

DATA MINING

Análise de Dados e Data Mining

Tema da Aula: **Aula 1 - Introdução ao Python**

Prof.: **Dino Magri**

Coordenação:

Prof. Dr. Adolpho Walter
Pimazzi Canton

Profa. Dra. Alessandra de
Ávila Montini

- Contatos:

- E-mail: professor.dinomagri@gmail.com
- Twitter: https://twitter.com/prof_dinomagri
- LinkedIn: <http://www.linkedin.com/in/dinomagri>
- Site: <http://www.dinomagri.com>

Coordenação:

Prof. Dr. Adolpho Walter
Pimazzi Canton

Profa. Dra. Alessandra de
Ávila Montini

Currículo

- **(2014-Presente)** – Professor no curso de Extensão, Pós e MBA na Fundação Instituto de Administração (FIA) – www.fia.com.br
- **(2018-Presente)** – Pesquisa e Desenvolvimento de Big Data e Machine Learning na Beholder (<http://beholder.tech>)
- **(2013-2018)** – Pesquisa e Desenvolvimento no Laboratório de Arquitetura e Redes de Computadores (LARC) na Universidade de São Paulo – www.larc.usp.br
- **(2012)** – Bacharel em Ciência da Computação pela Universidade do Estado de Santa Catarina (UDESC) – www.cct.udesc.br
- **(2009/2010)** – Pesquisador e Desenvolvedor no Centro de Computação Gráfica – Guimarães – Portugal – www.ccg.pt
- **Lattes:** <http://lattes.cnpq.br/5673884504184733>

Material das aulas

- Caso esteja utilizando seu próprio computador, realize o download de todos os arquivos e salve na **Área de Trabalho** para facilitar o acesso.
 - Lembre-se de instalar os softwares necessários conforme descrito no documento de Instalação (**InstalaçãoPython3v1.2.pdf**).
- Nos computadores da FIA os arquivos já estão disponíveis, bem como a instalação dos softwares necessários.

Conteúdo da Aula

- Objetivo
- Introdução ao Python
- Exercícios
- Referências Bibliográficas

Conteúdo da Aula

- **Objetivo**
- Introdução ao Python
- Exercícios
- Referências Bibliográficas

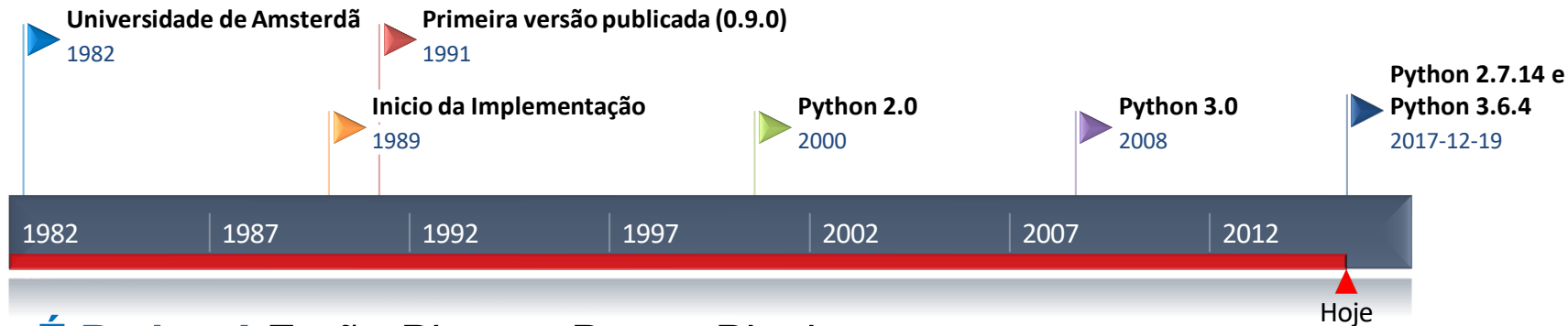
Objetivo

- O objetivo dessa aula é introduzir os conceitos básicos sobre a linguagem de programação **Python** com foco no desenvolvimento de aplicações.

Conteúdo da Aula

- Objetivo
- **Introdução ao Python**
- Exercícios
- Referências Bibliográficas

- **Guido van Rossum**

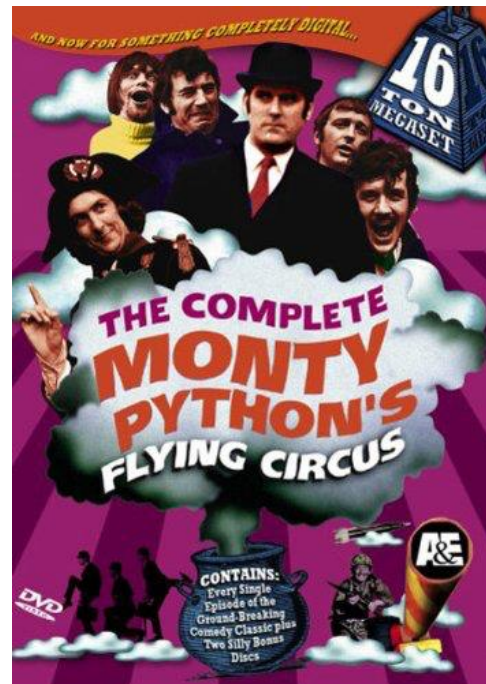


- **É Python!** E não Phytton, Pyton, Phython
- Licença de código aberto, compatível com GPL
- Interpretada - Ambiente Interativo
- Linguagem de altíssimo Nível (VHLL)



Introdução

- **Monty Python's Flying Circus**
 - Quando Guido estava implementando a linguagem de programação Python ele estava lendo o roteiro de "Monty Python's Flying Circus", uma série de comédia da BBC da década de 70.
 - Ele pensou que o nome precisava ser pequeno, único e ligeiramente misterioso.



Fonte: <https://docs.python.org/2/faq/general.html#why-is-it-called-python>

Fonte: <http://ecx.images-amazon.com/images/I/51HrI5sUrwL.jpg>

Por que Python para Data Mining?

Por que utilizar Python?

- **Qualidade de Software**

- O código do Python foi **projetado** para ser **legível**, **reutilizável** e **fácil manutenção**
- Suporte para mecanismos de reutilização de software, como **Programação Orientada à Objetos** (OOP)

Por que utilizar Python?

- **Produtividade no desenvolvimento**
 - Python **aumenta** a **produtividade do desenvolvedor** em muitas vezes além das linguagens compiladas com C, C++ e Java.
 - **20 a 30% da quantidade de linhas de código**, se comparado com C++ e Java.
 - Redução na quantidade de linhas a serem analisadas para **encontrar um erro** ou **dar manutenção** no código

Por que utilizar Python?

- **Sintaxe Clara**, muito próxima do pseudocódigo:

```
nome = input('Digite seu nome: ')\nprint("Olá {}".format(nome))
```

Por que utilizar Python?

```
#include <stdio.h>

int main(){
    char nome[200];
    printf("Digite seu nome: ");
    scanf("%s", nome);
    printf("Olá %s \n", nome);
    return 0
}
```

C

Java



```
public class Hello {
    public static void main(String args[]) {
        java.util.Scanner s = new java.util.Scanner(System.in);
        System.out.print("Digite seu nome: ");
        String nome = s.nextLine();
        System.out.println("Olá " + nome );
    }
}
```

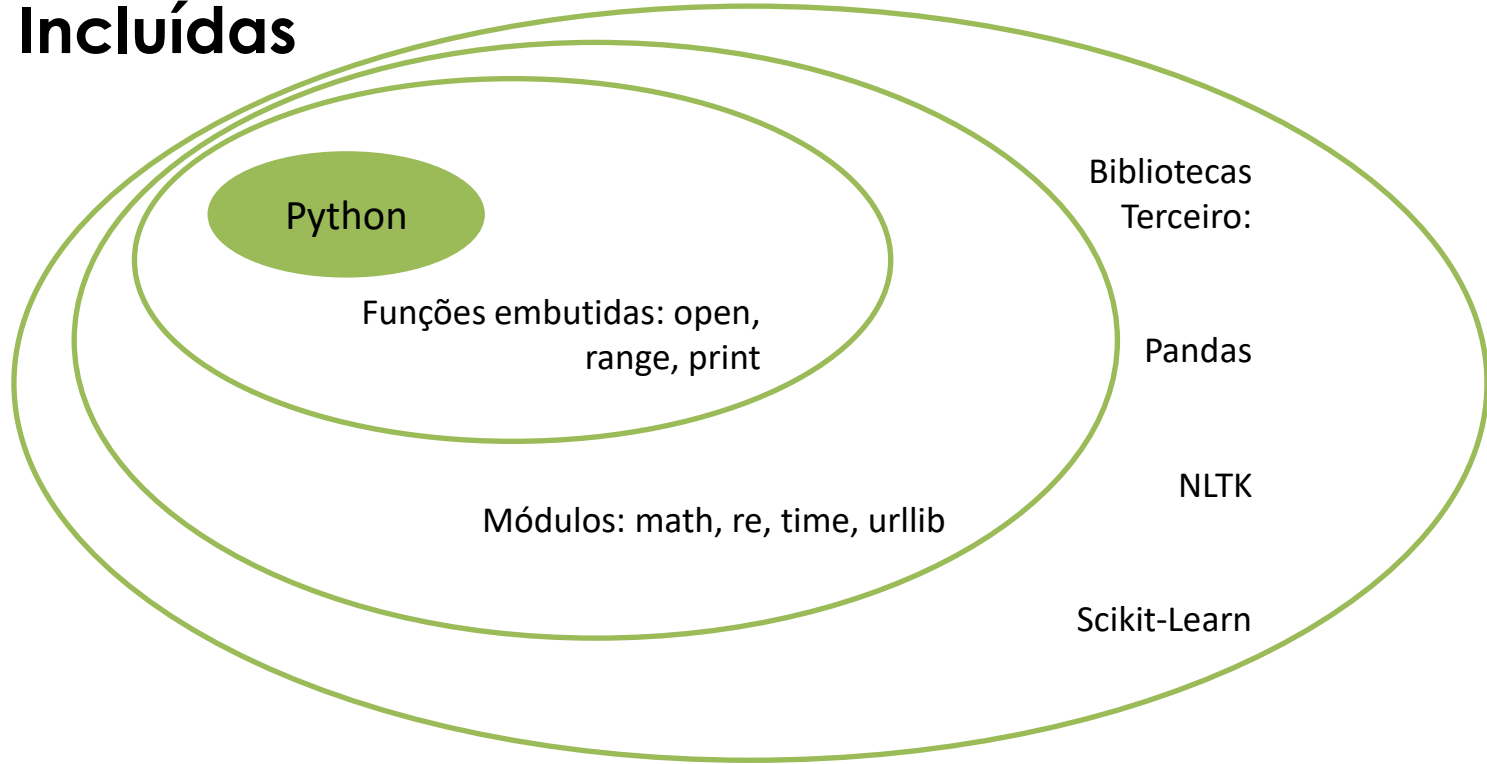

Por que utilizar Python?

- **Baterias Incluídas**

- Python vem com **uma grande** coleção de funcionalidades (*standard library*)
 - <https://docs.python.org/3/library/>
- Python pode ser estendido com bibliotecas de terceiros
 - Mais de 60.000 no **Python Package Index**
 - <https://pypi.org/>

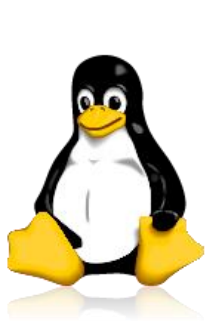
Por que utilizar Python?

- **Baterias Incluídas**



Por que utilizar Python?

- Roda em diversas plataformas

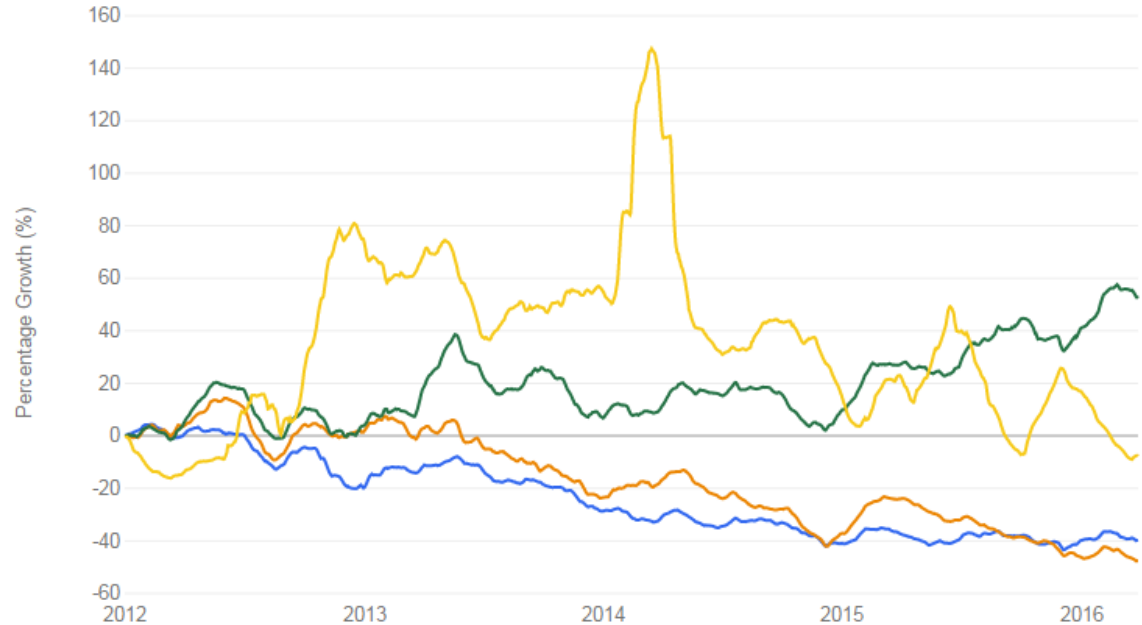


Por que utilizar Python?

- **Simples, Legível e Elegante**
- **Python ...**
 - ... vai direto ao ponto.
 - ... é simples de usar.
 - ... permite focar no problema, sem perder tempo na sintaxe.
 - ... permite que o primeiro contato com a linguagem seja menos complicado possível.
 - ... permite evoluir dentro da linguagem com aplicações reais.

Por que utilizar Python?

- Crescimento na intenção de trabalhar com:



Cases de Sucesso



Como conversar com Python?

- Como vimos Python é uma linguagem de programação e portanto é **necessário aprender** essa linguagem para realizar as tarefas que queremos.
- O primeiro passo é **instalar** o Python em seu computador.
 - Para um passo a passo detalhado acesse o documento:
[InstalaçãoPython3v1.2.pdf](#)

Como conversar com Python?

- **Linux & Mac OS X**

- **Linux**

- A maioria das distribuições Linux já possui o interpretador Python pré-instalado.

- **Mac**

- Também tem um interpretador Python pré-instalado
- Em ambos é possível atualizar para a última versão no site oficial da linguagem – <https://www.python.org/>

Como conversar com Python?

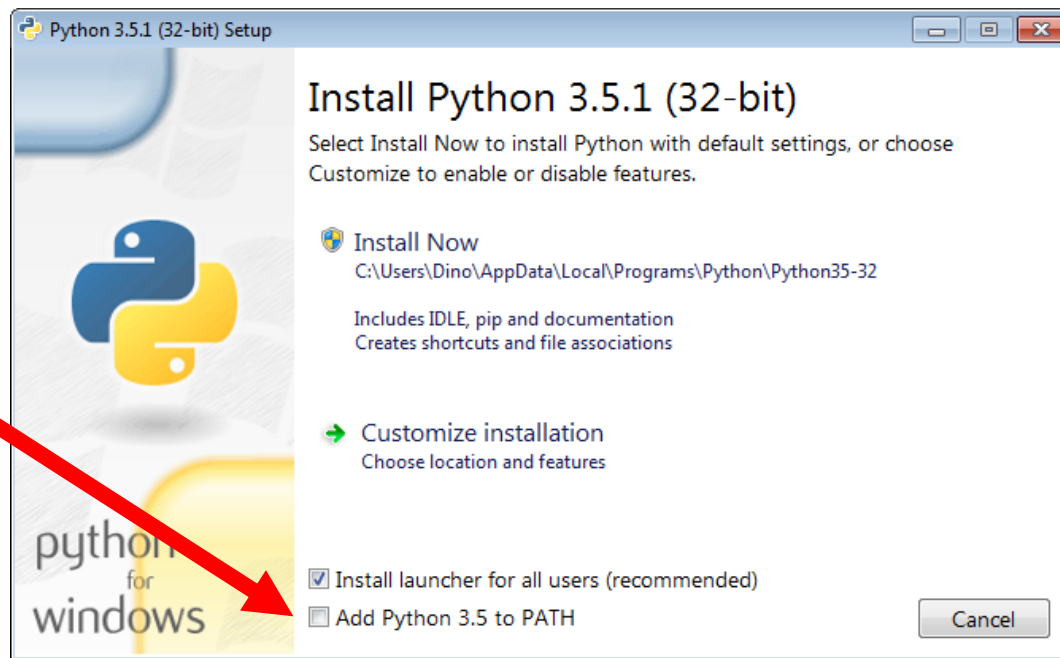
- **Windows 7, 8 e 10**

- Acesse a página: <https://www.python.org/downloads/>
- Realize o download da versão 3.6.x ou superior
- Dê preferência para a versão **64 bits**:
 - » <https://www.python.org/ftp/python/3.6.3/python-3.6.3-amd64.exe>
- Durante o processo de instalação é necessário selecionar a opção de adicionar o Python no PATH do Windows.

Como conversar com Python?

IMPORTANTE!!!!

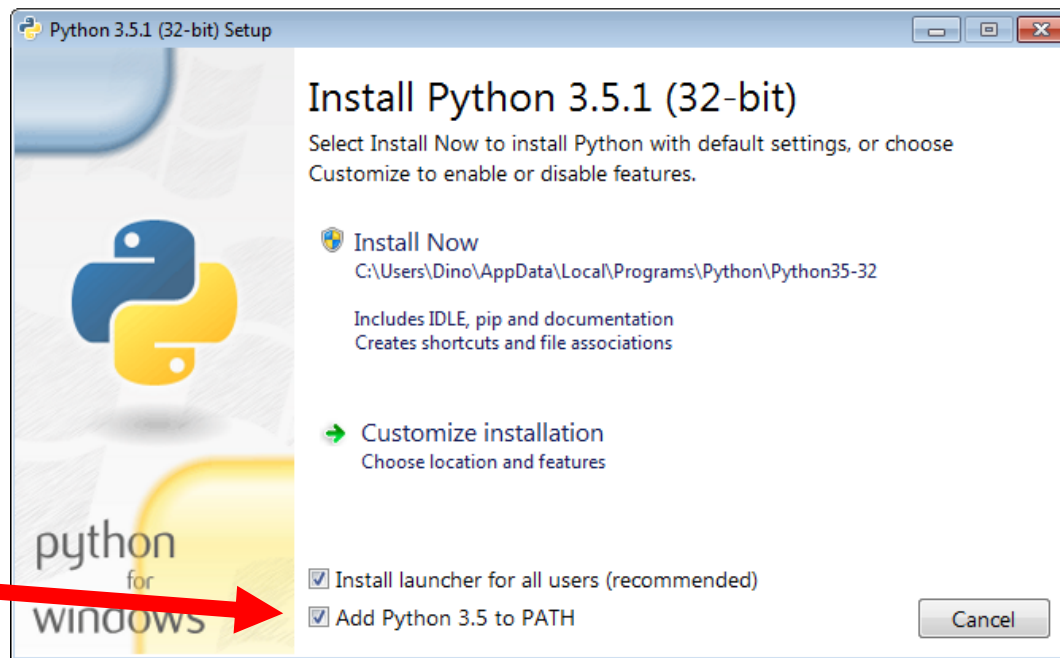
**Marque
essa opção!**



Como conversar com Python?

IMPORTANTE!!!!

**Deve ficar
assim ...**



Como conversar com Python?

- E agora?
 - Como posso praticar para aprender essa nova linguagem?
- Existem duas maneiras de praticar:
 - **Modo Interativo** – É excelente para testar comandos e obter respostas imediatas.
 - **Modo Editor** - É utilizado para desenvolver os programas. A extensão utilizada na hora de salvar o arquivo deve ser .py.
 - Exemplo: **meu-arquivo.py**

Como conversar com Python?

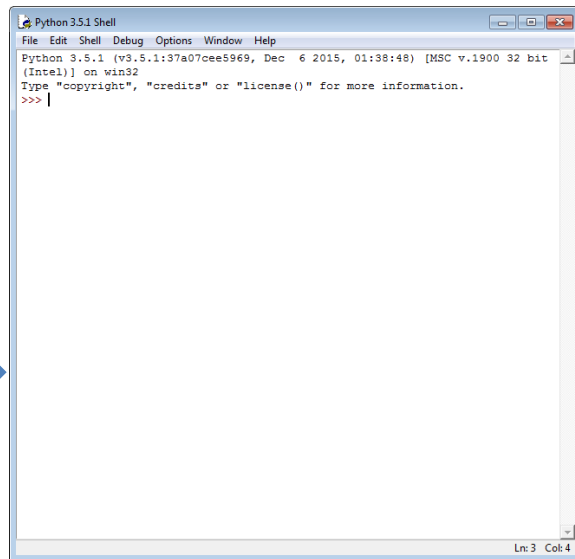
- E onde posso escrever essa linguagem?
 - Temos que ter uma **IDE!!!**
- E o que é uma IDE?
 - É um ambiente integrado de desenvolvimento (*Integrated Development Environment*)
- Ou seja, **é um programa onde iremos adicionar código** para que o Python consiga interpretar e executar o que está sendo solicitado.

Como conversar com Python?

- Exemplos de IDEs:
 - IDLE (já vem junto instalado com o Python!)
 - Sublime Text - <http://www.sublimetext.com>
 - Ninja IDE - <http://ninja-ide.org>
 - Syper - <https://pythonhosted.org/spyder/>
 - PyCharm - <https://www.jetbrains.com/pycharm/>
- O Python já vem com uma IDE simplificada, chamada **IDLE**
 - Para abrir, clique em Iniciar ou aperte o botão do Windows do teclado → no campo de busca → digite IDLE!
 - Caso utilize Linux é necessário instalar: `sudo apt install idle3`

Como conversar com Python?

- **Ambiente de Desenvolvimento Integrado para Python**
 - Permite **editar**, **rodar**, **navegar** e **depurar** programas em Python
 - **Gratuito**
 - Fácil de utilizar 😊
 - Disponível em todas as plataformas
 - Este é o Python Shell
 - **>>>** são chamados de *prompt*



Como conversar com Python?

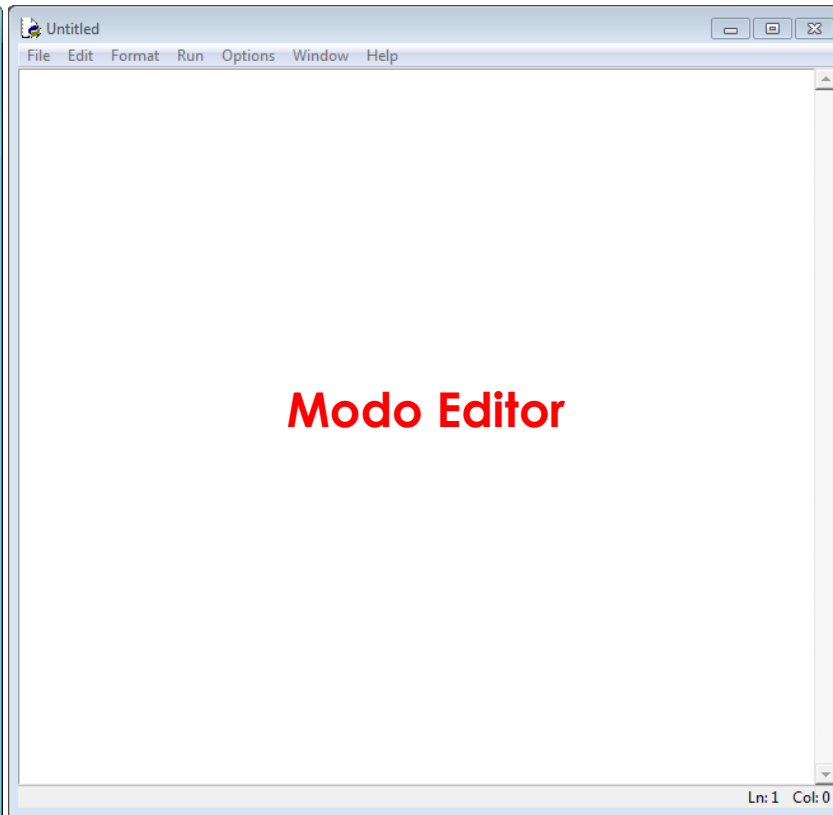


A screenshot of the Python 3.5.1 Shell window. The title bar reads "Python 3.5.1 Shell". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The text area shows the following output:

```
Python 3.5.1 (v3.5.1:37a07cee5969, Dec 6 2015, 01:38:48) [MSC v.1900 32 bit  
(Intel)] on win32  
Type "copyright", "credits" or "license()" for more information.  
>>> |
```

The status bar at the bottom right indicates "Ln: 3 Col: 4".

Modo Interativo



Como conversar com Python?

- Antes de criar nossos códigos, temos que notar algumas regras básicas de sintaxe:
 - Não tem necessidade de colocar ponto e virgula no final dos comandos!
 - Sem delimitadores de código – { }
 - **A endentação é obrigatória**
 - **Comentar código:**
 - # - Toda linha iniciada com #, será um comentário
 - """ Tudo o que estiver dentro de três aspas duplas será um comentário """

A endentação é obrigatória

```
if idade < 10:
    → print("Criança")
else:
    → if idade < 18:
        → print("Adolescente")
    → else:
        → print("Adulto")
```

O dois pontos no final da linha, sempre inicia um novo bloco de código

Como conversar com Python?

- Ótimo! Agora já sabemos como podemos criar códigos em Python!
- O modo iterativo no IDLE é muito bom e prático, porém não é possível salvar de maneira relativamente simples o código que criamos.
- Portanto, para facilitar os estudos, iremos utilizar uma biblioteca chamada **Jupyter!**

E o que são Bibliotecas?

- Lembram das baterias incluídas que comentei anteriormente?
- Uma das grandes vantagens do Python é a possibilidade de utilizar **diversas funcionalidades que já foram implementadas** por outras pessoas e que estão disponíveis.

Bibliotecas Python

- Em Python existem diversas bibliotecas disponíveis para áreas como:
 - Bibliotecas fundamentais para Computação Científica
 - IPython Notebook, Numpy, **Pandas**, SciPy
 - Matemática e Estatística
 - **Statsmodels** e SymPy
 - Aprendizagem de Máquina
 - **Scikit-learn**, TensorFlow e Theano

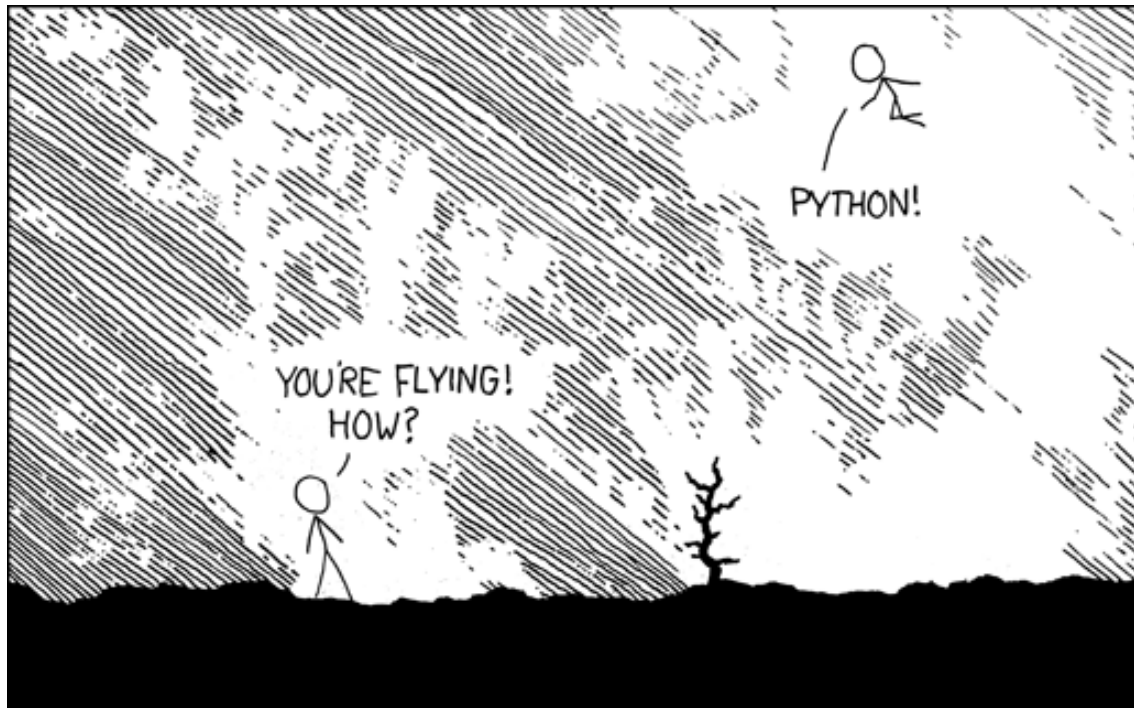
Bibliotecas Python

- Em Python existem diversas bibliotecas disponíveis para áreas como:
 - Visualização e Plotagem
 - **Matplotlib**, Bokeh, Seaborn, Plotly, Basemap, NetworkX
 - Biblioteca para Data Mining e Processamento de Linguagem Natural
 - Scrapy, NLTK, Pattern e Gensim

Poderees Ilimitados !!!!

- As bibliotecas são compostas por módulos:
 - Um módulo é composto por códigos Python em um arquivo com a extensão `.py` que pode ser utilizado em outro arquivo de código Python!
 - Para utilizar um código que está em um arquivo **A.py** no arquivo **B.py** será necessário utilizar o comando `import` para importar as definições e comandos que estão em A.
 - Ou seja, dentro do arquivo `B.py`, adicione: `import A`

Poderes Ilimitados !!!!



Fonte: <https://xkcd.com/353/>

Poderes Ilimitados !!!!



Fonte: <https://xkcd.com/353/>

Poderees Ilimitados !!!!

- A biblioteca padrão do Python (<https://docs.python.org/3/library/>) contém uma vasta quantidade de módulos, como por exemplo:
 - `sys` – Contém parâmetros específicos do sistema
 - `math` – Contém funções matemáticas prontas para serem utilizadas!
 - `datetime` – Tipos básicos de data e hora

Poderes Ilimitados !!!!

- Exemplos de uso desses módulos:

```
>>> import sys
>>> print(sys.version)
3.5.1 (v3.5.1:37a07cee5969, Dec 6 2015, 01:38:48) [MSC v.1900 32 bit
(Intel)]
```

```
>>> import math
>>> math.factorial(4)
24
```

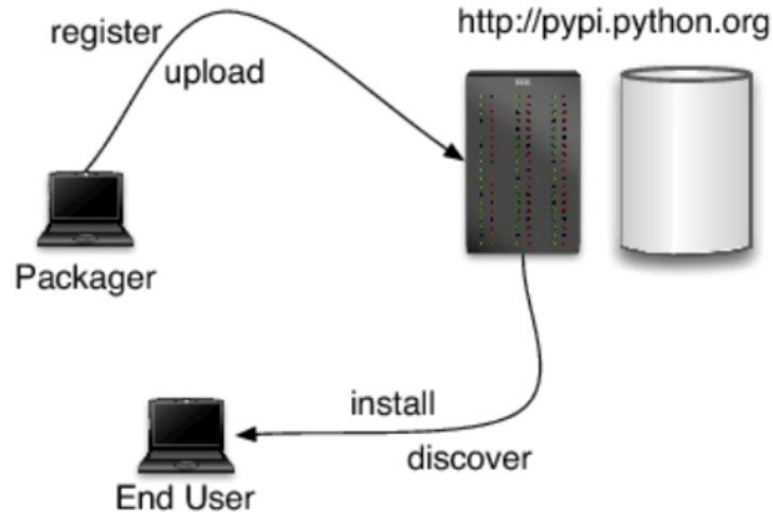
```
>>> import datetime
>>> print(datetime.date.today())
2017-06-30
```

Como faço para utilizar outras bibliotecas?

- Professor, você disse no começo da aula que Python tem mais de 60000 pacotes que podem ser utilizados, porém na biblioteca padrão do Python não tem tudo isso! **E agora?**
- Será necessário instalar essas bibliotecas! E isso é feito de maneira bem simples!
- O Python tem um gerenciador que possibilita procurar, instalar e remover pacotes.
- Esse gerenciador, chama-se **PIP! - Python Package Index**

Como faço para utilizar outras bibliotecas?

- Como funciona?



Fonte: <http://www.aosabook.org/en/packaging.html>

Como faço para utilizar outras bibliotecas?

- Como instalar o pip?
 - Nas versões 3.3 ou superior do Python o Pip é instalado por padrão!
 - Na versão 2.7.9 ou superior o Pip também é instalado por padrão!
 - Para as versões antigas do Python acesse:
<https://pip.pypa.io/en/stable/installing/>

Como faço para utilizar outras bibliotecas?

- Como utilizá-lo?
 - Abra o CMD ou Terminal e digite:
 - Para pesquisar, digite:

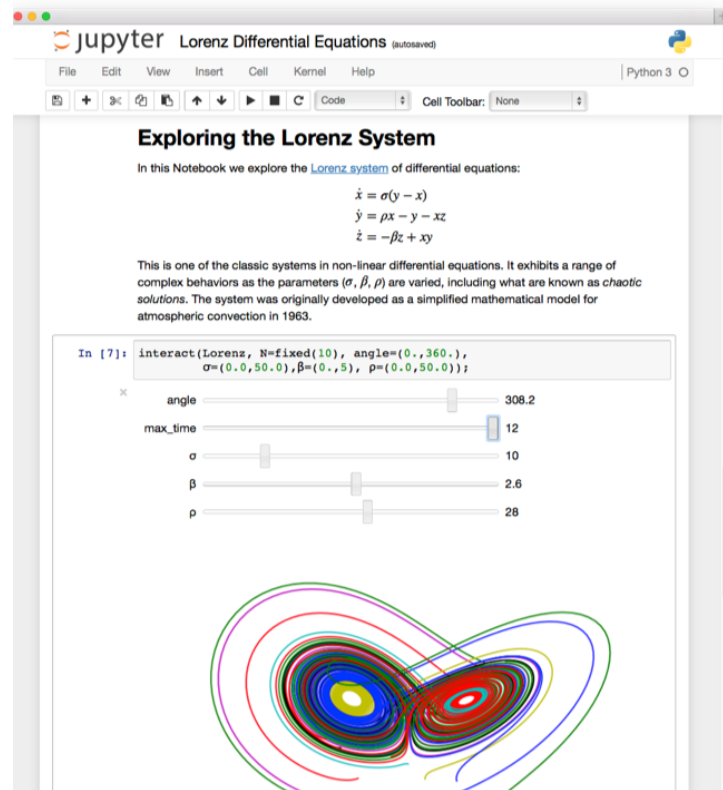
```
$ pip search [nome do pacote]
```
 - Para instalar, digite (utilize o parâmetro `-U` para atualizar um pacote caso seja necessário)

```
$ pip install [nome do pacote]
```
 - Para desinstalar, digite:

```
$ pip uninstall [nome do pacote]
```
- **Nota:** Se as versões 2 e 3 do Python coexistirem, é necessário utilizar `pip3` para a versão do Python 3.

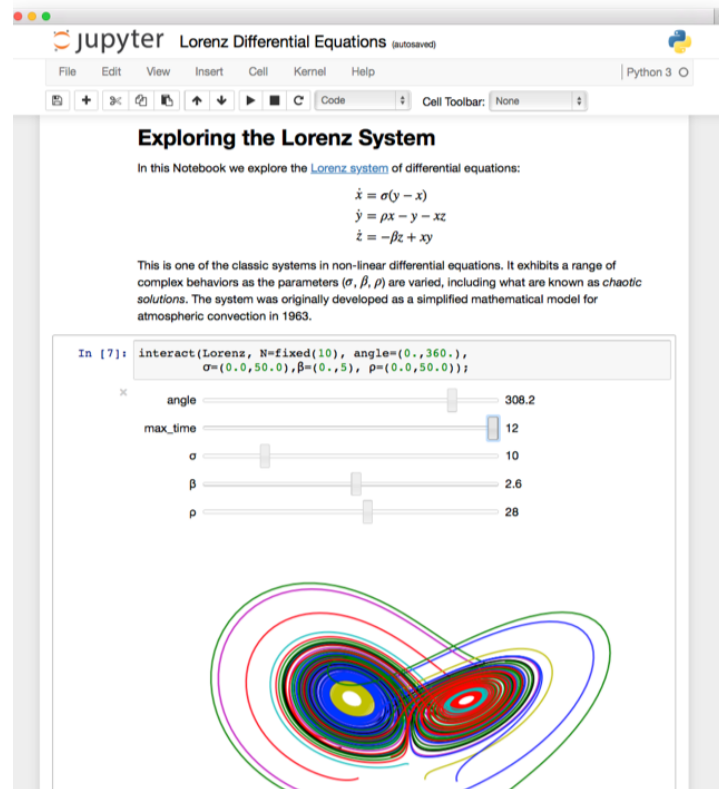
Como faço para utilizar outras bibliotecas?

- Para testar o uso do `pip` iremos instalar um pacote chamado **Jupyter**.
- Jupyter é uma aplicação web que permite criar e compartilhar **notebooks** que contém código, equações, visualizações e textos explicativos.



Como faço para utilizar outras bibliotecas?

- Podemos criar códigos para limpeza e transformação de dados, simulações numéricas, modelagem estatística, aprendizagem de máquina, entre outros!
- <http://jupyter.org/>



Como faço para utilizar outras bibliotecas?

- **Vamos instalar o pacote Jupyter!**
- Abra o CMD ou Terminal e digite:

```
$ pip install jupyter
```

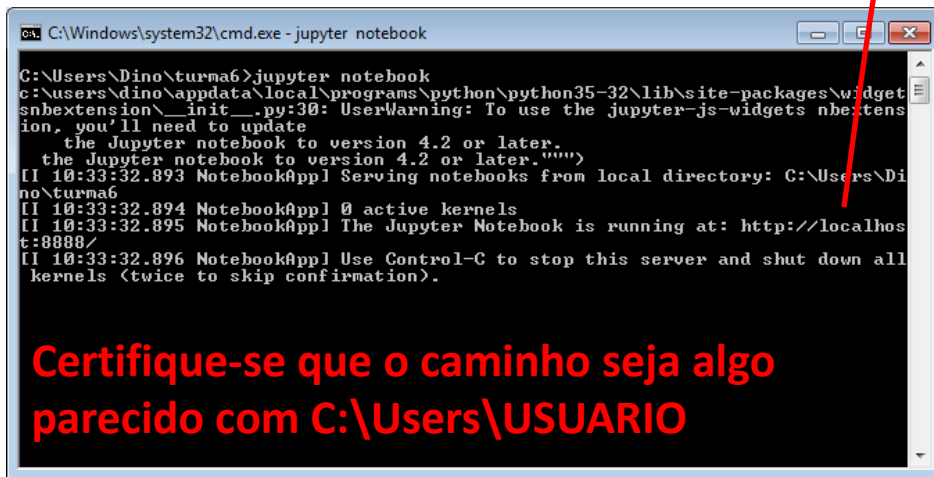
- Note que durante a instalação, o pip irá baixar todos os pacotes necessários para instalar corretamente o pacote Jupyter!

Como faço para utilizar outras bibliotecas?

- Para utilizar o Jupyter, abra o CMD ou Terminal e digite:

```
$ cd Desktop
```

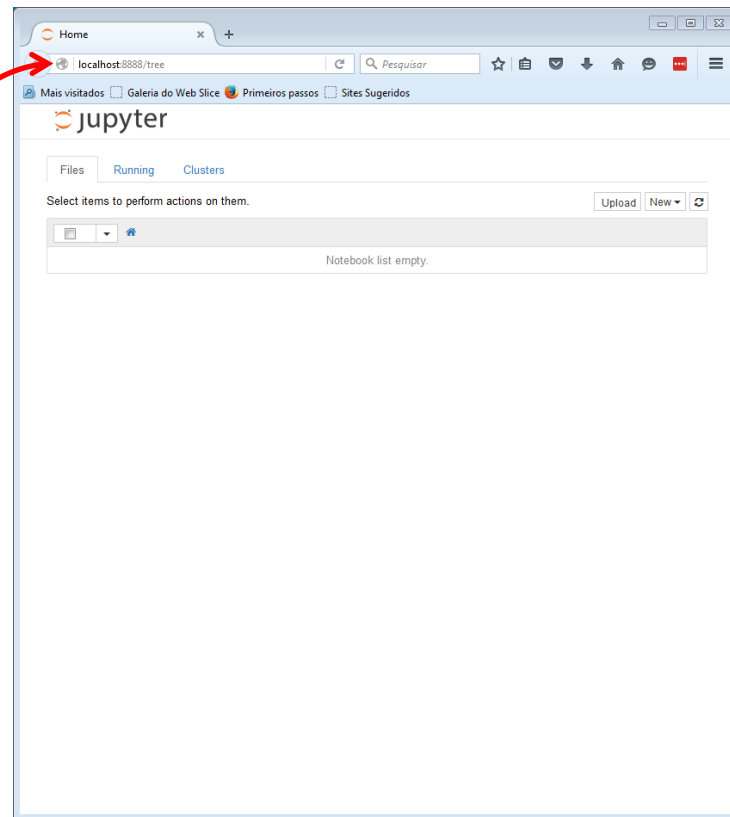
```
$ jupyter notebook
```



```
C:\Windows\system32\cmd.exe - jupyter notebook

C:\Users\Dino\turna6>jupyter notebook
c:\users\dino\appdata\local\programs\python\python35-32\lib\site-packages\widgetsnbextension\_init_.py:30: UserWarning: To use the jupyter-js-widgets nbextension, you'll need to update
  the Jupyter notebook to version 4.2 or later.
  the Jupyter notebook to version 4.2 or later.
[I 10:33:32.893 NotebookApp] Serving notebooks from local directory: C:\Users\Dino\turna6
[I 10:33:32.894 NotebookApp] 0 active kernels
[I 10:33:32.895 NotebookApp] The Jupyter Notebook is running at: http://localhost:8888/
[I 10:33:32.896 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
```

Certifique-se que o caminho seja algo parecido com C:\Users\USUARIO



10 minutos de introdução – Jupyter!

- No navegador web (Firefox ou Chrome), digite:
 - <http://localhost:8888/notebooks/>
 - Note que é listado todos os arquivos que existem dentro da pasta Desktop.
- Se os arquivos das aulas estiverem salvos no Desktop, eles irão aparecer na interface do Jupyter.

 Abra o arquivo **"aula1-parte1-Jupyter.ipynb"**

Introdução ao Python

- Iremos aprender os seguintes tópicos introdutórios sobre Python:
 - **Objetos Pythonicos**
 - Strings
 - Listas
 - Sets
 - Dicionários
 - Estruturas de Controle
 - Funções

Objetos Pythônicos

- Programa (ou script) em Python é uma sequência de **definições** e **comandos**.
 - Definições são avaliadas e comandos são executados pelo Python (Lembre-se o modo editor e interativo).
- Comando (ou declaração) instrui o interpretador a fazer algo.

Objetos Pythônicos

- De fato, os **programas irão manipular objetos de dados**.
- Cada objeto tem um **tipo** que define o que os programas podem fazer.
- Objetos podem ser:
 - **Escalar** (e.g. não podem ser subdivididos), ou
 - **Não-escalar** (e.g. tem uma estrutura interna que pode ser acessada).

Objetos Pythônicos

- **Objeto Escalar**

- **int** – utilizado para representar inteiros (e.g. 5 ou 10000)
- **float** – utilizado para representar números reais (e.g. 3.14 ou 27.0)
- **bool** – utilizado para representar valores booleanos (**True** e **False**)
- **None** – utilizado para representar a ausência de valor
- A função interna do Python **type** retorna um tipo de um objeto

```
>>> type(3)
```

```
<type 'int'>
```

```
>>> type(3.0)
```

```
<type 'float'>
```


Objetos Pythônicos

- **Objeto Não-Escalar**

- Iremos ver diferentes tipos de objetos compostos.
- As **strings** são as mais simples desses, são objetos do tipo `str`.
- As strings podem ser escritas utilizando aspas simples ou duplas.
 - `'abc'`
 - `"abc"`
 - `'123'` – essa é uma string de caracteres, não os números.
- Outros objetos Não-Escalar: **Listas**, Sets e Dicionários

Identificadores e atribuições

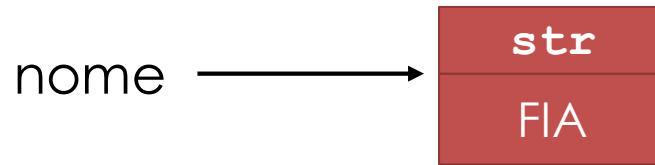
- Um comando do Python muito importante é a atribuição:

```
>>> nome = "FIA"
```

- Esse comando estabelece que `nome` é um identificador (ou variável) e está associado a um **objeto** expressado pelo tipo `string` e tem valor "FIA".

Identificadores e atribuições

- O identificador `nome` faz referência a uma instância da classe `string` que tem o valor `FIA`.



Identificadores e atribuições

- Identificadores (ou **variáveis**) são do tipo *case-sensitive*, o que significa que, uma variável chamada **nome** é diferente de **Nome**.
- É possível armazenar informações como números, textos, listas de números e textos, entre outros tipos de dados.
- O sinal de igual é utilizado para atribuir um valor a uma variável.

```
>>> nome = "FIA"
```

```
>>> print(nome)
```

```
FIA
```

Identificadores e atribuições

- É importante atribuir um valor, antes de utilizar uma variável

```
>>> f
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
NameError: name 'f' is not defined
```

Identificadores e atribuições

- Repare que em nenhum momento foi necessário definir qual o tipo do dado que será armazenado na variável, apenas atribuímos o valor. **Por que?**
- Pois em Python, **o tipo da variável** é definido em tempo de execução do programa.
- Ou seja, Python tem **tipagem dinâmica e forte**.
 - Objetos não podem mudar de tipo
 - É forte pois não há conversão automática de tipo

Identificadores e atribuições

- Objetos e operadores podem ser combinados para formarem **expressões**, cada um denota um objeto de algum tipo.
- A sintaxe para a expressão mais simples é:

$i + j$	Soma
$i - j$	Subtração
$i * j$	Multiplicação
i / j	Divisão
$i \% j$	Resto da divisão
$i ** j$	Exponenciação

<objeto> <operador> <objeto>

Operadores Condicionais

- Resultado Verdadeiro (**True**) e Falso (**False**)

```
>>> print (10 == 15)
```

```
False
```

```
>>> print (10 != 15)
```

```
True
```

```
>>> print ("a" == "a")
```

```
True
```

```
>>> print ("a" != "b")
```

```
True
```

<code>i > j</code>	Retorna True se i for maior que j
<code>i >= j</code>	Retorna True se i for maior ou igual que j
<code>i < j</code>	Retorna True se i for menor que j
<code>i <= j</code>	Retorna True se i for menor ou igual que j
<code>i == j</code>	Retorna True se i e j forem iguais
<code>i != j</code>	Retorna True se i e j não forem iguais

Operadores Lógicos

- **not, and e or**

```
>>> nome = "FIA"  
>>> idade = 35
```

i and j	Retorna True se i e j forem True
i or j	Retorna True se pelo menos um deles for True
not i	Retorna True se i for False; retorna False se i for True

```
>>> nome == "FIA" and idade == 35  
True
```

```
>>> nome == "FIA" or idade > 36  
True
```

```
>>> len(nome) < 10 and not nome == "FIA"  
False
```

Introdução ao Python

- Iremos aprender os seguintes tópicos introdutórios sobre Python:
 - Objetos Pythônicos
 - **Strings**
 - **Listas**
 - Sets
 - Dicionários
 - Estruturas de Controle
 - Funções

Strings

- Na programação, normalmente chamamos um conjunto de caracteres de string.
- Para criar uma string é necessário delimitar o conjunto de caracteres com aspas duplas ou simples.

```
>>> "Dino"
```

```
>>> '123'
```

```
>>> "1+1"
```

Strings

- Podemos substituir símbolos em strings
- Podemos concatenar strings
- Podemos multiplicar strings
- Podemos indexar e fatiar strings
- E muito mais ...

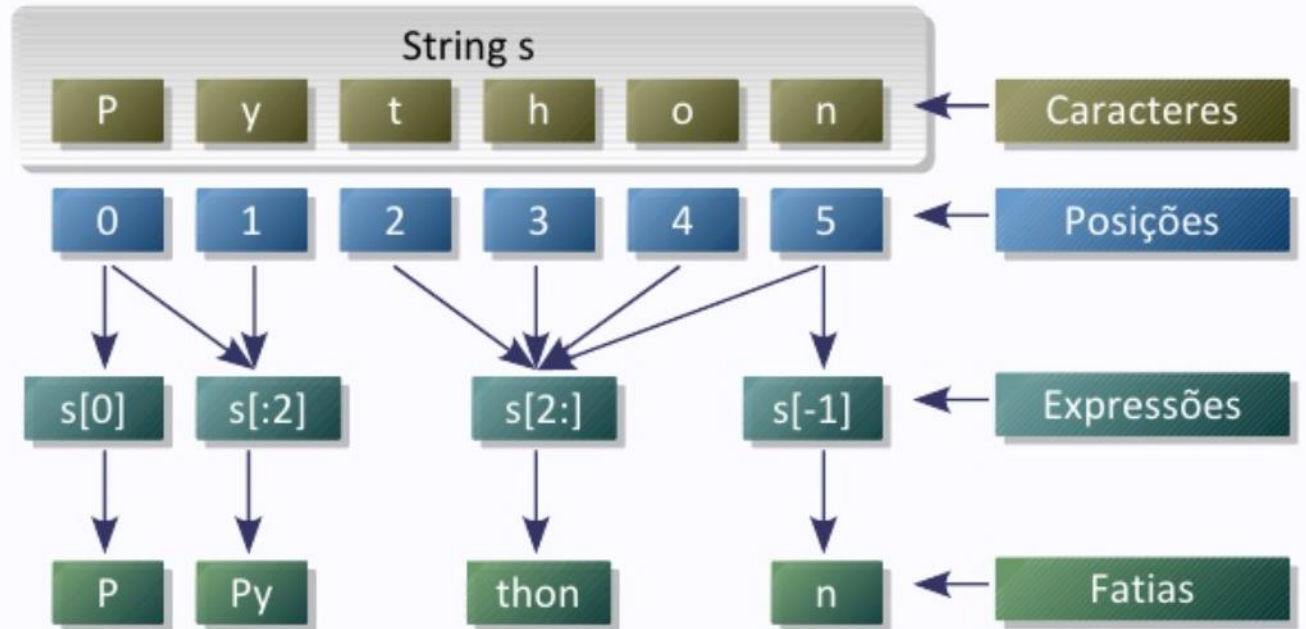
 Abra o arquivo **"aula1-parte2-strings.ipynb"**

O que podemos fazer com strings?

- Podemos substituir símbolos em strings
- Podemos concatenar strings
- Podemos multiplicar strings
- **Podemos indexar e fatiar strings**
- E muito mais ...

Strings

- Indexar (index) e fatiar (slice) strings



Fonte: <http://goo.gl/agfSe5>

O que podemos fazer com strings?

- Podemos substituir símbolos em strings
- Podemos concatenar strings
- Podemos multiplicar strings
- Podemos indexar e fatiar strings
- **E muito mais ...**

O que podemos fazer com strings?

- Como vimos no início da aula, tudo em Python é um objeto, portanto existem ações associadas a cada um desses objetos.
- Em `strings`, existem diversos métodos (ações) que podem ser utilizados, por exemplo:
 - Contar a quantidade de um caractere específico
 - Deixar toda a string em minúsculo, maiúsculo ou no formato de título
 - Verificar se uma string inicia ou finaliza com caracteres desejados
 - Outras ações (métodos) podem ser visualizados em:


<https://docs.python.org/3/library/stdtypes.html#string-methods> Abra o arquivo "aula01-parte2-strings.ipynb"

Listas

- Conjunto linear de valores indexados por um número inteiro.
 1. Índices são iniciados em zero
 2. Tipos mistos
 3. E até outras listas

```
>>> list("abcd")  
['a', 'b', 'd', 'c']
```

```
>>> numeros = [1, 2, 3, 4, 5, 6]
```



O que podemos fazer com listas?

- Podemos concatenar listas
- Podemos modificar o seu conteúdo
- Podemos indexar e fatiar listas
- E muito mais ...



Abra o arquivo "**aula1-parte3-listas.ipynb**"

O que podemos fazer com listas?

- Em listas, também existem diversos métodos (ações) que podem ser utilizados, por exemplo:
 - Adicionar um item ao fim da lista
 - Inserir um item em uma posição específica
 - Contar a quantidade de elementos que aparecem na lista
 - Ordenar os itens da lista
 - Inverter a ordem da lista
 - Outras ações (métodos) podem ser visualizados em:
<https://docs.python.org/3/tutorial/datastructures.html#more-on-lists>

Introdução ao Python

- Iremos aprender os seguintes tópicos introdutórios sobre Python:
 - Objetos Pythônicos
 - Strings
 - Listas
 - **Sets**
 - **Dicionários**
 - Estruturas de Controle
 - Funções

Sets

- É uma coleção de elementos não ordenados sem elementos duplicados.
- É um estrutura que teste e elimina os elementos duplicados.
- Também é possível realizar operações matemáticas, como união, intersecção, diferença, entre outros.

Sets

```
>>> numeros = [1, 2, 1, 2, 3, 4, 5, 5, 6]
```

```
>>> aux = set(numeros)
```

```
>>> print(aux)
```

```
{1, 2, 3, 4, 5, 6}
```



Abra o arquivo **"aula1-parte4-sets.ipynb"**

Dicionários

- É uma coleção de elementos onde é possível utilizar um índice de qualquer tipo imutável.
- Dicionários são indexados por **chaves** (`keys`), que podem ser de qualquer tipo **imutável** (`strings` e `números`).

Dicionários

- O dicionário é um conjunto de **chave** : **valor** (`key` : `value`) não ordenado, com o requerimento que **a chave deve ser única**.
 - chave é o índice.
 - valor é a informação correspondente a chave.
 - { } é utilizado para iniciar um dicionário vazio.
 - : separa os pares índice-valor por vírgula

Dicionários

```
>>> alunos = {'jose' : 35, 'bilbo' : 28}
```

```
print(alunos)
```

```
{'jose' : 35, 'bilbo' : 28}
```



```
>>> alunos['jose']
```

```
35
```

```
>>> alunos['bilbo']
```

```
28
```

O que podemos fazer com dicionários?

- Podemos adicionar uma novo elemento no dicionário.
- Podemos deletar um elemento do dicionário.
- Podemos recuperar todas as chaves ou todos os valores do dicionário.
- Podemos verificar se uma chave existe no dicionário.
- E muito mais ...



Abra o arquivo **"aula1-parte5-dicionarios.ipynb"**

O que podemos fazer com dicionários?

- Em dicionários, também existem diversos métodos (ações) que podem ser utilizados, por exemplo:
 - Recuperar as chaves do dicionário
 - Recuperar os valores do dicionário
 - Atualizar o dicionário com base em outro dicionário

Exercícios de 5 minutos

1. Crie um dicionário chamado `palavras`. Esse dicionário irá conter algumas palavras que aparecem repetidamente em um determinado texto. As palavras estão listas abaixo, crie um dicionário sendo a chave o nome da palavra e o valor sendo a quantidade de vezes que a palavra apareceu.
 1. `big` = 182
 2. `data` = 342
 3. `python` = 423

Exercícios de 5 minutos

2. Utilizando o dicionário `palavras` que foi criado no exercício anterior, crie um programa para imprimir as frases:
 - a) A palavra `python` apareceu **423** vezes.
 - b) As palavras `big`, `data` e `python` apareceram **947** vezes.
- Utilize o dicionário `palavras` para imprimir os valores.

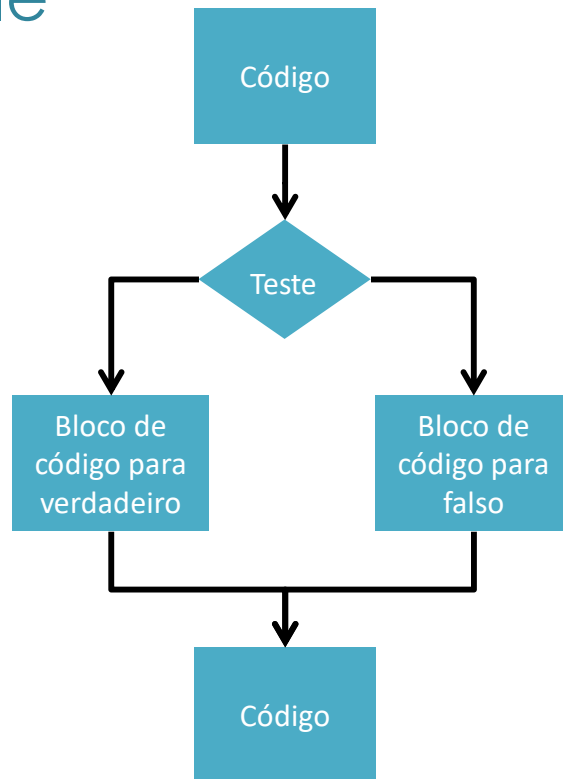
 Abra o arquivo "**aula1-parte5-dicionarios.ipynb**"

Introdução ao Python

- Iremos aprender os seguintes tópicos introdutórios sobre Python:
 - Objetos Pythônicos
 - Strings
 - Listas
 - Sets
 - Dicionários
 - **Estruturas de Controle**
 - Funções

Estruturas de Controle

- Para controlar o fluxo do nosso código, podemos avaliar uma determinada expressão.
 - Um **teste** (expressão que avalia para verdadeiro (True) ou falso (False)).
 - Um **bloco de código** que será executado se o teste for verdadeiro (True).
 - Um **bloco de código** que será executado se o teste for falso (False).



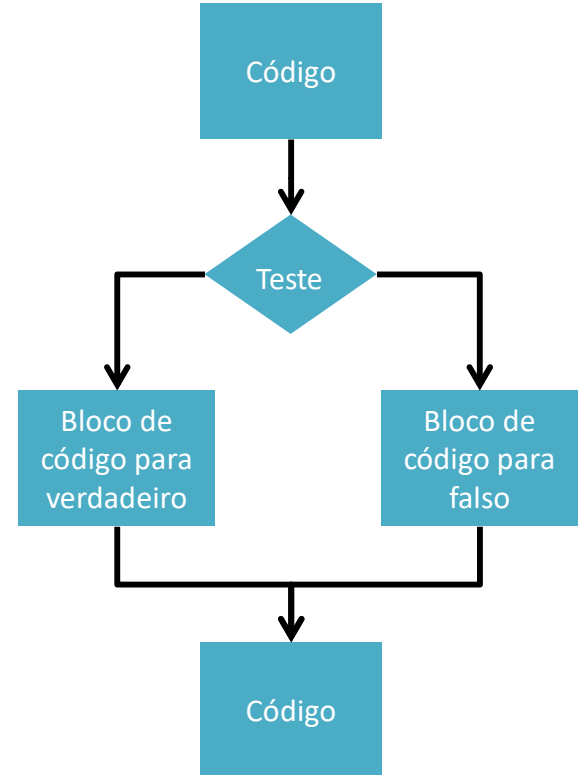
if, elif e else

- Um simples exemplo:

```
>>> n = int(input('Digite um número: '))
```

```
Digite um número: 10
```

```
>>> if n % 2 == 0:
    print('Par')
else:
    print('Impar')
```



if, elif e else

- Mais um exemplo

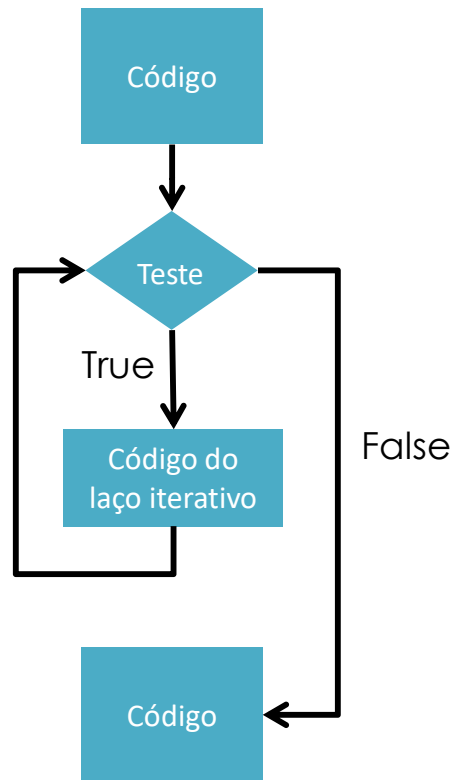
```
>>> nome = 'fia2'
>>> if nome == 'fia':
    idade = 35
    print(idade)
elif nome == 'usp':
    idade = 82
    print(idade)
else:
    print("Não corresponde a nenhum nome")
```

```
if condição:
    # bloco de código
elif condição:
    # outro bloco
else:
    # bloco final
```

 Abra o arquivo "[aula1-parte6-estruturas-controle.ipynb](#)"

Iteração (while)

- É utilizado para execução repetitiva enquanto uma expressão for verdadeira.
 - Inicia com um **teste**
 - Se o teste resultar em verdadeiro (`True`), então o código do laço iterativo será executado **uma única vez** e então o código será redirecionado para que o teste seja refeito.
 - Esse **processo é repetido até que o teste resulte em falso** (`False`), saindo do laço iterativo.



Iteração (while)

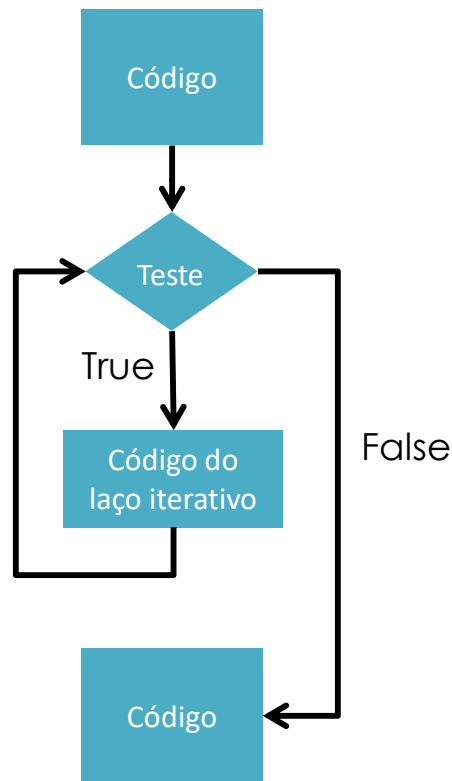
- Um simples exemplo

```
>>> lista = [1, 2, 3]
```

```
>>> n = len(lista) - 1
```

```
>>> while (n != -1):  
    print(lista[n])  
    n = n - 1
```

`</>` Abra o arquivo "**aula1-parte6-estruturas-controle.ipynb**"



Iteração (for)

- Para percorrer um conjunto de valores podemos utilizar o laço iterativo (for)

```
>>> produtos = ['ipad', 'celular', 'notebook', 'tv']
```

```
>>> for item in produtos:
```

```
...     print(item)
```

```
...
```

```
ipad
```

```
celular
```

```
notebook
```

```
tv
```

Iteração (for)

- A função `range(inicio, fim[, passo])` serve para criar listas contendo progressões aritméticas. O início é inclusivo e o fim é exclusivo. O passo não pode ser 0.

```
>>> list(range(5)) # Se o início não for indicado, assume-se 0
[0, 1, 2, 3, 4]
```

```
>>> list(range(1, 5)) # Se o passo não for indicado, assume-se 1
[1, 2, 3, 4]
```

```
>>> list(range(1, 6, 2)) # Utilizando 2 como valor do passo
[1, 3, 5]
```

 Abra o arquivo "[aula1-parte6-estruturas-controle.ipynb](#)"

Pontos de atenção em laços de repetição

- **break**: sai do loop mais próximo que a envolve
- **continue**: pula o início do loop mais próximo que a envolve
- **pass**: não faz absolutamente nada; trata-se de um lugar reservado de instrução, vazio.

```
>>> numeros = [4, 5, 6, 7, 8, -3, 9, -4]
```

```
>>> for num in numeros:
```

```
    if num < 0:
```

```
        print("negativo: %d" % num)
```

```
        break
```

???????

Pontos de atenção em laços de repetição

- Tanto o **if** quanto o **while** utilizam condições lógicas para controle, avaliando-as de maneira booleana.
- Em Python, podemos denotar falso:
 - Pelo booleano **False**,
 - Pelo valor 0 (**zero**)
 - Pela lista, dicionário, ou strings **vazios**, de **tamanho zero**
 - Pelo valor especial **None**, que significa nulo.



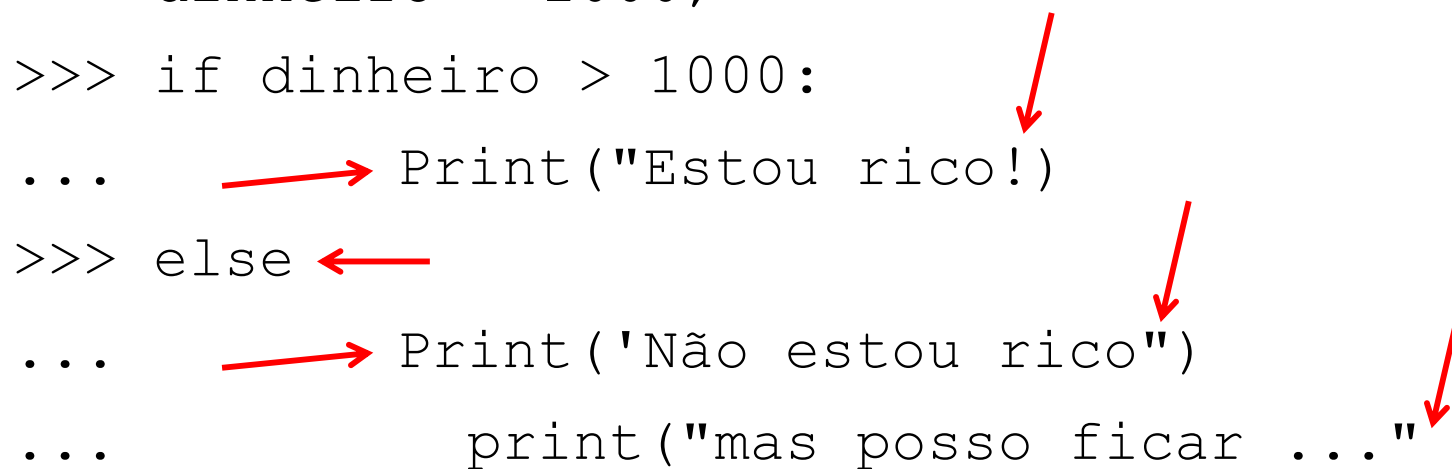
Qualquer outro valor é considerado verdadeiro.

Estruturas de Controle

- Estruturas **condicionais** (`if`, `elif` e `else`) permitem direcionar o fluxo do nosso código para uma determinada parte baseado em uma condição (teste).
- Enquanto que as estruturas de **repetição** (e.g. `while` e `for`) permitem repetir determinadas partes **do código** baseado em uma condição (teste).

Ache os 7 erros no código abaixo.

```
>>> dinheiro = 2000;
>>> if dinheiro > 1000:
...     Print("Estou rico!")
>>> else
...     Print('Não estou rico")
...     print("mas posso ficar ...")
```



Ache os 7 erros no código abaixo.

```
>>> dinheiro = 2000
>>> if dinheiro > 1000:
...     print("Estou rico!")
>>> else:
...     print('Não estou rico')
...     print("mas posso ficar ...")
```

Introdução ao Python

- Iremos aprender os seguintes tópicos introdutórios sobre Python:
 - Objetos Pythônicos
 - Strings
 - Listas
 - Sets
 - Dicionários
 - Estruturas de Controle
 - **Funções**

Funções

- Até agora, vimos diversos tipos de dados, atribuições, comparações e estruturas de controle.
- A ideia é **dividir para conquistar**, onde:
 - Um problema é dividido em diversos subproblemas
 - As soluções dos subproblemas são combinadas na solução do problema maior.
- Esses subproblemas têm o nome de **funções**.

Funções

- Funções possibilitam capturar a computação realizada e tratá-la como primitiva.
- Como exemplo, queremos que a variável **z** seja o máximo de dois números (x e y).
- Um programa simples seria:

```
>>> if x > y:
        z = x
    else:
        z = y
```

Funções

- A ideia da função é encapsular essa computação dentro de um escopo que pode ser tratado como primitiva.
 - Sendo que os detalhes internos estão escondidos dos usuários
 - Para utilizá-las, basta chamar o nome da função e fornecendo os parâmetros necessários.
- Uma função tem **3 partes importantes**:
 - Nome, parâmetros e corpo da função

Funções em Python

```
def <nome> ( <parametros> ) :  
    <corpo da função>
```

- `def` é uma palavra chave e serve para definir uma função.
- `<nome>` é qualquer nome aceito pelo Python.
- `<parametros>` é a quantidade de parâmetros que será passado para a função (pode ser nenhum).
- `<corpo da função>` contém o código da função.

Funções

- Voltando ao nosso exemplo, podemos reescrever:

```
def maximo(x, y):
```

```
    if x > y:
```

```
        z = x
```

```
    else:
```

```
        z = y
```

- Ótimo **temos** uma **função** e podemos reaproveitá-la.
- Porém, para tratá-la como primitiva precisamos utilizar o comando `return` para retornar o valor.

Funções

- Voltando ao nosso exemplo, podemos reescrever:

```
def maximo(x, y):
```

```
    if x > y:
```

```
        return x
```

```
    else:
```

```
        return y
```

- Agora sim! Já podemos reaproveitar nossa função!

- **E como podemos fazer isso?**

Funções

- Voltando ao nosso exemplo:

```
>>> def maximo(x, y):  
    if x > y:  
        return x
```

```
    else:
```

```
        return y
```

Desta forma,
o retorno é
atribuído na
variável z

- Agora podemos chamar a função:

```
>>> z = maximo(3, 4)
```

```
>>> print(z)
```

```
4
```

2
Todas as expressões são avaliadas, e caso não se encontre correspondência, é retornado o valor `None`.

3
Ou até que encontre a palavra especial **return**

1
Quando chamamos a função `maximo(3, 4)` estamos definindo que `x = 3` e `y = 4`.

Funções

- Já entendemos o que é e como criar funções.
- Vamos testar e codificar um pouco mais 😊



Abra o arquivo **"aula1-parte7-funções.ipynb"**

Conteúdo da Aula

- Objetivo
- Introdução ao Python
- **Exercícios**
- Referências Bibliográficas

Exercícios

 Abra o arquivo **"aula1-parte8-exercicios.ipynb"**

Conteúdo da Aula

- Objetivo
- Introdução ao Python
- Exercícios
- **Referências Bibliográficas**

Referências Bibliográficas

- **Use a Cabeça! Python** – Paul Barry - Rio de Janeiro, RJ: Alta Books, 2012.
- **Use a Cabeça! Programação** – Paul Barry & David Griffiths – Rio de Janeiro RJ: Alta Books, 2010.
- **Aprendendo Python: Programação orientada a objetos** – Mark Lutz & David Ascher – Porto Alegre: Bookman, 2007

Referências Bibliográficas

- **Python for kids – A playful Introduction to programming** – Jason R. Briggs – San Francisco – CA: No Starch Press, 2013.
- **Python for Data Analysis** – Wes McKinney – USA: O'Reilly, 2013.
- **Python Cookbook** – David Beazley & Brian K. Jones – O'Reilly, 3th Edition, 2013.
- As referências de links utilizados podem ser visualizados em <http://urls.dinomagri.com/refs>