

BIG DATA



Introdução ao Big Data

Tema da Aula: **Introdução ao Python**

Prof.: **Dino Magri**

Coordenação:

Prof. Dr. Adolpho Walter
Pimazzi Canton

Profa. Dra. Alessandra de
Ávila Montini

Coordenação:

Prof. Dr. Adolpho Walter
Pimazzí 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>



Dino Magri

Head of Data Science at Beholder



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>

Conteúdo da Aula

- Objetivo
- Introdução ao Python
- Referências Bibliográficas

Conteúdo da Aula

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

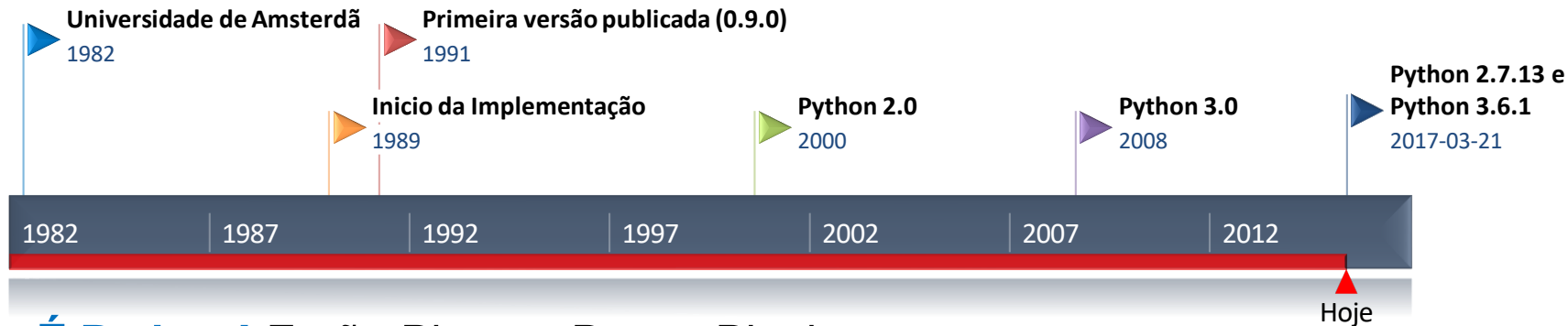
Objetivo

- O objetivo dessa aula é **introduzir conceitos básicos sobre** a linguagem de programação **Python** para Big Data.

Conteúdo da Aula

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

- **Guido van Rossum**

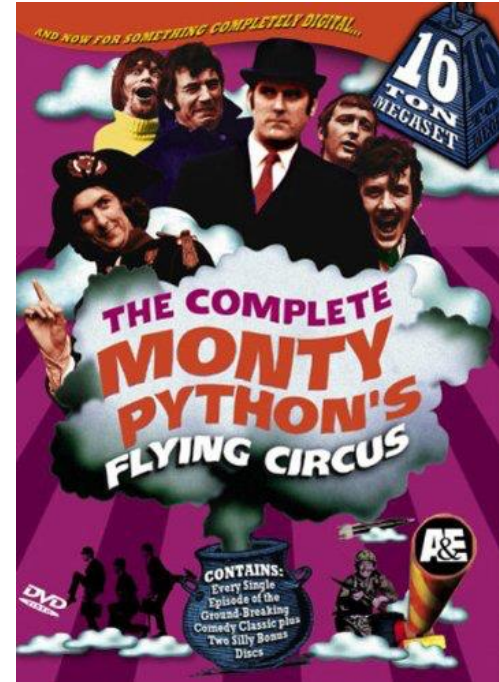


- **É Python!** E não Phytton, Pyton, Phython
- Licença de código aberto, compatível com GPL
- Interpretada - Ambiente Interativo
- Linguagem de altíssimo Nível (VHLL)



Por que utilizar Python?

- **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.

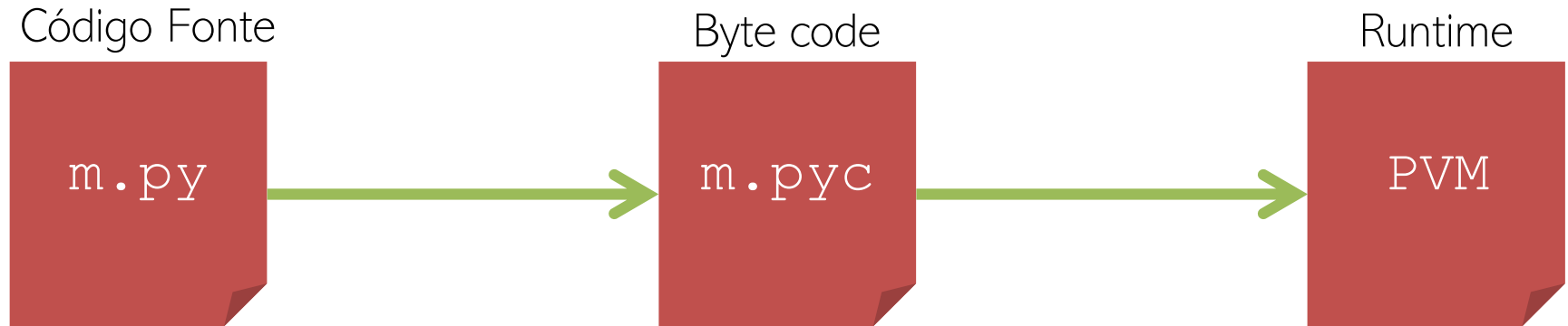


Fonte: <https://docs.python.org/2/faq/general.html#why-is-it-called-python>

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

Por que utilizar Python?

- **E o que quer dizer Interpretada?**
 - Uma linguagem interpretada necessita de interpretador interno de código, o qual irá traduzir a linguagem de alto nível escrita por nós para a linguagem de máquina entendida pelo computador.



Por que utilizar Python?

Por que Python para Big Data?

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

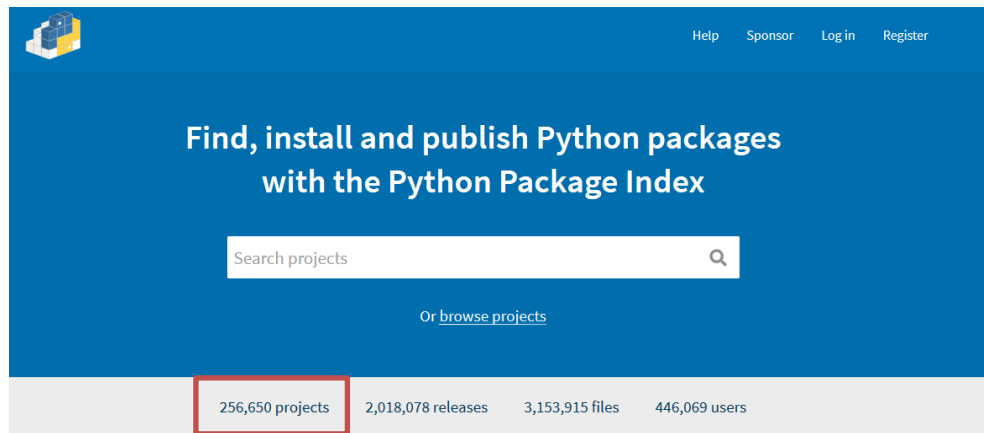


```
public class Hello {
    public static void main(String args[]) {
        java.util.Scanner s = new java.util.Scanner(System.in);
        System.out.print("Digite seu nome: ");
        String nome = s.nextLine();
        System.out.println("Olá " + nome);
    }
}
```

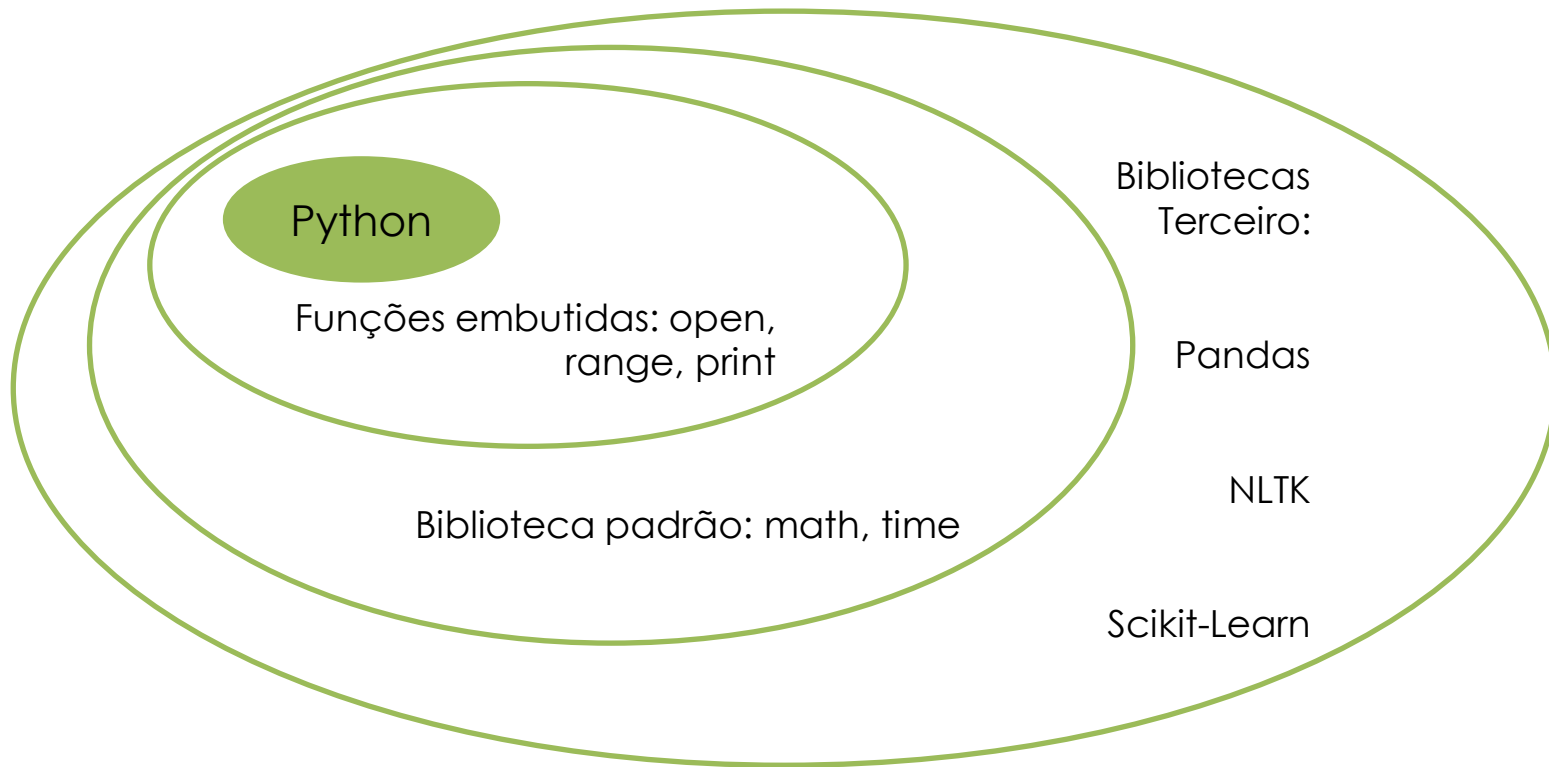

Por que utilizar Python?

- **Baterias Incluídas**

- Python vem com uma grande coleção de funcionalidades (*standard library*)
- <https://docs.python.org/3/library/>
- Existem mais de **200k** projetos disponíveis que podemos utilizar.
- Esses projetos são as bibliotecas de terceiros
- <https://pypi.org/>



Por que utilizar Python?



Fonte: <https://docs.python.org/pt-br/3/library/>

Por que utilizar Python?

- Roda em diversas plataformas



maemo.ORG



1 → ✕
one laptop per child



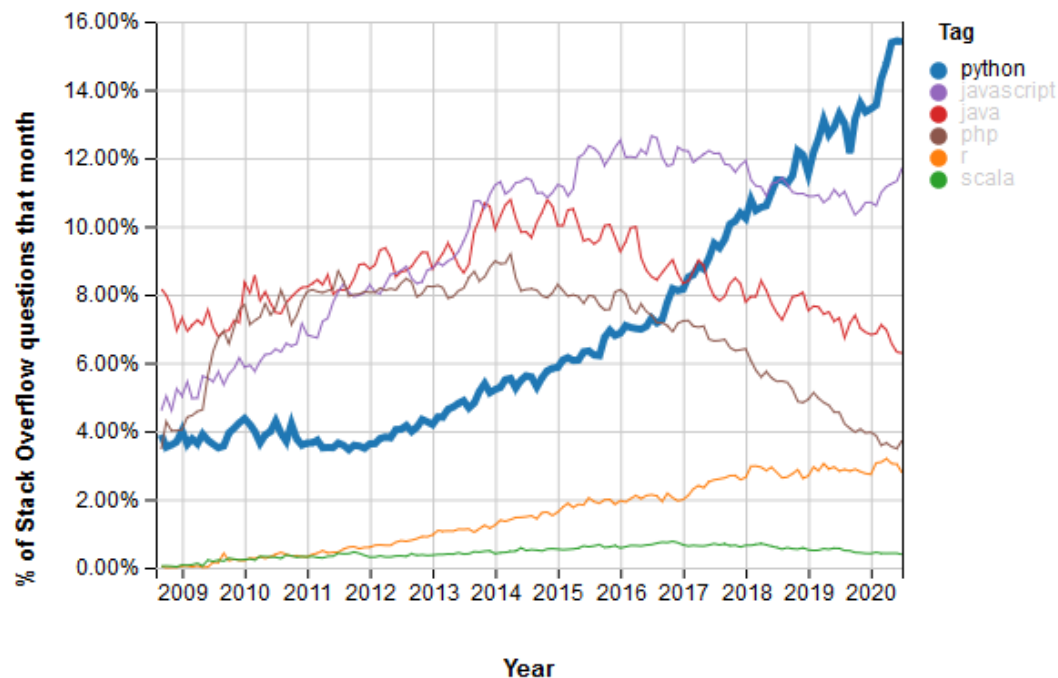
Por que utilizar Python?

- **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 busca por questões sobre determinada linguagem de programação:

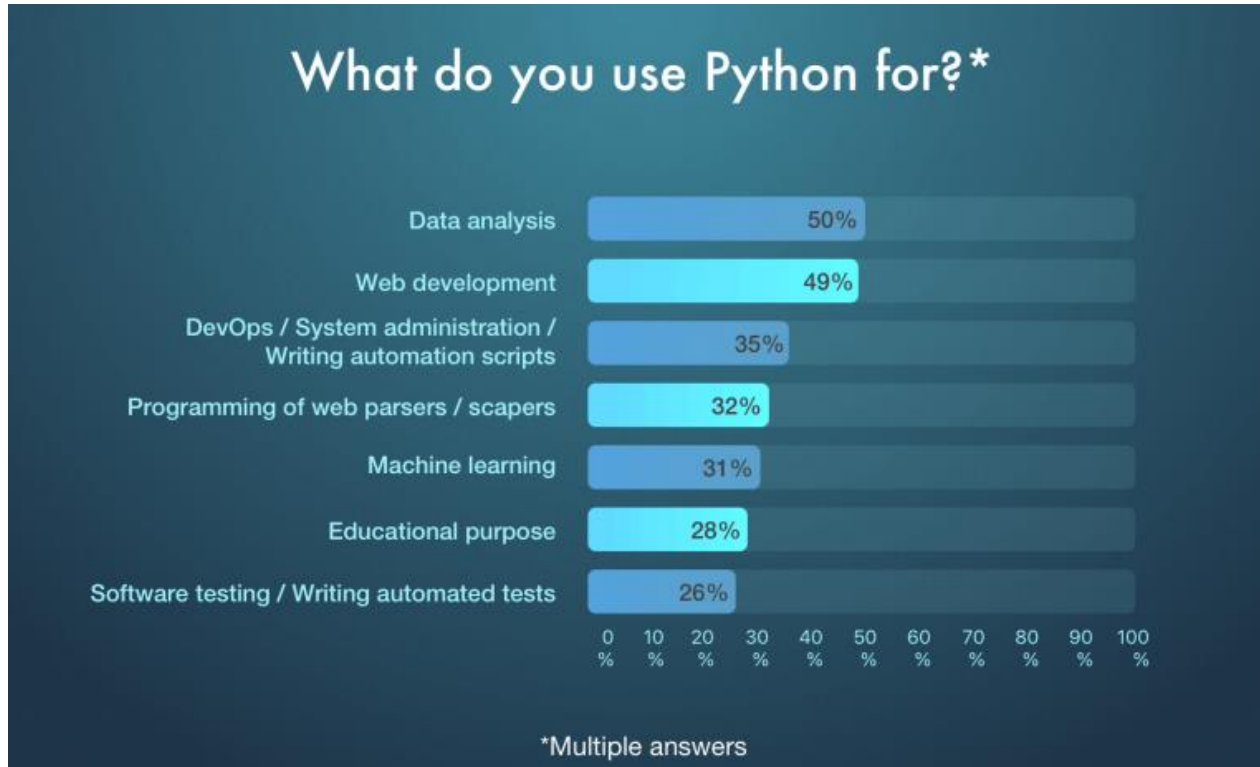


Fonte: <https://bit.ly/linguagens-programacao>

Cases de Sucesso



Cases de Sucesso



Fonte: <https://www.botreetechnologies.com/blog/top-10-python-use-cases-and-applications>

INSTALAÇÃO

Windows 10

Instalação

- 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.
- Acesse: <https://www.python.org>

Como conversar com Python?

- 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?

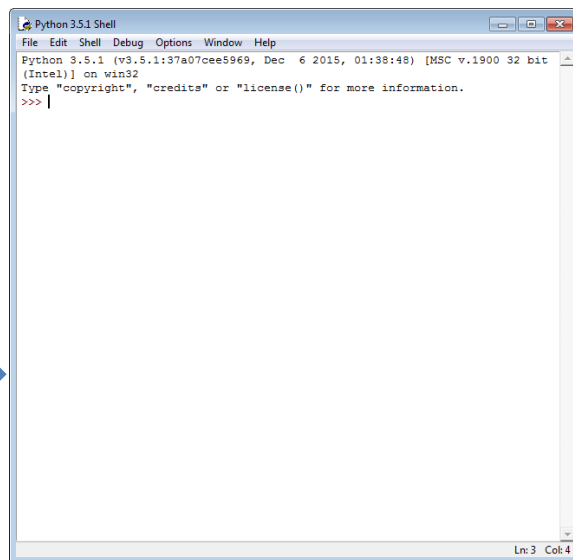
- E onde posso escrever essa linguagem?
 - Temos que ter uma **IDE!!!**
- E o que é uma IDE?
 - É um ambiente integrado de desenvolvimento (*Integrated Development Environment*)
- Ou seja, **é um programa onde iremos adicionar código** para que o Python consiga interpretar e executar o que está sendo 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/>
- Para a nossa primeira conversa com Python vamos utilizar o **IDLE**
 - Para abrir, clique em Iniciar ou aperte o botão do Windows do teclado → no campo de busca → digite IDLE!

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?

- **Primeiro programa (Modo Interativo)**
 - Vamos iniciar o IDLE e imprimir o texto "Olá Mundo" utilizando a função `print()` :

```
>>> print("Olá Mundo")
```

```
Olá Mundo
```

```
>>>
```

Como conversar com Python?

- **Primeiro programa (Modo Editor)**

- Ainda no IDLE, acesse o menu **File** → **New File** (uma nova janela do IDLE será aberta)
- Clique em **File** → **Save** (selecione uma pasta para salvar o arquivo)
- Defina um nome para o seu programa, por exemplo, **ola-mundo.py**
- Adicione o código abaixo e execute (F5 ou Run → Run Module)

```
print("Olá Mundo")
```

Como conversar com Python?

- **Primeiro programa (Modo Editor)**
 - Algumas considerações importantes
 - É necessário adicionar a extensão `".py"` quando salvar arquivo
 - Para rodar o programa utilize o menu **Run** → **Run Module**
 - Não existe uma opção para limpar a tela
 - » Aperte e segure a tecla **Enter**
 - Para copiar uma linha no IDLE, clique na linha que deseja copiar e aperte a tecla Enter!

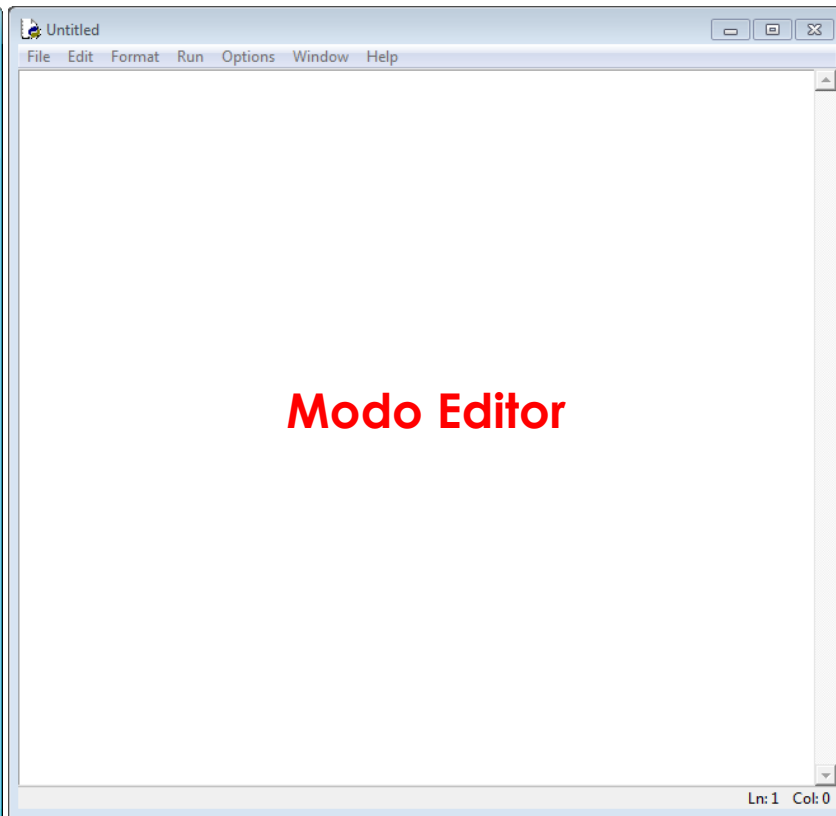
Como conversar com Python?



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.", and the interactive prompt ">>> |". The status bar at the bottom right indicates "Ln: 3 Col: 4".

```
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.
>>> |
```

Modo Interativo





**KEEP CALM AND CODE IN
PYTHON**

Olá Mundo

- Abra o IDLE no modo interativo e teste os seguintes casos:

```
>>> Print("Olá Mundo")
```

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

```
  File "<pyshell#49>", line 1, in <module>
```

```
    Print("Olá Mundo")
```

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

```
>>> print (Olá Mundo)
```

```
      ^
```

```
SyntaxError: invalid syntax
```

Olá Mundo

- Abra o IDLE no modo interativo e teste os seguintes casos:

```
>>> print ("Olá Mundo")
```

```
File "<pyshell#51>", line 1
```

```
    print ("Olá Mundo")
```

```
    ^
```

```
IndentationError: unexpected indent
```

Como conversar com Python?

- É importante notar algumas regras básicas de sintaxe:
 - Não tem necessidade de colocar ponto e virgula no final dos comandos!
 - Sem delimitadores de código – { }
 - **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**

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 conversar com o Python!

Conteúdo da Aula

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

Conteúdo da Aula

- Introdução ao Python
 - **Primeiros Passos**
 - O poder do Python
 - Strings e Listas
 - Tuplas e Dicionários
 - Estruturas de Controle e Repetição
 - Resumo

Objetos Pythonicos

- 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 Pythonicos

- **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.

***Existem outros tipos de objeto não-escalar
que serão abordados durante o curso.***

- `'abc'`
- `"abc"`
- `'123'` – essa é uma string de caracteres, não os números.

Identificadores e atribuições

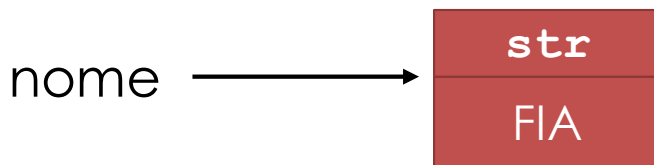
- Um comando do Python muito importante é a atribuição:

```
>>> 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



**KEEP CALM AND CODE IN
PYTHON**

Identificadores e atribuições

- Como Python interpreta os tipos?

```
>>> a = 1           # int
>>> b = 1.0         # float
>>> c = True        # bool
>>> d = None        # None
```

```
>>> print(type(a))
<class 'int'>
```

```
>>> print(type(b))
<class 'float'>
```

```
>>> print(type(c))
<class 'bool'>
```

```
>>> print(type(d))
<class 'NoneType'>
```

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
```

Exemplos

```
>>> nome == "Fia"
```

```
False
```

```
>>> (nome == "Fia" or nome == "FIA")
```

```
True
```

```
>>> (nome == "Fia" or nome == "FIA") and idade == 35
```

```
True
```


Conteúdo da Aula

- Introdução ao Python
 - Primeiros Passos
 - **O poder do Python**
 - Strings e Listas
 - Tuplas e Dicionários
 - Estruturas de Controle e Repetição
 - Resumo

Poderes Ilimitados !!!!

- Conforme vimos anteriormente, Python tem uma vasta lista de módulos da **biblioteca padrão** e também de **bibliotecas de terceiros!**
- Um módulo é composto por códigos Python em um arquivo com a extensão `.py`
- Exemplificando, o arquivo `minhas_funcoes.py` contém diversas funções matemáticas.

Poderes Ilimitados !!!!

`minhas_funcoes.py`

```
def soma(n,m):  
    return n + m  
  
def sub(n,m):  
    return n - m  
  
def mult(n,m):  
    return n * m  
  
def nota(n, m):  
    n1 = 0.4 * n  
    n2 = 0.6 * m  
    total = soma(n1, n2)  
    return total
```

- Suponha agora que estou criando um novo projeto e ele precisará utilizar essas funções, como podemos fazer?
- **Basta utilizar o comando `import`**

`novo_projeto.py`

```
import minhas_funcoes as mf  
  
N1 = 10  
N2 = 8  
print(f'Nota Final: {nota(N1, N2)}')
```

Atenção, ambos os arquivos devem estar na mesma pasta para funcionar corretamente.

Poderes Ilimitados !!!!

`minhas_funcoes.py`

```
def soma(n,m):  
    return n + m  
  
def sub(n,m):  
    return n - m  
  
def mult(n,m):  
    return n * m  
  
def nota(n, m):  
    n1 = 0.4 * n  
    n2 = 0.6 * m  
    total = soma(n1, n2)  
    return total
```

- Suponha agora que estou criando um novo projeto e ele precisará utilizar essas funções, como podemos fazer?
- **Basta utilizar o comando `import`**

`novo_projeto.py`

```
import minhas_funcoes as mf  
  
N1 = 10  
N2 = 8  
print(f'Nota Final: {mf.nota(N1, N2)}')
```

Atenção, ambos os arquivos devem estar na mesma pasta para funcionar corretamente.

Poderes 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)
3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:57:54) [MSC v.1924 64 bit
(AMD64)]
```

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

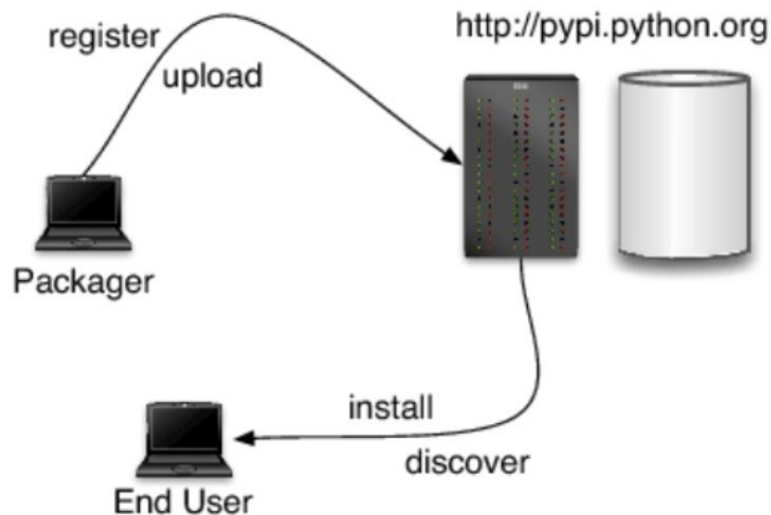
```
>>> import datetime
>>> print(datetime.date.today())
2020-08-16
```

Como faço para utilizar outras bibliotecas?

- Para realizar a instalação de qualquer biblioteca, iremos utilizar um gerenciador de pacotes já disponível quando instalamos o Python.
- Esse gerenciador permite pesquisar, instalar e remover as bibliotecas (ou pacotes).
- O gerenciador chama-se **PIP – Python Package Index**
- Outras bibliotecas podem ser visualizadas em <https://pypi.org/>

Como faço para utilizar outras bibliotecas?

- Como funciona?



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

Como faço para utilizar outras bibliotecas?

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

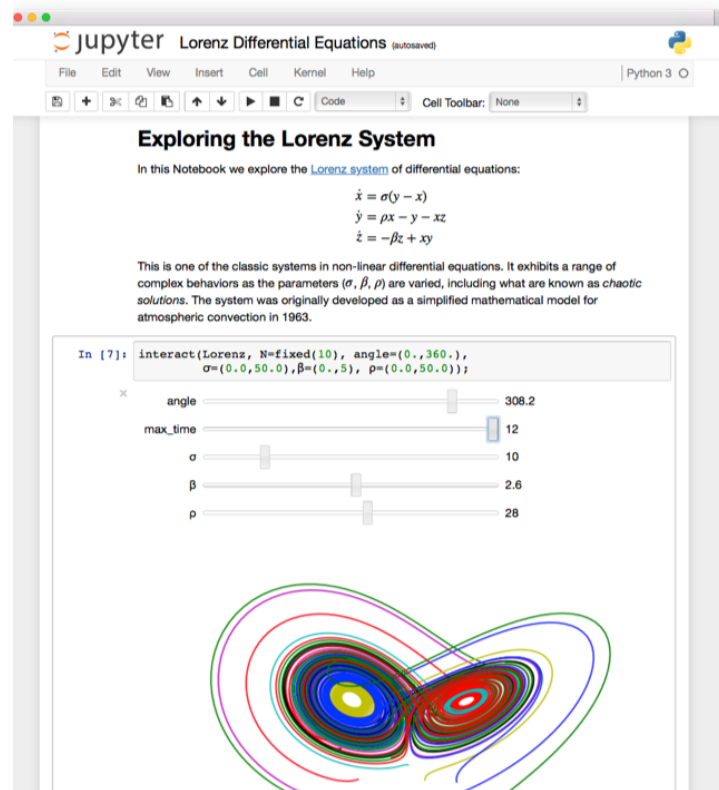
```
$ pip search [nome do pacote]
```
 - Para instalar, digite (utilize o parâmetro `-U` para atualizar um pacote caso seja necessário)

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

```
$ pip uninstall [nome do pacote]
```

Como faço para utilizar outras bibliotecas?

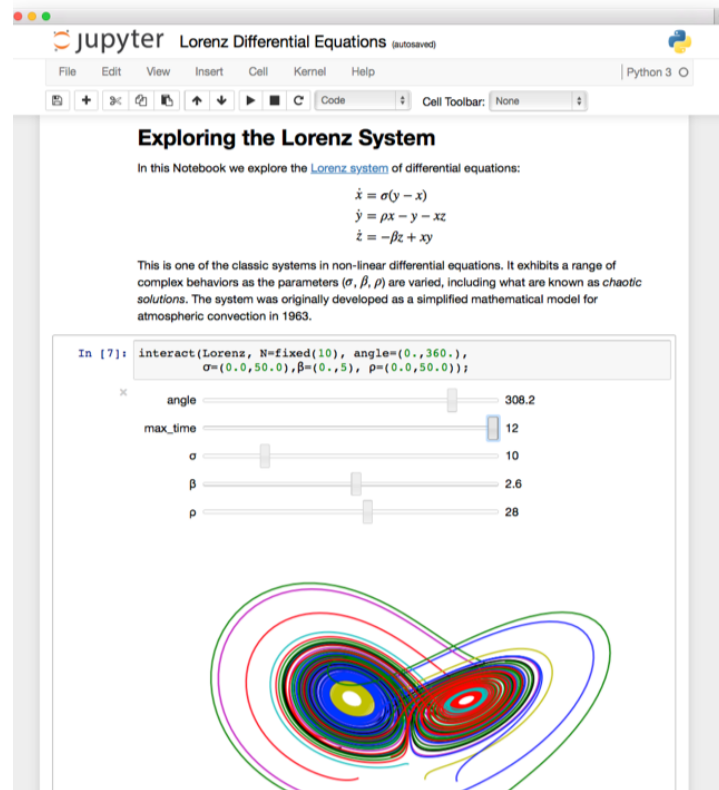
- Para testar o uso do `pip` iremos instalar um pacote chamado **Jupyter**.
- Jupyter é uma aplicação web que permite criar e compartilhar **notebooks** que contém código, equações, visualizações e textos explicativos.



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
```

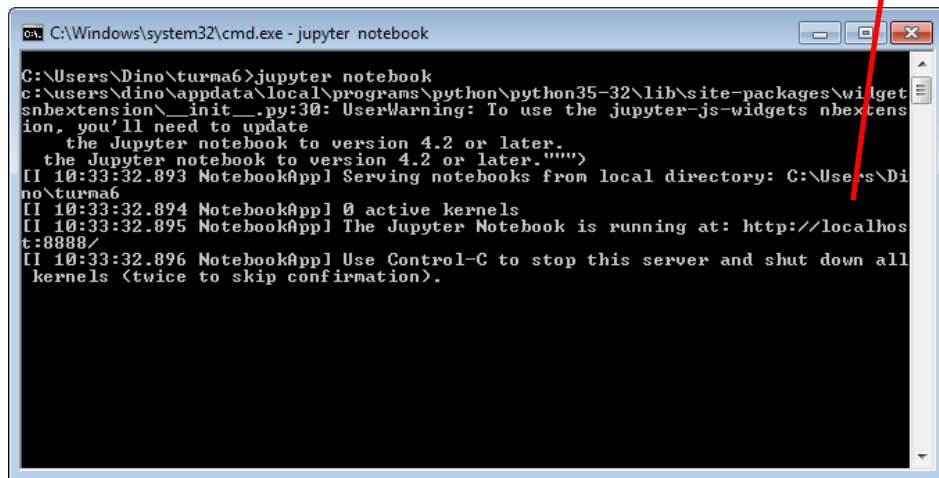
- Note que durante a instalação, o pip irá baixar todos os pacotes necessários para instalar corretamente o pacote Jupyter!

Como faço para utilizar outras bibliotecas?

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

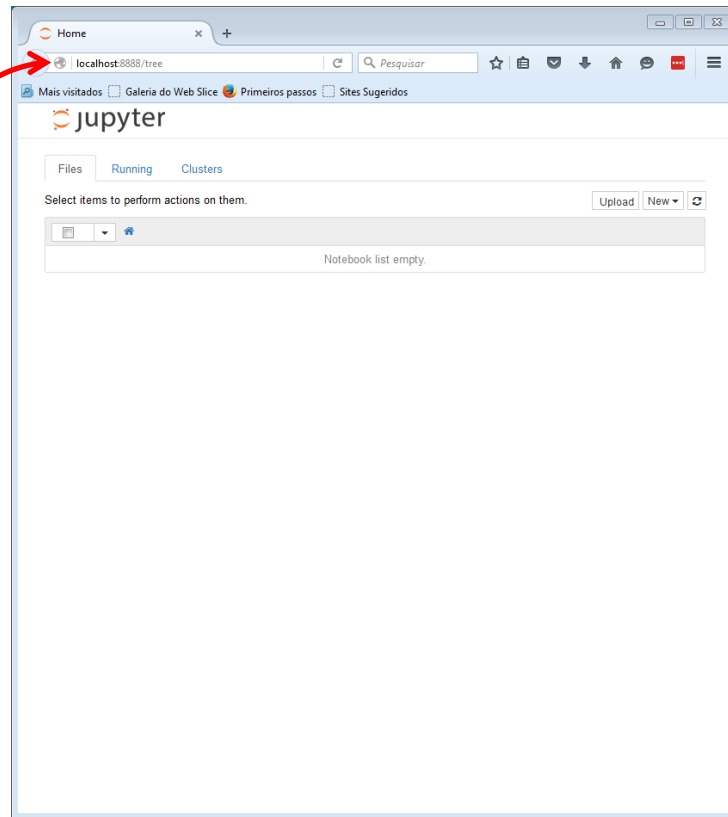
```
$ cd Desktop
```

```
$ jupyter notebook
```



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

C:\Users\Dino\turma6>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\turma6
[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).
```



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 foram salvos no Desktop, já será possível acessar os notebooks de todas as aulas.



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



**KEEP CALM AND CODE IN
PYTHON**

Conteúdo da Aula

- Introdução ao Python
 - Primeiros Passos
 - O poder do Python
 - **Strings e Listas**
 - Tuplas e Dicionários
 - Estruturas de Controle e Repetição
 - Resumo

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

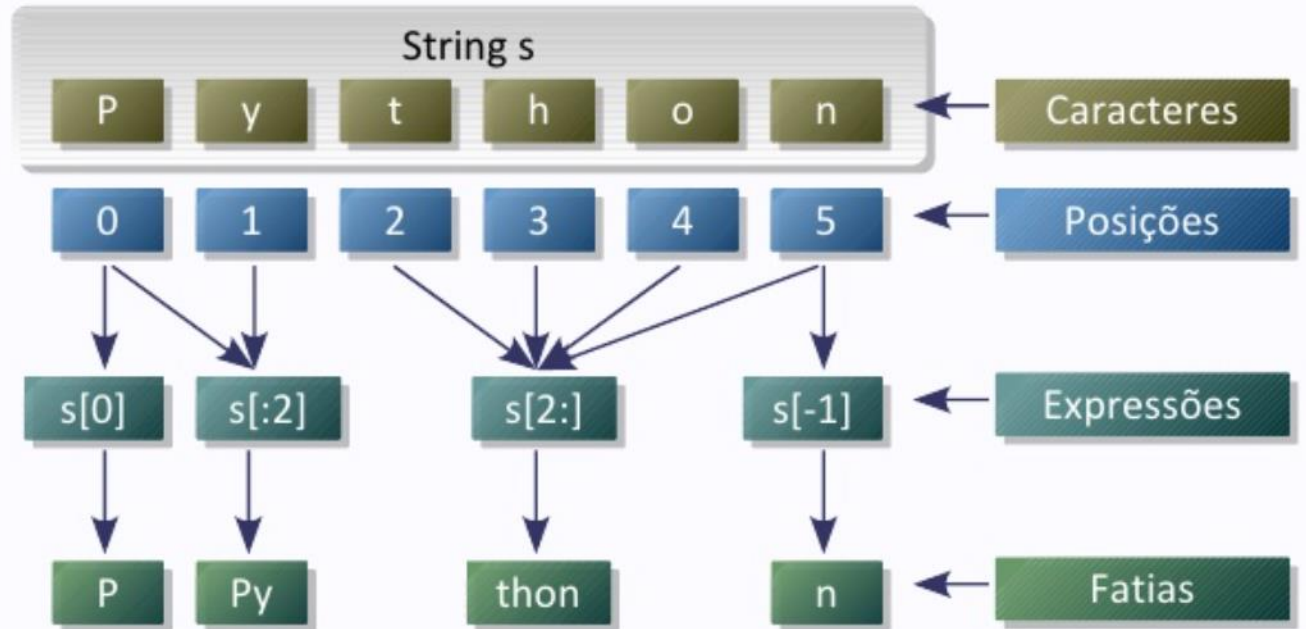
- Podemos substituir símbolos em strings
- Podemos concatenar strings
- Podemos multiplicar strings
- Podemos indexar e fatiar strings
- E muito mais ...

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>



**KEEP CALM AND CODE IN
PYTHON**

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 "**aula1-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 outros 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 ...

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:

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

<https://docs.python.org/3/tutorial/datastructures.html#more-on-lists>



**KEEP CALM AND CODE IN
PYTHON**

Conteúdo da Aula

- Introdução ao Python
 - Primeiros Passos
 - O poder do Python
 - Strings e Listas
 - **Tuplas e Dicionários**
 - Estruturas de Controle e Repetição
 - Resumo

Tuplas

- **Tuplas** constroem grupos simples de objetos.
- Elas trabalham exatamente como listas, exceto que tuplas não podem ser modificadas (são **imutáveis**) e são normalmente escritas como uma série de itens entre parênteses, não entre colchetes.
- Os itens das tuplas são acessados via índice.
- **Não suporta operações de alteração.**

Tuplas

- Para criar uma tupla vazia:

```
>>> t1 = ()
```


- Para criar uma tupla com 1 item

```
>>> t2 = (1,)
```

```
>>> tuple("abcd")  
('a', 'b', 'd', 'c')
```

- Para criar uma tupla com 6 itens

```
>>> numeros = (1, 2, 3, 4, 5, 6)
```



O que podemos fazer com tuplas?

- Podemos concatenar tuplas
- Podemos multiplicar seus valores
- Podemos indexar e fatiar tuplas
- Outras ações (métodos) podem ser visualizados em:
<https://docs.python.org/3/tutorial/datastructures.html#tuples-and-sequences>

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



**KEEP CALM AND CODE IN
PYTHON**

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 ...
- <https://docs.python.org/3/tutorial/datastructures.html#dictionaries>

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



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



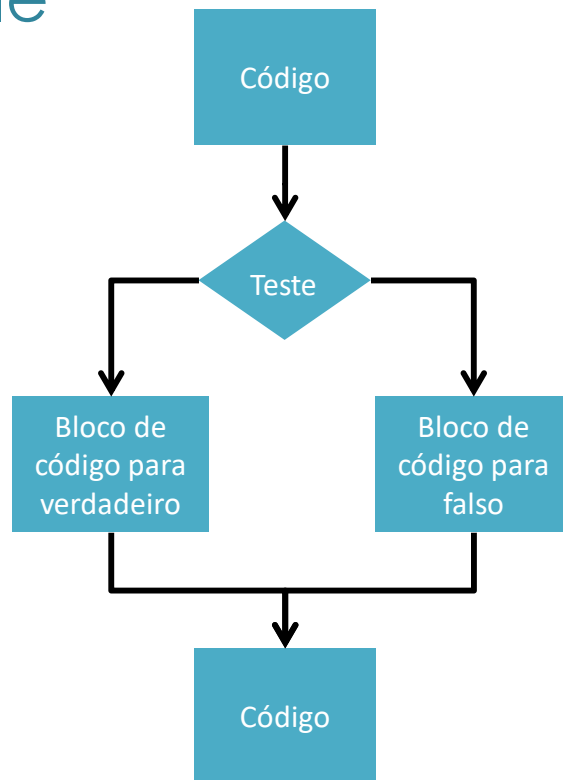
**KEEP CALM AND CODE IN
PYTHON**

Conteúdo da Aula

- Introdução ao Python
 - Primeiros Passos
 - O poder do Python
 - Strings e Listas
 - Tuplas e Dicionários
 - **Estruturas de Controle e Repetição**
 - Resumo

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).



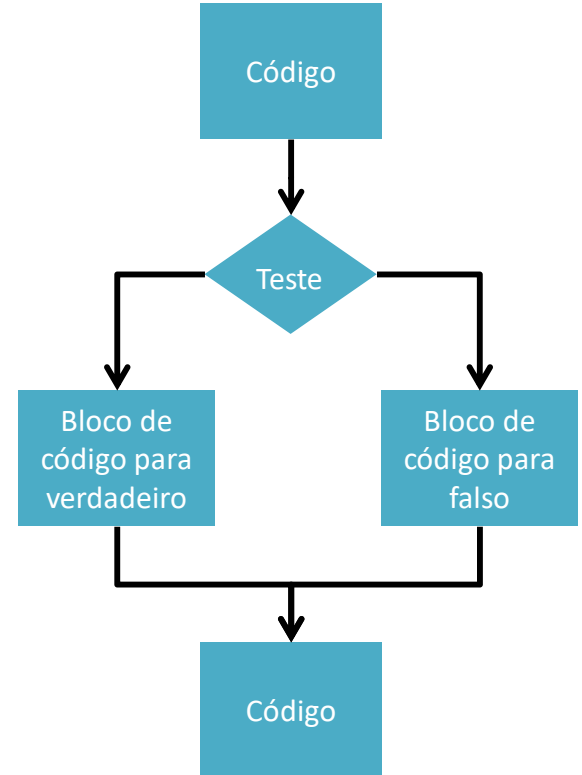
if, elif e else

- 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
```



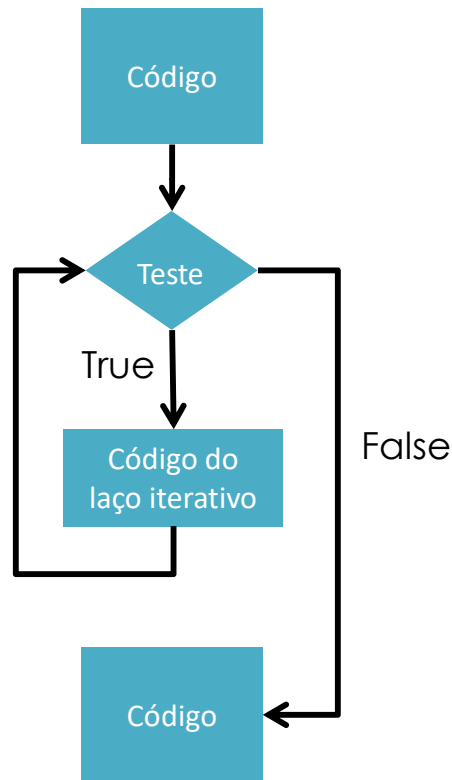
Abra o arquivo **"aula1-parte6-estruturas-controle.ipynb"**



**KEEP CALM AND CODE IN
PYTHON**

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)

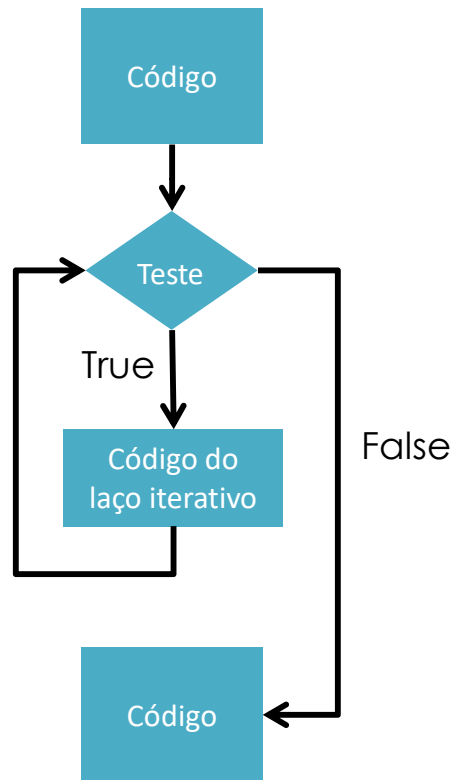
- Um simples exemplo

```
>>> 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"](#)





**KEEP CALM AND CODE IN
PYTHON**

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](#)"



**KEEP CALM AND CODE IN
PYTHON**

Pontos de atenção em laços de repetição

- **break**: sai do loop mais próximo que a envolve

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

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

```
    if num < 0:
```

```
        print(f"negativo: {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`) permite 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).

Conteúdo da Aula

- Introdução ao Python
 - Primeiros Passos
 - O poder do Python
 - Strings e Listas
 - Tuplas e Dicionários
 - Estruturas de Controle e Repetição
 - **Resumo**

Resumo

```
import sys

print('Bem vindo')

num = input ('Escolha um número: ')

if num == 10:

    print('Uhuu!!!')

else:

    print('Errou :(')

print('Obrigado!')
```

Resumo

Módulos

Funções

Strings

Variáveis

Diretivas

Endentação

```
import sys
```

```
print('Bem vindo')
```

```
num = input('Escolha um número: ')
```

```
if num == 10:
```

```
    print('Uhhu!!!')
```

```
else:
```

```
    print('Errou :(')
```

```
print('Obrigado!')
```

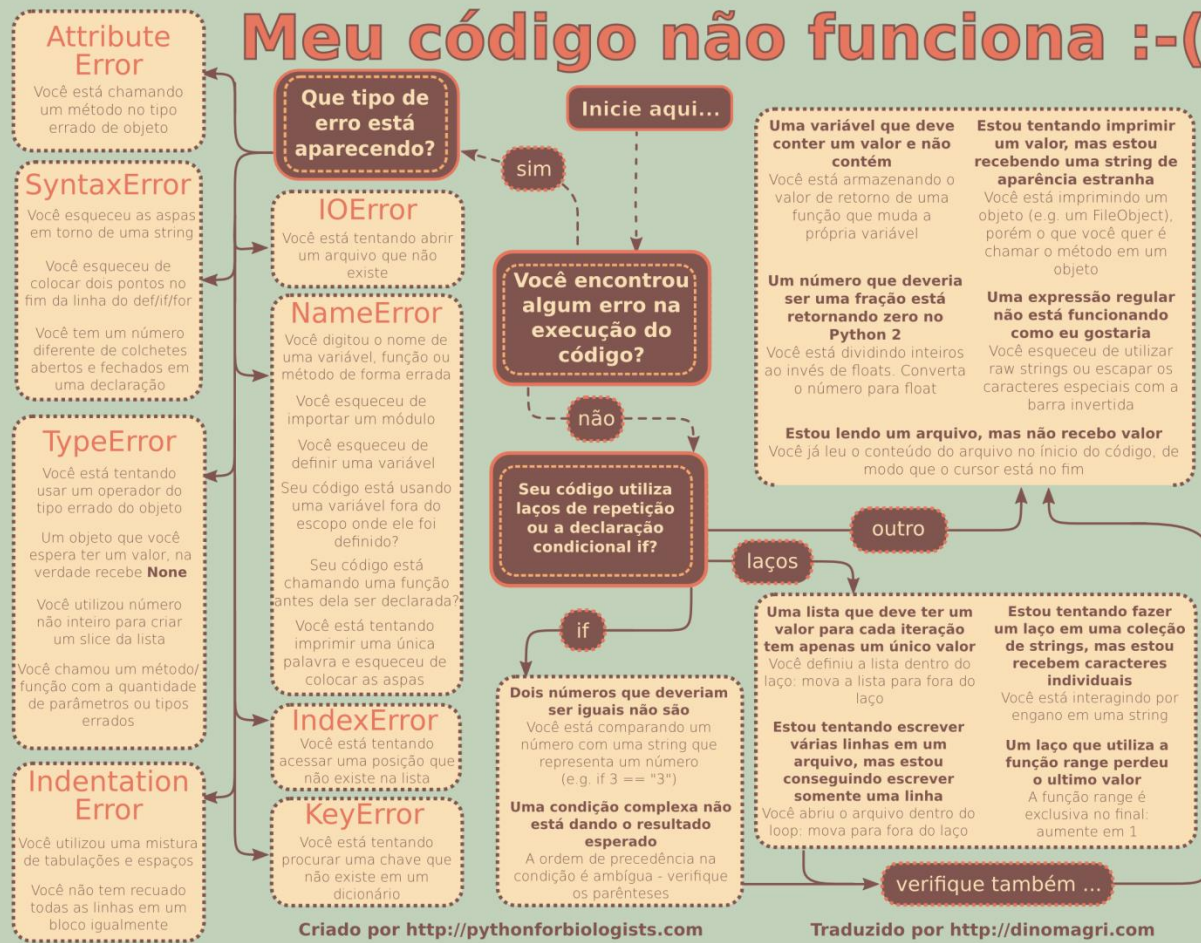
= atribuição

== comparação

Resumo

- **Módulos** devem ser importados antes de serem utilizados
- **Strings** devem estar dentro de aspas
- **Variáveis** controlam dados na memória e possuem tipos
- **Diretivas** são comandos da linguagem
- **Endentação** separa blocos de comandos
- **Funções** definem ações a serem realizadas
- **=** atribuição (a=10)
- **==** comparação (a==10?)
- **:** abrem blocos de instruções

Meu código não funciona :-)



Criado por <http://pythonforbiologists.com>

Traduzido por <http://dinomagri.com>

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>