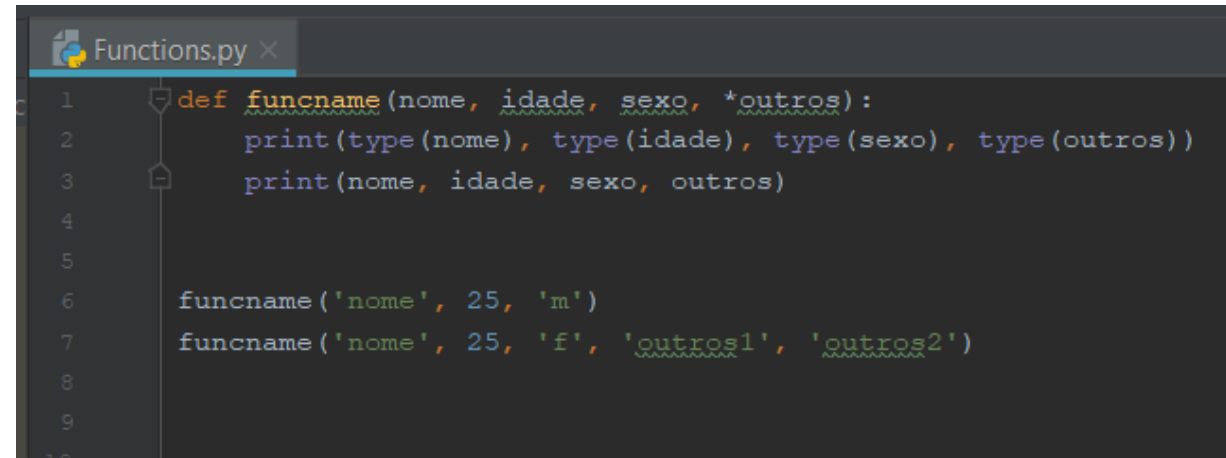
```
def NOME(PARÂMETROS):
 COMANDOS
```

- Note a indentação na parte de comandos, ela é fundamental para que o interpretador identifique quais são de fato, as instruções que pertencem à função

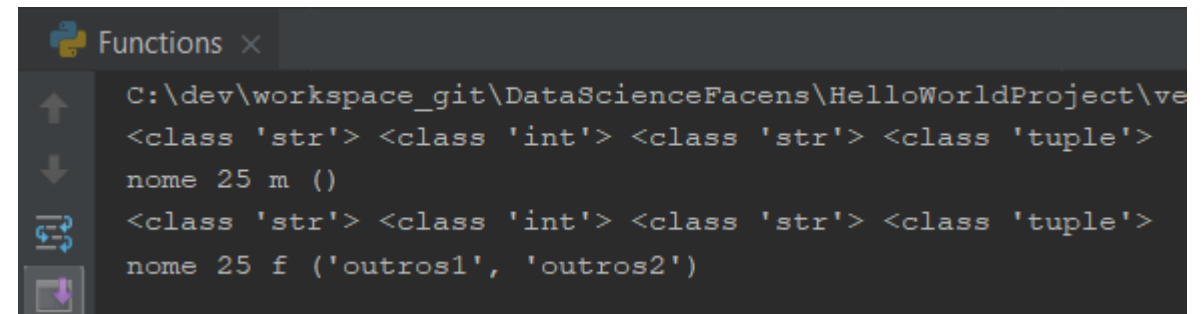


## Mais sobre Funções

- Os parâmetros de uma função podem ser passados na mesma ordem em que foram definidos ou podem ser identificados através da função chamadora, assim não necessitando ser colocados na mesma ordem da definição da função
- Parâmetros de funções podem ter valores *default*, dessa forma não é necessário sua passagem como argumento (desde sejam os últimos argumentos em ordem reversa)
- Funções suportam parâmetros com número indefinido de valores. Para isso é necessário que sejam o último parâmetro da função e o argumento seja precedido por um \* (asterisco). O argumento será passado para a função como uma tupla de valores.
- Funções podem retornar valores, através da instrução *return*. Não é necessário declarar que a função irá ter um retorno



```
Functions.py x
1 def funcname(nome, idade, sexo, *outros):
2 print(type(nome), type(idade), type(sexo), type(outros))
3 print(nome, idade, sexo, outros)
4
5
6 funcname('nome', 25, 'm')
7 funcname('nome', 25, 'f', 'outros1', 'outros2')
```



```
Functions x
C:\dev\workspace_git\DataScienceFacens\HelloWorldProject\ve
<class 'str'> <class 'int'> <class 'str'> <class 'tuple'>
nome 25 m ()
<class 'str'> <class 'int'> <class 'str'> <class 'tuple'>
nome 25 f ('outros1', 'outros2')
```

## Operadores Aritméticos

| Operador | Descrição                 | Exemplo                       |
|----------|---------------------------|-------------------------------|
| +        | Adição                    | $10 + 15 = 25$                |
| -        | Subtração                 | $25 - 15 = 10$                |
| *        | Multiplicação             | $10 * 3 = 30$                 |
| /        | Divisão                   | $30 / 4 = 7.5$                |
| //       | Parte inteira da divisão  | $30 // 4 = 7$                 |
| %        | Módulo (Resto da divisão) | $30 \% 4 = 2$                 |
| **       | Exponenciação (Potência)  | $3 ** 4 = 3 * 3 * 3 * 3 = 81$ |

## Operadores Relacionais

| Operador | Descrição      | Exemplo                                                |
|----------|----------------|--------------------------------------------------------|
| ==       | Igual          | 10 > 15 é falso; 'Python' == 'python' é falso          |
| !=       | Diferente      | 5 != 7 é verdadeiro, 'Python' != 'python' é verdadeiro |
| >        | Maior          | 5 > 5 é falso; 5 > 2 é verdadeiro                      |
| <        | Menor          | 7 < 12 é verdadeiro; 7 < 5 é falso                     |
| >=       | Maior ou igual | 5 >= 5 é verdadeiro; 5 >= 6 é falso                    |
| <=       | Menor ou igual | 5 <= 5 é verdadeiro; 5 < 7 é verdadeiro                |



## Operadores Lógicos

| Operador   | Descrição                                    |
|------------|----------------------------------------------|
| <b>and</b> | Retorna True se os dois operandos forem True |
| <b>or</b>  | Retorna True se um dos operandos for True    |
| <b>not</b> | Negativa ou reverte a estado do operando     |

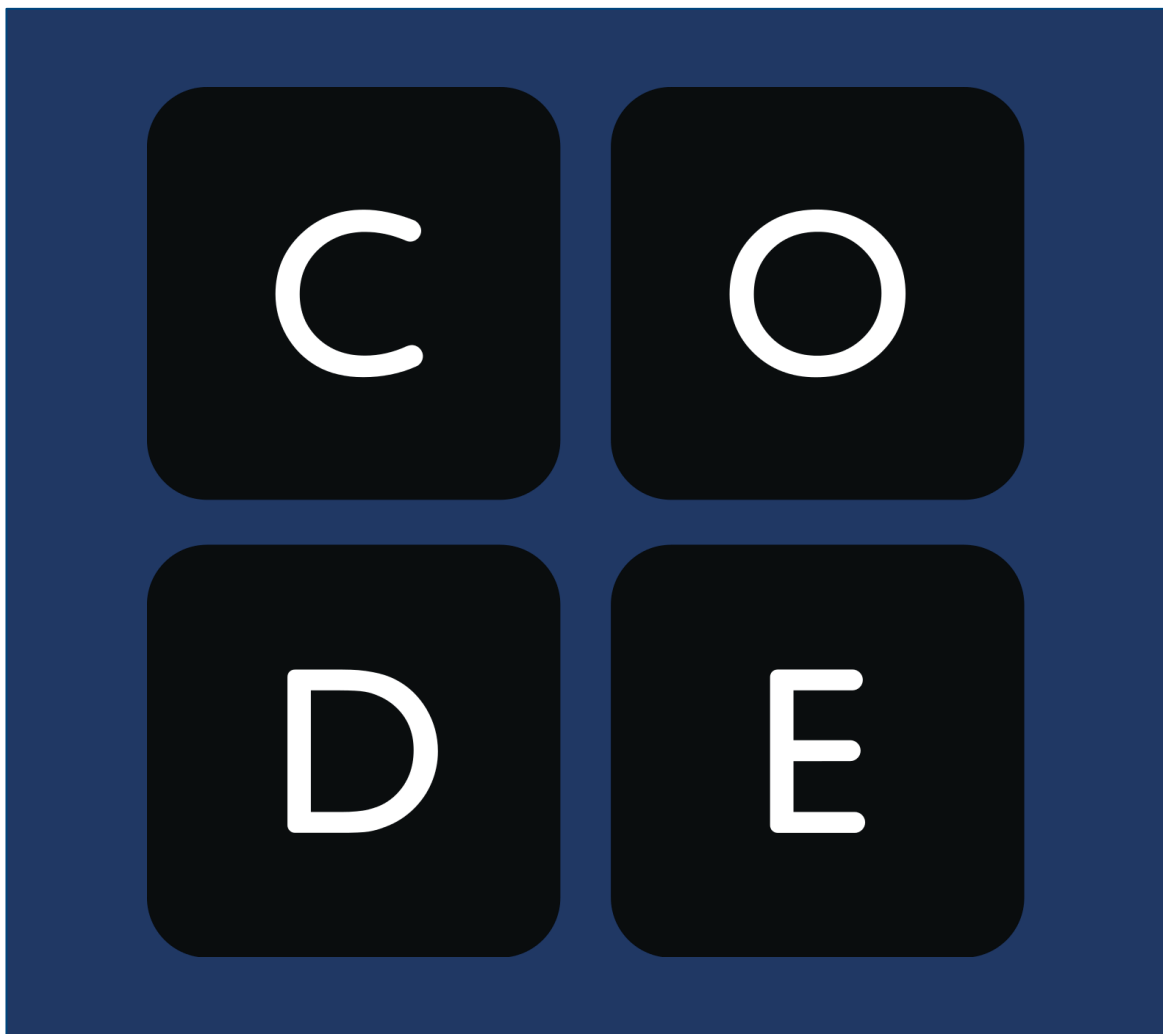
## Operadores de Membro

| Operador  | Descrição                               |
|-----------|-----------------------------------------|
| <b>in</b> | True se encontra a expressão pesquisada |

## Operadores de Identidade

| Operador  | Descrição                                                        |
|-----------|------------------------------------------------------------------|
| <b>is</b> | True se os operadores apontarem para o mesmo endereço de memória |

## Challenges Time



## Referências



- <https://www.python.org/>
- <https://python.org.br/>
- <http://www.diveintopython.net/>
- <https://www.w3schools.com/python>
- <https://www.codecademy.com/tracks/python>
- <https://www.jetbrains.com/pycharm/>



Facens

AQUI TEM ENGENHARIA