



Análise de dados com **Python 3**

Nível introdutório

Fabio A. Fajardo Molinares

Laboratório de Estatística e Computação Natural – LECON
DEST/UFES.

July 12, 2021



Conhecendo o Python



Vantagens do Python

- ▶ Python é uma linguagem de programação de alto nível.



Vantagens do Python

- ▶ Python é uma linguagem de programação de alto nível.
- ▶ **Portável:** o que significa que pode rodar em diferentes tipos de computador, com pouca ou nenhuma modificação.

Linguagem de programação de alto nível é uma linguagem com um nível de abstração relativamente elevado, longe do código de máquina e mais próximo à linguagem humana.

Outras linguagens de alto nível de que você já pode ter ouvido falar são C/C++, R, Basic/VisualBasic, PHP, Java, JavaScript entre outros.



Vantagens do Python

- ▶ Python é uma linguagem de programação de alto nível.
- ▶ **Portável:** o que significa que pode rodar em diferentes tipos de computador, com pouca ou nenhuma modificação.

Linguagem de programação de alto nível é uma linguagem com um nível de abstração relativamente elevado, longe do código de máquina e mais próximo à linguagem humana.

Outras linguagens de alto nível de que você já pode ter ouvido falar são C/C++, R, Basic/VisualBasic, PHP, Java, JavaScript entre outros.

- ▶ **Linguagem Interpretada:** Python é considerada uma linguagem interpretada, pois os programas em Python são executados por um interpretador.



Interpretadores vs Compiladores

Dois tipos de programas processam linguagens de alto nível, traduzindo-as em linguagens de baixo nível: **interpretadores** e **compiladores**.

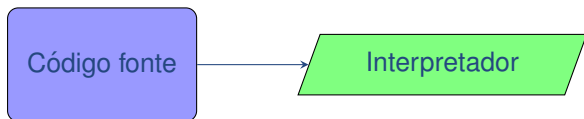
Código fonte

O interpretador lê um programa escrito em linguagem de alto nível e o executa, ou seja, faz o que o programa diz.



Interpretadores vs Compiladores

Dois tipos de programas processam linguagens de alto nível, traduzindo-as em linguagens de baixo nível: **interpretadores** e **compiladores**.



O interpretador lê um programa escrito em linguagem de alto nível e o executa, ou seja, faz o que o programa diz.



Interpretadores vs Compiladores

Dois tipos de programas processam linguagens de alto nível, traduzindo-as em linguagens de baixo nível: **interpretadores** e **compiladores**.



O interpretador lê um programa escrito em linguagem de alto nível e o executa, ou seja, faz o que o programa diz.

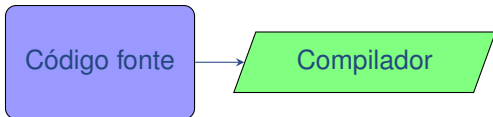


O compilador lê o programa e o traduz completamente antes que o programa comece a rodar.

Código fonte



O compilador lê o programa e o traduz completamente antes que o programa comece a rodar.



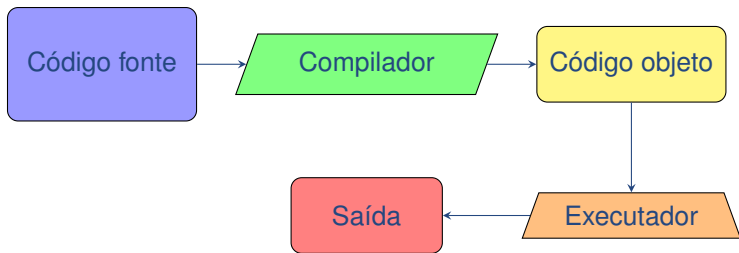


O compilador lê o programa e o traduz completamente antes que o programa comece a rodar.





O compilador lê o programa e o traduz completamente antes que o programa comece a rodar.



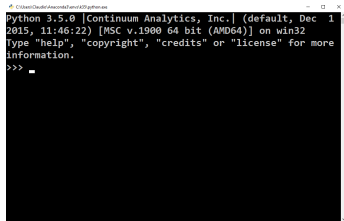
Neste caso, o programa escrito em linguagem de alto nível é chamado de **código fonte**, e o programa traduzido é chamado de **código objeto** ou **executável**. Uma vez que um programa é compilado, você pode executá-lo repetidamente, sem que precise de nova tradução.



Modos de usar o interpretador

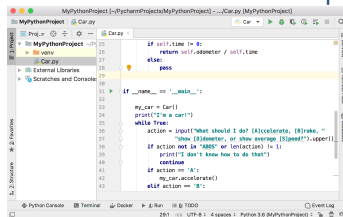
Existem duas maneiras de usar o **interpretador**: modo de linha de comando e no modo de script.

No modo de **linha de comando**, você digita programas em Python e o interpretador mostra o resultado.



```
Python 3.5.0 [Continuum Analytics, Inc.] (default, Dec 1
2015, 11:46:22) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more
information.
>>>
```

No modo **script** Você escreve um programa em um arquivo e usa o interpretador para executar o conteúdo desse arquivo.



```
MyPythonProject - [PythonProjects/MyPythonProject] - Car.py [MyPythonProject]
25
26 if self.time != 0:
27     return self.odometer / self.time
28 else:
29     pass
30
31 if __name__ == '__main__':
32
33     my_car = Car()
34     print("I'm a car!")
35     while True:
36         action = input("What should I do? (A)ccelerate, (B)rake, "
37                        "show (S)peedometer, or show average (S)peed?").upper()
38         if action not in "ABSS" or len(action) != 1:
39             print("I don't know how to do that")
40             continue
41         if action == 'A':
42             my_car.accelerate()
43         elif action == 'B':
```



Vantagens do Python

- ▶ **multiplataforma:** Roda em ambientes Linux, Windows, MacOS, smartphones, celulares, entre outros sistemas.



Vantagens do Python

- ▶ **multiplataforma:** Roda em ambientes Linux, Windows, MacOS, smartphones, celulares, entre outros sistemas.
- ▶ **Velocidade:** Python é considerada uma das linguagens de programação mais velozes atualmente.



Vantagens do Python

- ▶ **multiplataforma:** Roda em ambientes Linux, Windows, MacOS, smartphones, celulares, entre outros sistemas.
- ▶ **Velocidade:** Python é considerada uma das linguagens de programação mais velozes atualmente.
- ▶ **Análise de dados:** Python é usada principalmente quando a sua análise de dados precisa ser integrada com aplicativos web ou se o código estatístico precisa ser integrado em um servidor em ambiente de produção, que vai servir muitos usuários.



Python vs R



► Data Science Wars





Round 1: Facilidade de desenvolvimento

- O **Python** se presta mais facilmente a desenvolvedores que têm experiência com outras linguagens de programação, pois a sintaxe é mais familiar do que **R**, ao mesmo tempo em que é mais próxima do inglês regular, o que facilita a leitura e a depuração.





Round 1: Facilidade de desenvolvimento

- ▶ O **Python** se presta mais facilmente a desenvolvedores que têm experiência com outras linguagens de programação, pois a sintaxe é mais familiar do que  **R**, ao mesmo tempo em que é mais próxima do inglês regular, o que facilita a leitura e a depuração.
- ▶  **R** é muito popular entre pesquisadores e analistas de dados. Difícilmente é usado para programação e desenvolvimento de software. Nesse caso, o **Python** oferece maior flexibilidade, legibilidade e muita semelhança com o tipo de programação que você já conhece e adora.



Round 1: Facilidade de desenvolvimento

- ▶ O **Python** se presta mais facilmente a desenvolvedores que têm experiência com outras linguagens de programação, pois a sintaxe é mais familiar do que , ao mesmo tempo em que é mais próxima do inglês regular, o que facilita a leitura e a depuração.
- ▶  é muito popular entre pesquisadores e analistas de dados. Dificilmente é usado para programação e desenvolvimento de software. Nesse caso, o **Python** oferece maior flexibilidade, legibilidade e muita semelhança com o tipo de programação que você já conhece e adora.

Vencedor:  Python




Round 2: Robustez e rapidez

- ▶ O **Python** se encaixa mais naturalmente em um ambiente de codificação complexo. Embora as aplicações de **R** estejam definitivamente em uma trajetória de crescimento, o **Python** ainda é uma linguagem de programação mais completa e usada para muitos propósitos, além da sua utilidade na ciência de dados.



Round 2: Robustez e rapidez

- ▶ O **Python** se encaixa mais naturalmente em um ambiente de codificação complexo. Embora as aplicações de **R** estejam definitivamente em uma trajetória de crescimento, o **Python** ainda é uma linguagem de programação mais completa e usada para muitos propósitos, além da sua utilidade na ciência de dados.
- ▶  **R**, por outro lado, ainda é usado principalmente para modelagem estatística avançada e análise de dados.



Round 2: Robustez e rapidez

- Supondo que você queira integrar seus algoritmos de aprendizado de máquina (AM) em algum tipo de interface que esteja se comunicando com outro código, escrito por outros programadores, o **Python** pode ser a melhor escolha. **R** pode ser usado para prototipagem rápida ou para resolver um problema específico, mas o **Python** será mais fácil de manter e escalar a longo prazo (especialmente considerando que seu controle de versão e documentação são muito mais consistentes).

Vencedor:  Python





Round 3: Bibliotecas externas

- ▶ Ambas as linguagens têm uma variedade de bibliotecas externas que podem ser facilmente usadas nos seus projetos de AM, as do **Python** são um pouco mais maduras. Temos, por exemplo, o **scikit-learn**, um pacote de AM de código aberto extremamente popular.





Round 3: Bibliotecas externas

- ▶ Ambas as linguagens têm uma variedade de bibliotecas externas que podem ser facilmente usadas nos seus projetos de AM, as do **Python** são um pouco mais maduras. Temos, por exemplo, o **scikit-learn**, um pacote de AM de código aberto extremamente popular.
- ▶ As bibliotecas  estão atualizando constantemente, mas ainda não estão no nível do **Python** quando se trata de funcionalidade. Com o , você poderá criar e lançar seu primeiro modelo mais rapidamente - mas o domínio do pacote **scikit** e bibliotecas semelhantes fornecerá um conjunto de ferramentas mais profundo e completo para as suas análises.




Round 3: Bibliotecas externas

- ▶ Ambas as linguagens têm uma variedade de bibliotecas externas que podem ser facilmente usadas nos seus projetos de AM, as do **Python** são um pouco mais maduras. Temos, por exemplo, o **scikit-learn**, um pacote de AM de código aberto extremamente popular.
- ▶ As bibliotecas  estão atualizando constantemente, mas ainda não estão no nível do **Python** quando se trata de funcionalidade. Com o , você poderá criar e lançar seu primeiro modelo mais rapidamente - mas o domínio do pacote **scikit** e bibliotecas semelhantes fornecerá um conjunto de ferramentas mais profundo e completo para as suas análises.

Vencedor:  Python







Round 4: Desempenho com megadatos (*Big data*)

- ▶  pode oferecer melhor desempenho ao realizar cálculos grandes. O AM geralmente envolve o trabalho com grandes conjuntos de dados e cálculos altamente complexos para treinar e testar seus algoritmos - portanto, você deve garantir que a linguagem de programação que você usa irá executar nesse tipo de cenário.







Round 4: Desempenho com megadatos (*Big data*)

- ▶  pode oferecer melhor desempenho ao realizar cálculos grandes. O AM geralmente envolve o trabalho com grandes conjuntos de dados e cálculos altamente complexos para treinar e testar seus algoritmos - portanto, você deve garantir que a linguagem de programação que você usa irá executar nesse tipo de cenário.
- ▶ Enquanto o  e o **Python** podem se integrar ao **Hadoop** para megadatos, os pacotes  mais recentes utilizam linguagem C para fornecer melhor desempenho para computação em grande escala. Portanto, você pode obter resultados mais rápidos ao usar o  nessas situações.






Round 4: Desempenho com megadatos (*Big data*)

- ▶  pode oferecer melhor desempenho ao realizar cálculos grandes. O AM geralmente envolve o trabalho com grandes conjuntos de dados e cálculos altamente complexos para treinar e testar seus algoritmos - portanto, você deve garantir que a linguagem de programação que você usa irá executar nesse tipo de cenário.
- ▶ Enquanto o  e o **Python** podem se integrar ao **Hadoop** para megadatos, os pacotes  mais recentes utilizam linguagem **C** para fornecer melhor desempenho para computação em grande escala. Portanto, você pode obter resultados mais rápidos ao usar o  nessas situações.

Vencedor: 







Round 5: Estatísticas e Visualização de Dados

- ▶ Aqui,  é o **vencedor indiscutível!** O  foi criado como uma ferramenta para fornecer uma plataforma robusta para análises estatísticas avançadas e nesse aspecto o  leva uma grande vantagem. A sua integração com **ggplot2** também permite a criação de visualizações realmente bacanas, incluindo gráficos e tabelas interativos.







Round 5: Estatísticas e Visualização de Dados

- ▶ Aqui,  é o **vencedor indiscutível!** O  foi criado como uma ferramenta para fornecer uma plataforma robusta para análises estatísticas avançadas e nesse aspecto o  leva uma grande vantagem. A sua integração com **ggplot2** também permite a criação de visualizações realmente bacanas, incluindo gráficos e tabelas interativos.
- ▶ Embora o **Python** possa ser usado para análises estatísticas e visualização de dados,  sem dúvida será a **melhor escolha** para esse tipo de funcionalidade, especialmente quando se trata de operações pontuais, prototipagem e teste de várias hipóteses.



Round 5: Estatísticas e Visualização de Dados

- ▶ Aqui,  é o **vencedor indiscutível!** O  foi criado como uma ferramenta para fornecer uma plataforma robusta para análises estatísticas avançadas e nesse aspecto o  leva uma grande vantagem. A sua integração com **ggplot2** também permite a criação de visualizações realmente bacanas, incluindo gráficos e tabelas interativos.
- ▶ Embora o **Python** possa ser usado para análises estatísticas e visualização de dados,  sem dúvida será a **melhor escolha** para esse tipo de funcionalidade, especialmente quando se trata de operações pontuais, prototipagem e teste de várias hipóteses.

Vencedor: 



Instalando o Python 3

- **Windows:** Acesse o site oficial <https://www.python.org/> e clique em ► Downloads :

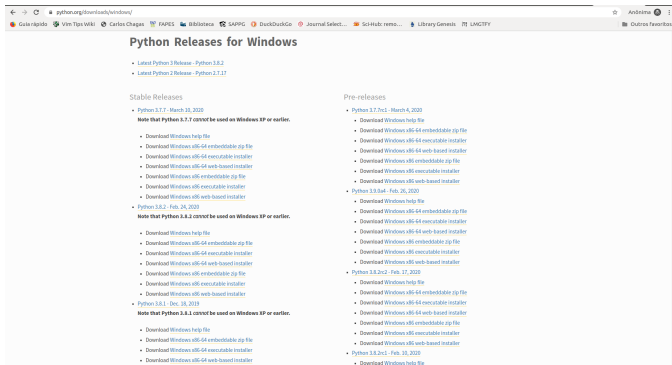
The screenshot shows the Python.org website with the following content:

- Navigation bar: Python, PEP, Docs, PyPI, Jobs, Community
- Search bar: Search, Go, Socialize
- Menu: About, Downloads, Documentation, Community, Success Stories, News, Events
- Section: Download the latest source release
- Buttons: Download Python 3.8.2, Looking for Python with a different OS? Python for Windows, Linux/OSX, Mac OS X, Other
- Text: Want to help test development versions of Python? Pre-releases, Under review, Looking for Python 2.7? See below for specific releases
- Image: Illustration of two parachutes with boxes hanging from them.
- Section: Looking for a specific release?
- Text: Python releases by version number:

Release version	Release date		Click for more
Python 3.7.7	March 16, 2020	Download	Release Notes
Python 3.8.2	Feb. 24, 2020	Download	Release Notes
Python 3.6.1	Dec. 18, 2019	Download	Release Notes
Python 3.7.6	Dec. 18, 2019	Download	Release Notes
Python 3.6.10	Dec. 18, 2019	Download	Release Notes
Python 3.5.9	Nov. 2, 2019	Download	Release Notes
Python 3.5.8	Oct. 20, 2019	Download	Release Notes



Selecione o instalador apropriado para a arquitetura do seu computador (32 ou 64 bits):



The screenshot shows the 'Python Releases for Windows' page on the official Python website. The page is organized into three main sections: 'Stable Releases', 'Pre-releases', and a 'Downloads' section at the bottom. The 'Stable Releases' section lists Python 3.7.7 (March 18, 2020) and Python 3.8.2 (Feb. 24, 2020). For each version, there is a note about Windows XP compatibility and a list of download links for different architectures (x86, x86-64) and installation methods (help file, executable zip file, executable installer, web-based installer). The 'Pre-releases' section lists Python 3.7.7rc1 (March 4, 2020) and Python 3.8.0a1 (Feb. 26, 2020), also with download links. The 'Downloads' section at the bottom provides links for the Python 3.8.2 installer and the Python 3.8.2 web-based installer.

python.org/downloads/windows/

Python Releases for Windows

- Latest Python 3 Release - Python 3.8.2
- Latest Python 2 Release - Python 2.7.17

Stable Releases

- Python 3.7.7 - March 18, 2020
Note that Python 3.7.7 cannot be used on Windows XP or earlier.
 - Download Windows help file
 - Download Windows x86-64 embeddable zip file
 - Download Windows x86-64 executable installer
 - Download Windows x86-64 web-based installer
 - Download Windows x86 embeddable zip file
 - Download Windows x86 executable installer
 - Download Windows x86 web-based installer
- Python 3.8.2 - Feb. 24, 2020
Note that Python 3.8.2 cannot be used on Windows XP or earlier.
 - Download Windows help file
 - Download Windows x86-64 embeddable zip file
 - Download Windows x86-64 executable installer
 - Download Windows x86-64 web-based installer
 - Download Windows x86 embeddable zip file
 - Download Windows x86 executable installer
 - Download Windows x86 web-based installer
- Python 3.8.1 - Dec. 18, 2019
Note that Python 3.8.1 cannot be used on Windows XP or earlier.
 - Download Windows help file
 - Download Windows x86-64 embeddable zip file
 - Download Windows x86-64 executable installer
 - Download Windows x86-64 web-based installer

Pre-releases

- Python 3.7.7rc1 - March 4, 2020
 - Download Windows help file
 - Download Windows x86-64 embeddable zip file
 - Download Windows x86-64 executable installer
 - Download Windows x86-64 web-based installer
 - Download Windows x86 embeddable zip file
 - Download Windows x86 executable installer
 - Download Windows x86 web-based installer
- Python 3.8.0a1 - Feb. 26, 2020
 - Download Windows help file
 - Download Windows x86-64 embeddable zip file
 - Download Windows x86-64 executable installer
 - Download Windows x86-64 web-based installer
 - Download Windows x86 embeddable zip file
 - Download Windows x86 executable installer
 - Download Windows x86 web-based installer
- Python 3.8.0rc2 - Feb. 17, 2020
 - Download Windows help file
 - Download Windows x86-64 embeddable zip file
 - Download Windows x86-64 executable installer
 - Download Windows x86-64 web-based installer
 - Download Windows x86 embeddable zip file
 - Download Windows x86 executable installer
 - Download Windows x86 web-based installer
- Python 3.8.0rc1 - Feb. 10, 2020
 - Download Windows help file



O processo de instalação é bem simples. Marque a opção **“Add Python to PATH”** e clique em **“Install Now”**:





Instalando o **Python 3**

- **Linux:** Se você usa GNU/Linux, provavelmente já possui alguma versão do **Python** instalada.
Para conferir, digite no terminal:

Terminal

```
$ which python3
```

deve retornar algo como `/usr/bin/python`. Isso significa que **Python** está instalado nesse endereço.



Se o **Python** não está instalado. Digite no terminal:

Terminal

```
$ sudo apt-get install python3
```

Para instalar o gerenciador de pacotes **pip**, digite no terminal:

Terminal

```
$ sudo apt-get install python3-pip
```

para verificar a versão que está instalada digite no terminal:

Terminal

```
$ python --version
```



Alguns editores de texto recomendados

- ▶ **ViM:** é o MELHOR editor de textos do mundo! O **ViM** (Vi IMproved) usa atalhos de teclado para receber comandos sobre como processar o seu texto (em vez de ícones clicáveis), o que permite dar mais eficiência à edição dos textos. Ele é indicado para qualquer linguagem de programação.

Site oficial: <https://www.vim.org/download.php>

SO: Windows, macOS, Linux.



- **GNU Emacs:** é um editor de texto usado por programadores e usuários que necessitam desenvolver documentos técnicos, em diversos sistemas operacionais. Ele é indicado para qualquer linguagem de programação.

Site oficial: <http://www.gnu.org/software/emacs/>

SO: Windows, macOS, Linux.



- **SpaceEmacs**: é uma customização do **GNU Emacs**, uma ferramenta profissional para desenvolvimento. Ele pode tirar proveito de todos os recursos do **GNU Emacs**, incluindo interfaces gráficas e de linha de comando, e ser executável no X Window System e dentro de um terminal do Unix. Basicamente, tudo que tem no **GNU Emacs** e no **ViM** em um único lugar! Precisa possuir o **GNU Emacs** instalado.

Site oficial: <https://www.spacemacs.org/>

SO: Windows, macOS, Linux.

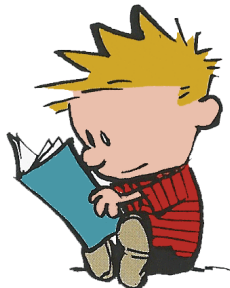
Mais editores de texto

https://en.wikipedia.org/wiki/Comparison_of_text_editors



Qual a melhor IDE para programar em **Python?**

IDE: *Integrated Development Environment* ou **Ambiente de Desenvolvimento Integrado**.





Qual a melhor IDE para programar em **Python?**

Essa simples indagação costuma despertar a ira de algumas pessoas.





Alguns IDE's para **Python**

- ▶ **IDLE**: é um ambiente de desenvolvimento integrado para **Python**. É um dos melhores IDE para **Python**, especialmente para iniciantes. Ele vem com o **Python** e é completamente escrito na mesma linguagem, toda a parte gráfica foi desenvolvida utilizando a biblioteca Tkinter, um bind (versão) da biblioteca Tk disponibilizada nativamente para **Python**. De forma geral, o objetivo do **IDLE** é proporcionar uma maneira rápida e fácil para o uso das funções e bibliotecas do **Python**, como também, proporcionar uma plataforma de estudo simples de ser utilizada, disponibilizando num único local, todos os recursos e bibliotecas do **Python**.

Site oficial: <https://www.python.org/downloads/>

SO: Windows, macOS, Linux.



Alguns IDE's para **Python**

- ▶ **PyCharm**: é um IDE **Python** gratuito, personalizável e de código aberto . Acredita-se que seja um dos melhores softwares de **Python** que inclui todos os recursos de desenvolvimento. Além disso, inclui o desenvolvimento de **Python** para mecanismos do Google. Alguns recursos que oferece: Inspeção de código, Capaz de corrigir erros de forma eficaz e ainda pode destacá-los adequadamente, entre outras.

Site oficial: <http://www.jetbrains.com/pycharm/>

SO: Windows, macOS, Linux.



Alguns IDE's para **Python**

- **Spyder**: foi projetado especialmente para programação científica com **Python**. O Spyder foi desenvolvido por e para cientistas que podem se integrar ao Matplotlib, SciPy, NumPy, Pandas, Cython, IPython, SymPy e outros softwares de código aberto. O Spyder está disponível na distribuição **Anaconda**.

Site oficial: <https://www.spyder-ide.org/>

SO: Windows, macOS, Linux.



Alguns IDE's para Python

- ▶ **Spyder**: foi projetado especialmente para programação científica com **Python**. O Spyder foi desenvolvido por e para cientistas que podem se integrar ao Matplotlib, SciPy, NumPy, Pandas, Cython, IPython, SymPy e outros softwares de código aberto. O Spyder está disponível na distribuição **Anaconda**.

Site oficial: <https://www.spyder-ide.org/>

SO: Windows, macOS, Linux.

- ▶ **Atom**: é uma ferramenta útil de edição de código preferida pelos programadores devido à sua interface simples em comparação com os outros editores. O **Atom** é desenvolvido pelo Github.

Site oficial: <https://atom.io/>

SO: Windows, macOS, Linux.



Alguns IDE's para Python

- **Thonny**: é uma IDE simples para iniciantes. Possui interface amigável e fácil de usar. Inclui alguns recursos básicos, como destaque de erro de sintaxe, escopos de explicação e uma GUI de **pip** simples e limpa. Possui janelas diferentes para aplicar chamadas de função.

Site oficial: <https://thonny.org/>

SO: Windows, macOS, Linux.

Mais IDE's

<https://wiki.python.org/moin/PythonEditors>



Qual a melhor IDE para programar em **Python?**

Existem muitas variáveis que devem ser consideradas para responder essa pergunta. De fato, é quase impossível ter uma resposta única e definitiva que vai te deixar feliz para sempre.



Qual a melhor IDE para programar em **Python?**

Análise de dados?

*Aplicações com **GUI** para desktop?*

Aplicações web?

Qual o sistema operacional que você usa?

Qual o tipo de aplicação que pretende desenvolver: Scripts para linha de comando?



Qual a melhor IDE para programar em Python?

A minha resposta

Não existe a “MELHOR” **IDE**, existe aquela que você melhor se adapta e que te deixa mais confortável para programar.

Se existisse a “MELHOR”, todos estariam usando.

Python é uma linguagem que se destaca por ser intuitiva, legível e de código fácil de manter. Por isso, um bom **editor de textos** e o **interpretador** interativo devem ser o suficiente para a maior parte dos casos.



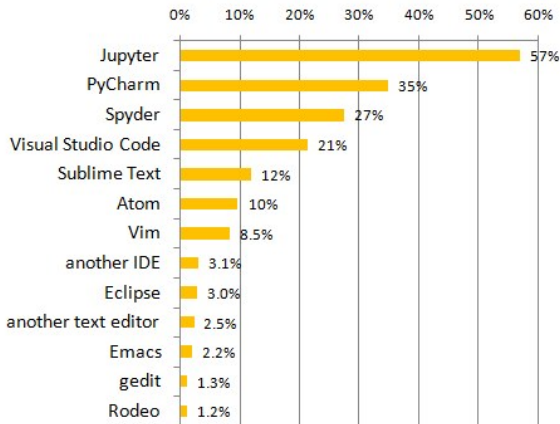
Qual a melhor IDE **Python** para Data Science?

Data Science

Em 2018, o KDnuggets criou uma enquete para avaliar dentre as IDE's **Python** para Ciência de Dados, qual delas os usuários julgaram ser a melhor.



Most Popular Python IDE, Editors





Jupyter Notebook

Experiência dos usuários

Os resultados da pesquisa mostram que, aproximadamente, 57% dos usuários destacam o **Jupyter notebook** como a melhor IDE. O que não é surpreendente, uma vez que o **projeto Jupyter**, desde 2014 tem investido bastante em **Data Science** oferecendo suporte interativo para construção de aplicações, o que de fato, tem atraído muitos desenvolvedores.

<https://jupyter.org/>



Jupyter Notebook

Jupyter notebook

O Jupyter Notebook (JN) é um ambiente computacional web interativo para criação de documentos. O documento é um documento JSON (*JavaScript Object Notation*) com um esquema e contém uma lista ordenada de células que podem conter código, texto, fórmulas matemáticas, plotagens e imagens. A extensão dos notebooks é “.ipynb”.





Jupyter Notebook

Jupyter notebook

O Jupyter Notebook (JN) é um ambiente computacional web interativo para criação de documentos. O documento é um documento JSON (*JavaScript Object Notation*) com um esquema e contém uma lista ordenada de células que podem conter código, texto, fórmulas matemáticas, plotagens e imagens. A extensão dos notebooks é “.ipynb”.

Algumas vantagens

- ▶ Os documentos JN podem ser convertidos em outros formatos como $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$, **Python**, PDF, HTML, etc.





Jupyter Notebook

Jupyter notebook

O Jupyter Notebook (JN) é um ambiente computacional web interativo para criação de documentos. O documento é um documento JSON (*JavaScript Object Notation*) com um esquema e contém uma lista ordenada de células que podem conter código, texto, fórmulas matemáticas, plotagens e imagens. A extensão dos notebooks é “.ipynb”.

Algumas vantagens

- ▶ Os documentos JN podem ser convertidos em outros formatos como $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$, **Python**, PDF, HTML, etc.
- ▶ Suporta mais de 40 linguagens de programação, entre elas:  **Python**,  **R**, Julia, Ruby, entre outras.



Jupyter Notebook

Instalação

A maneira mais simples de instalar o **Jupyter Notebook** é instalar a distribuição **Anaconda**.





Jupyter Notebook

Instalação

A maneira mais simples de instalar o **Jupyter Notebook** é instalar a distribuição **Anaconda**.

Execução

- ▶ Para rodar o Jupyter Notebook basta digitar na linha de comando:
`$ jupyter notebook` (ou `$ jupyter-notebook`).



Jupyter Notebook

Instalação

A maneira mais simples de instalar o **Jupyter Notebook** é instalar a distribuição **Anaconda**.

Execução

- ▶ Para rodar o Jupyter Notebook basta digitar na linha de comando:
`$ jupyter notebook` (ou `$ jupyter-notebook`).
- ▶ Ainda tem a opção de clicar no ícone do **Jupyter Notebook** na pasta de instalação do **Anaconda**.



Uma aba com a pasta padrão do **Jupyter Notebook** se abrirá no seu navegador padrão.

jupyter

Quit Logout

Files Running Clusters

Select items to perform actions on them.

Upload New

Name	Last Modified	File size
/		
anaconda3	2 dias atrás	
css	2 anos atrás	
deja-oup	9 meses atrás	
Documentos	13 dias atrás	
Downloads	8 minutos atrás	
Dropbox	8 horas atrás	
figure	2 anos atrás	
Imagens	7 horas atrás	
images	2 anos atrás	
js	2 anos atrás	
Meu PhotoFilmStrips	3 anos atrás	
Modelos	4 anos atrás	
Música	4 anos atrás	
ProgramasRFB	um ano atrás	
Público	4 anos atrás	
R	4 anos atrás	
seaborn-data	5 meses atrás	
snap	20 dias atrás	
videos_estatistica	3 anos atrás	
VirtualBox VMs	3 anos atrás	
Videos	2 anos atrás	
Área de Trabalho	25 dias atrás	



Para criar um novo notebook, clicar em **New** e escolher qual a linguagem do novo notebook. No nosso caso, o **Python 3**

The screenshot shows the JupyterLab web interface. At the top, there's a header with the 'jupyter' logo and 'Quit' and 'Logout' buttons. Below the header, there are tabs for 'Files', 'Running', and 'Clusters'. A message says 'Select items to perform actions on them.' Below this, there's a file browser view showing a list of files and folders. The 'New' button in the top right of the file browser is highlighted with a red circle. The file browser shows a list of files and folders with columns for 'Name', 'Last Modified', and 'File size'.

Name	Last Modified	File size
anaconda3	2 dias atrás	
css	2 anos atrás	
deja-dup	9 meses atrás	
Documentos	13 dias atrás	
Downloads	0 minutos atrás	
Dropbox	8 horas atrás	
figure	2 anos atrás	
Imagens	7 horas atrás	
images	2 anos atrás	
js	2 anos atrás	
Meu PhotoFilmStrips	3 anos atrás	
Modelos	4 anos atrás	
Música	4 anos atrás	
ProgramasRFB	um ano atrás	
Público	4 anos atrás	
R	4 anos atrás	
seaborn-data	8 meses atrás	
snap	20 dias atrás	
videos_estatistica	3 anos atrás	
VirtualBox VMs	3 anos atrás	
Videos	2 anos atrás	
Área de Trabalho	25 dias atrás	



Para criar um novo notebook, clicar em **New** e escolher qual a linguagem do novo notebook. No nosso caso, o **Python 3**.

jupyter

Quit

Login

Files Running Clusters

Select items to perform actions on them.

The screenshot shows the JupyterLab interface. On the left is a file browser with a tree view containing folders like 'anaconda3', 'css', 'deja-dup', 'Downloads', 'Dropbox', 'figure', 'Imagens', 'Images', 'js', 'Meu PhotoFilmStrips', 'Modelos', 'Música', 'ProgramasRFB', 'Público', 'R', 'seaborn-data', 'snap', 'videos_estatistica', 'VirtualBox VMs', 'Videos', and 'Área de Trabalho'. On the right is a table with columns 'Name' and 'Last modified'. The 'New' button in the top right is highlighted with a red box, and a dropdown menu is open showing options: 'Notebook: Julia 1.0.5', 'Python 3', 'R', 'Other...', 'Text File', 'Folder', and 'Terminal'. The 'Python 3' option is selected, and a tooltip says 'Create a new notebook with Python 3'. The table below shows a list of files with their names and last modified dates.

Name	Last modified
9 horas atrás	
2 anos atrás	
uma hora atrás	
2 anos atrás	
2 anos atrás	
2 anos atrás	
3 anos atrás	
4 anos atrás	
4 anos atrás	
4 anos atrás	
um ano atrás	
4 anos atrás	
4 anos atrás	
8 meses atrás	
20 dias atrás	
3 anos atrás	
3 anos atrás	
2 anos atrás	
25 dias atrás	




Observações

- ▶ A instalação do JN também instalará o kernel IPython. Isso permitirá trabalhar em notebooks usando o **Python**.




Observações

- ▶ A instalação do JN também instalará o kernel IPython. Isso permitirá trabalhar em notebooks usando o **Python**.
- ▶ Para executar notebooks em outras linguagens, como  **R** ou **Julia**, você precisará instalar kernels adicionais. Consulte a lista completa em:
<https://github.com/jupyter/jupyter/wiki/Jupyter-kernels>




Observações

- ▶ A instalação do JN também instalará o kernel IPython. Isso permitirá trabalhar em notebooks usando o **Python**.
- ▶ Para executar notebooks em outras linguagens, como  **R** ou **Julia**, você precisará instalar kernels adicionais. Consulte a lista completa em:
<https://github.com/jupyter/jupyter/wiki/Jupyter-kernels>
- ▶ A documentação do JN está disponível em:
<https://jupyter.readthedocs.io/en/latest/projects/doc-proj-categories.html>



Observações

- ▶ A instalação do JN também instalará o kernel IPython. Isso permitirá trabalhar em notebooks usando o **Python**.
- ▶ Para executar notebooks em outras linguagens, como  **R** ou **Julia**, você precisará instalar kernels adicionais. Consulte a lista completa em:
<https://github.com/jupyter/jupyter/wiki/Jupyter-kernels>
- ▶ A documentação do JN está disponível em:
<https://jupyter.readthedocs.io/en/latest/projects/doc-proj-categories.html>

Para parar a execução do JN deve-se fechar o terminal que abriu em paralelo. Fechar apenas a aba **não** para a execução.



Objetos no Python



Tipos de dados em **Python**

- ▶ Em **Python**, tudo é tratado como sendo um **objeto**, inclusive, o próprio código escrito por nós!

Objetos

- ▶ Objetos são a abstração do **Python** para dados;



Tipos de dados em **Python**

- ▶ Em **Python**, tudo é tratado como sendo um **objeto**, inclusive, o próprio código escrito por nós!

Objetos

- ▶ Objetos são a abstração do **Python** para dados;
- ▶ Todos os dados em um programa **Python** são representados por objetos ou por relações entre objetos;



Tipos de dados em **Python**

- ▶ Em **Python**, tudo é tratado como sendo um **objeto**, inclusive, o próprio código escrito por nós!

Objetos

- ▶ Objetos são a abstração do **Python** para dados;
- ▶ Todos os dados em um programa **Python** são representados por objetos ou por relações entre objetos;
- ▶ Todo objeto tem uma **identidade**, um tipo e um valor.



Tipos de dados em **Python**

- ▶ Em **Python**, tudo é tratado como sendo um **objeto**, inclusive, o próprio código escrito por nós!

Objetos

- ▶ Objetos são a abstração do **Python** para dados;
- ▶ Todos os dados em um programa **Python** são representados por objetos ou por relações entre objetos;
- ▶ Todo objeto tem uma **identidade**, um **tipo** e um valor.



Tipos de dados em **Python**

- ▶ Em **Python**, tudo é tratado como sendo um **objeto**, inclusive, o próprio código escrito por nós!

Objetos

- ▶ Objetos são a abstração do **Python** para dados;
- ▶ Todos os dados em um programa **Python** são representados por objetos ou por relações entre objetos;
- ▶ Todo objeto tem uma **identidade**, um **tipo** e um **valor**.



Tipos de dados em Python

Identidade

A identidade de um objeto **nunca** muda depois de criada. Você pode pensar nela como o endereço do objeto na memória.



Tipos de dados em Python

Identidade

A identidade de um objeto **nunca** muda depois de criada. Você pode pensar nela como o endereço do objeto na memória.

Tipo

- ▶ **Nenhum:** representa a ausência de um valor;



Tipos de dados em Python

Identidade

A identidade de um objeto **nunca** muda depois de criada. Você pode pensar nela como o endereço do objeto na memória.

Tipo

- ▶ **Nenhum:** representa a ausência de um valor;
- ▶ **Números:** Python tem três tipos de números:
 - ▶ Inteiro (`int`): Não possui parte fracionária;
 - ▶ Ponto flutuante (`float`): pode armazenar números com uma parte fracionária;
 - ▶ Complexo (`complex`): Pode armazenar partes reais e imaginárias;



Tipos de dados em **Python**

Tipo

- ▶ **Booleanos** (Booleans): Os booleanos em **Python** são: True (Verdadeiro) e False (Falso).



Tipos de dados em **Python**

Tipo

- ▶ **Booleanos** (Booleans): Os booleanos em **Python** são: True (Verdadeiro) e False (Falso).
- ▶ **Sequências**: São coleções ordenadas de elementos. Existem três tipos de sequências em **Python**:
 - ▶ Cadeias de caracteres ou *String* (`str`): Um *String* representa um conjunto de caracteres disposto numa determinada ordem. Todas as vezes em que falarmos o termo *String*, estaremos nos referindo a um conjunto de caracteres;



Tipos de dados em **Python**

Tipo

- ▶ **Booleanos** (Booleans): Os booleanos em **Python** são: True (Verdadeiro) e False (Falso).
- ▶ **Sequências**: São coleções ordenadas de elementos. Existem três tipos de sequências em **Python**:
 - ▶ Cadeias de caracteres ou *String* (`str`): Um *String* representa um conjunto de caracteres disposto numa determinada ordem. Todas as vezes em que falarmos o termo *String*, estaremos nos referindo a um conjunto de caracteres;
 - ▶ Listas (`list`): são sequencias ordenadas de valores;



Tipos de dados em **Python**

Tipo

- ▶ **Booleanos** (Booleans): Os booleanos em **Python** são: True (Verdadeiro) e False (Falso).
- ▶ **Sequências**: São coleções ordenadas de elementos. Existem três tipos de sequências em **Python**:
 - ▶ Cadeias de caracteres ou *String* (`str`): Um *String* representa um conjunto de caracteres disposto numa determinada ordem. Todas as vezes em que falarmos o termo *String*, estaremos nos referindo a um conjunto de caracteres;
 - ▶ Listas (`list`): são sequencias ordenadas de valores;
 - ▶ Tuplas (`Tuples`): também são sequências ordenadas de valores, com a diferença de que são **imutáveis**;



Tipos de dados em Python

Tipo

- **Sets:** são a abstração da definição matemática de conjunto, ou seja uma coleção (não ordenada) de valores;



Tipos de dados em Python

Tipo

- ▶ **Sets:** são a abstração da definição matemática de conjunto, ou seja uma coleção (não ordenada) de valores;
- ▶ **Dicionários** ou *Dictionaries*: são uma coleção não ordenada de objetos representados na forma de valores-chave (key-value).

Mais tipos

Claro que existem muitos mais tipos do que estes. Lembrar que **TUDO** é objeto no **Python**, de forma que há tipos de dados que correspondem a módulos, funções, classes, métodos, arquivos e código compilado.



Tipos de dados em Python

Valor

O **valor** é qualquer informação, seja um número, texto, vídeo, etc. O **tipo** é a estrutura da informação e a forma de classificarmos os dados.



Tipos de dados em Python

Valor

O **valor** é qualquer informação, seja um número, texto, vídeo, etc. O **tipo** é a estrutura da informação e a forma de classificarmos os dados.

- ▶ Todo valor numérico deve ser capaz de ser somado (ou subtraído) a outro. De igual forma, todo texto deve ser capaz de ser concatenado a outro.



Tipos de dados em Python

Valor

O **valor** é qualquer informação, seja um número, texto, vídeo, etc. O **tipo** é a estrutura da informação e a forma de classificarmos os dados.

- ▶ Todo valor numérico deve ser capaz de ser somado (ou subtraído) a outro. De igual forma, todo texto deve ser capaz de ser concatenado a outro.
- ▶ O **Python** é capaz de converter um tipo de informação num outro tipo. Essa ação é comumente chamada de **conversão de Dados** ou **coerção de tipos**.
- ▶ Podemos converter o tipo original de um objeto em **inteiro**, **float** ou **string** usando `int`, `float` e `string`, respectivamente.



Cálculos com Python



Meu primeiro programa

- ▶ Execute o **python** e digite:

```
# Meu primeiro programa em python  
print("Olá, mundo!")
```



Meu primeiro programa

- ▶ Execute o **python** e digite:

```
# Meu primeiro programa em python  
print("Olá, mundo!")
```

```
Olá, mundo!
```

- ▶ O simbolo “#” indica um comentário. Essa linha será ignorada pelo interpretador.



Meu primeiro programa

- ▶ Execute o **python** e digite:

```
# Meu primeiro programa em python  
print("Olá, mundo!")
```

Olá, mundo!

- ▶ O simbolo “#” indica um comentário. Essa linha será ignorada pelo interpretador.
- ▶ A função `print`, mostra na tela os argumentos passados nela.
- ▶ As aspas indicam que o argumento da função é um *string*.



- No caso de erro de sintaxe (SyntaxError) por causa dos caracteres especiais, por exemplo: á, â, ã, ç, Ç, etc. Use:

```
# -*- coding: utf-8 -*-  
# Meu primeiro programa em python  
print("Olá, mundo!")
```



- ▶ No caso de erro de sintaxe (SyntaxError) por causa dos caracteres especiais, por exemplo: á, â, ã, ç, Ç, etc. Use:

```
# -*- coding: utf-8 -*-  
# Meu primeiro programa em python  
print("Olá, mundo!")
```

- ▶ Você pode fazer comentários de linha multipla usando 3 aspas duplas:

```
# -*- coding: utf-8 -*-  
"""  
comentários  
de  
linha  
múltipla  
"""
```



Cálculos com Python

Operadores aritméticos:

Operação	Operador
Adição	+
Subtração	-
Multiplicação	*
Divisão	/
Exponenciação	**
Parte inteira	//
Módulo	%



Cálculos com Python

Operadores aritméticos:

Operação	Operador
Adição	+
Subtração	-
Multiplicação	*
Divisão	/
Exponenciação	**
Parte inteira	//
Módulo	%

Precedência dos operadores

Tome cuidado com a ordem de prioridade na avaliação dos operadores numa expressão aritmética.



Cálculos com **Python**

Precedência dos operadores

As primeiras sub-expressões a serem resolvidas serão os parênteses mais internos, depois as potências, depois as multiplicações e divisões, e assim por diante.



Cálculos com Python

Precedência dos operadores

As primeiras sub-expressões a serem resolvidas serão os parênteses mais internos, depois as potências, depois as multiplicações e divisões, e assim por diante.

Em caso de dúvidas use parênteses

A maneira de alterar a ordem de execução das operações numa expressão aritmética é por meio de parênteses, sendo que eles são executados antes de tudo, a partir dos mais internos para os mais externos.



```
1  # Algumas contas com Python
2  ((5 + 30) * 20 % 3 - 4) / 10 ** 2
3  quociente = 7 // 3
4  pow(2, 3) # Use também 2**3
5
6  # Constante  $\pi$  and  $e$  com math
7  import math # Módulo para funções matemáticas
8  print(math.pi)
9  print(math.e)
10
11 # Mostrando mais casas decimais
12 print("%1.30f" % (math.pi)) # podemos representar de maneira similar à
    ↪ linguagem C
13 print("%2.30f" % (math.e))
14
15 # Calculando o fatorial de um número
16 n = 5
17 k = 3
18 print ("fatorial de %d " %n, "é ", math.factorial(n), "\nfatorial de %i "
    ↪ %k, "é ", math.factorial(k))
```



Variáveis em Python

O que é uma variável?

Variável não é mais do que um espaço de memória que reservamos para armazenar valores **temporários** que estão sendo processados.



Variáveis em Python

O que é uma variável?

Variável não é mais do que um espaço de memória que reservamos para armazenar valores **temporários** que estão sendo processados.

- ▶ Os valores são armazenados na memória RAM (Random Access Memory);
- ▶ toda variável possui um **tipo** e este será inferido conforme a valor inicial que atribuímos à variável;
- ▶ não existe limite da quantidade de variáveis que podemos declarar. Essas quantias sempre serão definidas pela quantidade de memória RAM existente.



Variáveis em Python

- Nomes de variáveis podem ter o tamanho que você achar necessário e podem conter caracteres alfanuméricos e traço-baixo (*underscore*), porém **não podem começar com números**.

Exemplo 4

```
1 # Criando uma variável em python
2 criei_uma_variavel_com_nome_gigantesco = 1.5
3 # Você pode usar a tecla TAB para autocompletar!
```

É possível usar letras maiúsculas, porém a convenção será utilizar somente letras **minúsculas** para nomes de variáveis.



Variáveis em Python

Fique atento com os nomes das variáveis!

- ▶ não podem ter espaçamentos;
- ▶ não podem começar com números;
- ▶ não podem ter caracteres especiais, tais como: \$ % - * & # { } ;
- ▶ não use nomes que comecem e terminem com dois *underscore* (..), pois o **Python** define vários métodos especiais e variáveis que usam esse padrão;
- ▶ **Python** é uma linguagem *case-sensitive*, ou seja, ele diferencia nomes de variáveis com maiúsculas e minúsculas, por exemplo: **AREA**, **Area** e **area** são três variáveis diferentes.



Palavras reservadas (*Keywords*)

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

► Descrição das keywords

- Use `help("keywords")` para ver a listagem de palavras-chave;
- Você também pode fazer uma avaliação se uma palavra é chave ou não usando o módulo *keyword*. Ex:

```
import keyword
print(keyword.kwlist) # Listagem de keywords
keyword.iskeyword("pass")
```



Características

```
aluno = "Jorge"
```



Características

1. NOME

2. VALOR

aluno

=

"Jorge"



Características

1. NOME
2. VALOR
3. TIPO

```
aluno = "Jorge"  
type(aluno)  
<class 'str'>
```



Características

1. NOME

2. VALOR

```
aluno = "Jorge"
```

```
type(aluno)
```

```
<class 'str'>
```

3. TIPO

```
print(hex(id(aluno)))
```

```
0x7f1c99da3870
```

4. ESPAÇO NA MEMÓRIA



Mais operadores

Operadores de atribuição

Operador	Exemplo		
=	a = 5		
+=	a+=5	ou	a=a+5
-=	a-=5	ou	a=a-5
=	a=5	ou	a=a*5
/=	a/=5	ou	a=a/5
%=	a%=5	ou	a=a%5
//=	a//=5	ou	a=a//5
=	a=5	ou	a=a**5



Mais operadores

Operadores de comparação

Operador	Descrição
==	Igual
!=	Não Igual (diferente)
>	Maior que
<	Menor que
>=	Maior ou Igual que
<=	Menor ou Igual que



Mais operadores

Operadores de lógicos

Operador	Descrição
<code>and</code>	Conectivo de conjunção
<code>or</code>	Conectivo de disjunção
<code>not</code>	negação

Operadores de identidade

Operador	Descrição
<code>is</code>	avalia se os objetos são idênticos
<code>is not</code>	avalia se os objetos não são idênticos



Scripts 1

- ▶ script1.ipynb: Primeiros passos no Python
- ▶



Tomada de decisão no Python



Estrutura condicional - if e else

Instrução if

Com `if` podemos indicar o bloco de instruções que deve ser executado com a avaliação de uma determinada expressão, da seguinte forma:

```
if(<<expressão a ser avaliada>>):  
    <<Bloco de instruções 1>>  
else:  
    <<Bloco de instruções 2>>
```

Indentação

No **Python**, a indentação possui função bastante especial, até porque, os blocos de instrução são delimitados pela profundidade da indentação.



Estrutura condicional - if e else

Instrução if

Com `if` podemos indicar o bloco de instruções que deve ser executado com a avaliação de uma determinada expressão, da seguinte forma:

```
if(<<expressão a ser avaliada>>):  
    Tab<<Bloco de instruções 1>>  
else:  
    Tab<<Bloco de instruções 2>>
```

Indentação

No **Python**, a indentação possui função bastante especial, até porque, os blocos de instrução são delimitados pela profundidade da indentação. **Para indentar use a tecla Tab.**



Indentação

Todos os blocos são delimitados pela profundidade da indentação. Por exemplo:

```
1  if(<<exp 1>>):  
2      <<Bloco1>>  
3      if(<<exp 2>>):  
4          <<Bloco2>>  
5      else:  
6          <<Bloco 3>>  
7  else:  
8      <<Bloco 4>>
```

As linhas 4 (Bloco2) e 6 (Bloco3) são parte do mesmo nível hierárquico.



Indentação

Todos os blocos são delimitados pela profundidade da indentação. Por exemplo:

```
1  if(<<exp 1>>):  
2      <<Bloco1>>  
3      if(<<exp 2>>):  
4          <<Bloco2>>  
5      else:  
6          <<Bloco 3>>  
7  else:  
8      <<Bloco 4>>
```

As linhas 2 (Bloco1) e 8 (Bloco4) são parte do mesmo nível hierárquico.



Instrução elif

Também é possível avaliar mais de uma condição com a instrução elif, que é a abreviatura para else if.

```
1 valor_entrada = int(input("Escreva um número: "))
2 if valor_entrada == 1:
3     print("a entrada era 1")
4 elif valor_entrada == 2:
5     print("a entrada era 2")
6 elif valor_entrada == 3:
7     print("a entrada era 3")
8 elif valor_entrada == 4:
9     print("a entrada era 4")
10 else:
11     print("o valor de entrada não era esperado em nenhum if")
```



Laços de repetição no Python



Estruturas de repetição

Estrutura while

O laço de repetição `while` repete um bloco de instrução **enquanto** a condição definida em seu cabeçalho for verdadeiro.

```
1  # Mostra os números de 1 a 10
2  x = 1
3  while(x<11):
4      print("{0}".format(x))
5      x+=1
```

O laço de repetição `while` é usado preferencialmente quando uma condição precisa ser verificada a cada iteração.



Estruturas de repetição

if e while

O `while` pode-se pensar como sendo um `if`, mas que ao invés de executar o bloco de instruções uma única vez, o mesmo executará enquanto a expressão definida seja **verdadeira**.



Estruturas de repetição

if e while

O `while` pode-se pensar como sendo um `if`, mas que ao invés de executar o bloco de instruções uma única vez, o mesmo executará enquanto a expressão definida seja **verdadeira**.

Estrutura while

O **Python** define a instrução `else` como uma estrutura dependente da instrução `while` cujo funcionamento é análogo ao estudado na instrução `if`.



```
1  # Cenário 1: Mostra os números de 0 a 4 e a palavra 'stop'
2  i = 0
3  while i < 5:
4      print(i)
5      i += 1
6  else:
7      print('stop')
8  # -----
9  # Cenário 2: Mostra os números de 0 a 3
10 i = 0
11 while i < 5:
12     print(i)
13     if i == 3:
14         break # A instrução break finaliza abruptamente a execução
15     i += 1
16 else:
17     print('stop')
```



No Cenário 2 a instrução `break` foi invocada. Dessa forma o bloco `else` não será executado, pois, o bloco `else` só será executado uma vez que a condição dada na linha 6 for **verdadeira**.

```
1  # -----
2  # Cenário 2: Mostra os números de 0 a 3
3  i = 0
4  while i < 5:
5      print(i)
6      if i == 3:
7          break # A instrução break finaliza abruptamente a execução
8      i += 1
9  else:
10     print('stop')
```



No Cenário 2 a instrução `break` foi invocada. Dessa forma o bloco `else` não será executado, pois, o bloco `else` só será executado uma vez que a condição dada na linha 6 for **verdadeira**.

```
1  # -----
2  # Cenário 2: Mostra os números de 0 a 3
3  i = 0
4  while i < 5:
5      print(i)
6      if i == 3:
7          break # A instrução break finaliza abruptamente a execução
8          i += 1
9  else:
10     print('stop')
```

A instrução `break` finaliza abruptamente a execução das estruturas de repetição.



break e continue

As instruções `break` e `continue` são ferramentas das estruturas de repetição para a interrupção do laço de repetição ou de um único ciclo.



break e continue

As instruções `break` e `continue` são ferramentas das estruturas de repetição para a interrupção do laço de repetição ou de um único ciclo.

break

A instrução `break` interrompe não somente o ciclo em execução, mas sim, todo o laço.

continue

A instrução `continue` finaliza um único laço, fazendo com que o programa execute a partir do cabeçalho da estrutura de repetição.



O faz o seguinte código?

```
1 print("inicio")
2 i = 0
3 n=20
4 while(i<n):
5     i += 1
6     if(i%2==0):
7         continue
8     if(i>n*0.5):
9         break
10    print(i)
11 else:
12    print("else")
13 print("fim")
```



Estruturas de repetição

Instrução for

A instrução for se caracteriza por obrigar a definir a quantidade de vezes que será executado.



Estruturas de repetição

Instrução for

A instrução for se caracteriza por obrigar a definir a quantidade de vezes que será executado.

Estrutura for

A estrutura for exige, inicialmente, a definição de uma variável e, em seguida, um conjunto de itens iteráveis, i.e. itens de listas, tuplas, strings, chaves de dicionários, entre muitos outros objetos:

```
for <variável> in <itens iteráveis>:  
    <<bloco>>
```



```
1  # Calcula o fatorial de um número
2  n=10
3  prod = 1
4  print("i\t Fatorial")
5  for i in range(1,n+1):
6      prod = prod * i
7      print("{0}\t{1}".format(i, prod))
```



```
1  # Calcula o fatorial de um número
2  n=10
3  prod = 1
4  print("i\t Fatorial")
5  for i in range(1,n+1):
6      prod = prod * i
7      print("{0}\t{1}".format(i, prod))
```

A função `range()` retorna uma série numérica no intervalo definido como argumento. A série retornada é um objeto iterável tipo `range` e os elementos contidos serão gerados sob demanda.

Sintaxe: `range(inicio, final, passo)`, sendo os dois últimos opcionais. Todos os argumentos devem ser **inteiros**. Não admite argumentos de tipo `string` ou `float`.



Função range()

Observe que, se executamos `print(type(range(10)))` será retornado `<class 'range'>`, e não uma lista contendo os elementos propriamente dito. Uma forma de forçar a conversão do objeto iterável para o objeto `list` é a seguinte: `list(range(10))`

```
1 # Convertendo tipo range a list
2 numeros_pares = list( range(0, 10, 2))
3 print("lista de números pares", numeros_pares)
```



Função range()

Observe que, se executamos `print(type(range(10)))` será retornado `<class 'range'>`, e não uma lista contendo os elementos propriamente dito. Uma forma de forçar a conversão do objeto iterável para o objeto `list` é a seguinte: `list(range(10))`

```
1 # Convertendo tipo range a list
2 numeros_pares = list(range(0, 10, 2))
3 print("lista de números pares", numeros_pares)
```

O passo é o pulo entre cada elemento numérico. Se por exemplo, gerarmos uma sequência numérica entre 0 e 10 e definirmos o passo como sendo igual a 2, a seguinte lista será retornada: `[0, 2, 4, 6, 8]`.



```
1  # Calcula o fatorial de um número
2  n=10
3  prod = 1
4  print("i\t Fatorial")
5  for i in range (1,n+1):
6      prod = prod * i
7      print("{0}\t{1}".format(i, prod))
```



```
1  # Calcula o fatorial de um número
2  n=10
3  prod = 1
4  print("i\t Fatorial")
5  for i in range (1,n+1):
6      prod = prod * i
7      print("{0}\t{1}".format(i, prod))
```

A função `range()` retorna uma série numérica no intervalo definido como argumento. A série retornada é um objeto iterável tipo `range` e os elementos contidos serão gerados sob demanda.

Sintaxe: `range(inicio, final, passo)`, sendo os dois últimos opcionais. Todos os argumentos devem ser **inteiros**. Não admite argumentos de tipo `string` ou `float`.



Estruturas de repetição

Instrução else no for

A instrução else pode ser utilizada com o for da mesma forma que estudamos com a instrução while.

```
1  # utilizando a instrução ELSE
2  for x in [1,2,3,4,5]:
3      print(x)
4  else:
5      print("Loop finalizado com sucesso!")
```



Estruturas de repetição

Instrução else no for

A instrução else pode ser utilizada com o for da mesma forma que estudamos com a instrução while.

```
1  # utilizando a instrução ELSE
2  for x in [1,2,3,4,5]:
3      print(x)
4  else:
5      print("Loop finalizado com sucesso!")
```

Ao executar o código acima, vemos que a estrutura for executou todos os ciclos definidos e no final executou o bloco da instrução else.



for vs while

for

- ▶ Número de iterações é conhecido;

while

- ▶ Número de iterações ilimitados;



for vs while

for

- ▶ Número de iterações é conhecido;
- ▶ Pode finalizar antecipadamente através do break;

while

- ▶ Número de iterações ilimitados;
- ▶ Pode finalizar antecipadamente por meio do break;



for vs while

for

- ▶ Número de iterações é conhecido;
- ▶ Pode finalizar antecipadamente através do break;
- ▶ utiliza um contador.

while

- ▶ Número de iterações ilimitados;
- ▶ Pode finalizar antecipadamente por meio do break;
- ▶ Pode utilizar um contador, porém é necessário inicializar ele antes do loop e incrementá-lo dentro do loop.



Listas



A classe list

Listas

Listas são estruturas de dados capazes de armazenar múltiplos elementos. **Todo elemento contido numa lista é também um objeto.**

Criando listas

Todos os elementos que estiverem delimitados por um par de colchetes para o Python serão uma **lista**.

```
1 lista1 = ["nome", "sobrenome", "idade", "endereço"]
2 lista2 = [1, 2, 3, 4, 5]
3 lista3 = [1, 2.3, "Rafael", [7, "oito", 9]]
4 lista4 = list((4,5,6,7)) # Usando a função list
```



Acessando aos elementos de uma lista

Assim como nos objetos de tipo **strings**, é possível acessar separadamente cada item de uma **lista** a partir de seu índice. Por exemplo:

```
1  # Uma lista dentro de outra lista é uma lista aninhada.
2  lista3 = [1, 2.3, "Rafael", [7, "oito", 9]]
3  lista3[2] # 'Rafael'
4  lista3[3] # [7, "oito", 9]
5  lista3[3][1] # 'oito'
6  lista3[3][1][2] # 't'
7  lista3[3][1][-3] # 'i'
```

Como no caso das **strings**, ao utilizar um índice negativo os elementos são acessados de trás pra frente.



Diferente das **strings**, as **listas** são objetos mutáveis. Por exemplo:

```
1 lista3 = [1, 2.3, "Rafael", [7, "oito", 9]]
2 lista3[3] # [7, "oito", 9]
3 lista3[3] = "novo elemento da lista"
4 lista3[3] # [1, 2.3, "Rafael", "novo elemento da lista"]
```

Índices de listas funcionam igual que nos objetos de tipo **strings**:

- ▶ Qualquer expressão de números inteiros pode ser usada como índice;
- ▶ Se tentar ler ou escrever um elemento que não existe, você recebe um **IndexError**;
- ▶ Um índice com valor negativo conta de trás para a frente, a partir do final da lista.



- Com os operadores `in` e `not in` podemos avaliar se um valor existe ou não em uma **lista**:

```
1  # Lista multidimensional
2  lista = ['frutas', ['cítricas', ['laranja', 'limão']], [7, "oito",
   ↪      9]]
3  "Frutas" in lista # False
4  "Laranja" in lista # False
5  "oito" in lista # False
6
7  "oito" not in lista # True
8  [7, "oito", 9] in lista # True
9  "oito" in lista[2] # True
10 "frutas" in lista # True
```



- ▶ Com os operadores `in` e `not in` podemos avaliar se um valor existe ou não em uma **lista**:

```
1  # Lista multidimensional
2  lista = ['frutas', ['cítricas', ['laranja', 'limão']], [7, "oito",
   ↪      9]]
3  "Frutas" in lista # False
4  "Laranja" in lista # False
5  "oito" in lista # False
6
7  "oito" not in lista # True
8  [7, "oito", 9] in lista # True
9  "oito" in lista[2] # True
10 "frutas" in lista # True
```

- ▶ Podemos obter o tamanho da **lista** utilizando a função `len()`.



Apesar de uma lista poder conter outra lista, a lista aninhada ainda conta como um único elemento. O comprimento de `lista` é 3:

```
1  lista = ['frutas', ['cítricas', ['laranja', 'limão']], [7, "oito",  
    ↪      9]]  
2  len(lista) # 3
```



Apesar de uma lista poder conter outra lista, a lista aninhada ainda conta como um único elemento. O comprimento de `lista` é 3:

```
1  lista = ['frutas', ['cítricas', ['laranja', 'limão']], [7, "oito",  
    ↪ 9]]  
2  len(lista) # 3
```

► Um `for` que passe por uma lista vazia nunca executa:

```
1  lista = []  
2  for i in lista:  
3      print('Nunca será impresso')
```



- ▶ A forma mais comum de percorrer os elementos em uma **lista** é com um **for**:

```
1  # Podemos percorrer listas com for de duas maneiras:
2  vingadores = ['Homem de Ferro', 'Hulk', 'Capitão America', 'Viúva
   ↳ Negra', 'Gavião Arqueiro']
3  # ***** Alternativa 1 *****
4  for k in vingadores:
5      print('Vingador:', k)
6
7  # ***** Alternativa 2 *****
8  for i in range(0, len(vingadores)):
9      print('Vingador:', vingadores[i])
```

A Alternativa 1 funciona bem se você precisa apenas ler os elementos da lista. Mas, se o interesse está em escrever ou atualizar os elementos da lista, você precisa dos índices.



- Para remover um elemento de uma **lista** podemos utilizar:
 - a. **O comando del:** deleta um elemento em um índice específico;

```
1 lista = [2, 1, 3, [1,"a"], 3, 7, "manga"]
2 del(lista[2]) # lista = [2, 1, [1,"a"], 3, 7, "manga"]
3 del(lista) # caso não especifique um índice, elimina sua lista
  ↪ por completo!
```

- b. **O método pop:** remove o último elemento da lista e retorna o elemento removido;

```
1 lista = [2, 1, 3, [1,"a"], 3, 7, "manga"]
2 lista.pop() # "manga"
3 lista # [2, 1, 3, [1,"a"], 3, 7]
```



c. **O método remove:** remove um elemento específico.

```
1 lista = [2, 1, 3, [1,"a"], 3, 7, "manga"]
2 lista.remove(2)
3 lista # [1, 3, [1,"a"], 3, 7, "manga"]
4 # Se o elemento ocorre múltiplas vezes, remove a 1a ocorrência
5 lista.remove(3)
6 lista # [1, [1,"a"], 3, 7, "manga"]
7 lista.remove(4) # ValueError: list.remove(x): x not in list
```

Caso o elemento não estiver presente na lista, mostrará um erro **ValueError**;



Outros métodos

- ▶ Outros métodos para objetos de tipo `list`:
 - ▶ **clear**: Remove todos os itens da lista; exemplo
 - ▶ **extend**: Adiciona um elemento de uma lista para outra lista;
 - ▶ **append**: Adiciona um único elemento para a lista; exemplo
 - ▶ **insert**: Insere um elemento na lista;
 - ▶ **copy**: Retorna uma cópia da lista; exemplo
 - ▶ **sort**: Organiza os elementos da lista em ordem ascendente; exemplo
 - ▶ **reverse**: Inverte a lista;
 - ▶ **index**: Retorna o índice de um elemento na lista.
 - ▶ **count**: Retorna ocorrências de um elemento na lista; exemplo



Funções internas (*built-in*) para listas

- ▶ As funções internas são nativas na linguagem, ou seja, já vem incorporadas no interpretador **Python** e estão sempre disponíveis para utilização.
 - ▶ **max**: Retorna o maior elemento;
 - ▶ **min**: Retorna o menor elemento;
 - ▶ **reduce**: Aplica uma função a todos os valores da lista, de forma a agregá-los em um único valor;
 - ▶ **sum**: Soma todos os elementos de uma lista; exemplo
 - ▶ **all**: Verifica se todos os elementos contidos na lista são verdadeiros ou se a lista for vazia;
 - ▶ **any**: Retorna verdadeiro se algum dos elementos da lista for verdadeiro. Se a lista for vazia ela retorna falso.



Funções internas (*built-in*) para listas

- ▶ **enumerate**: Retorna uma tupla de dois elementos a cada iteração: um número sequencial e um item da sequência correspondente;
- ▶ **filter**: filtra os elementos de uma sequência. O processo de filtragem é definido a partir de uma função que o programador passa como primeiro argumento da função; exemplo
- ▶ **map**: Aplica uma função a cada elemento de uma lista, retornando uma nova lista contendo os elementos resultantes da aplicação da função;
- ▶ **zip**: retorna uma lista de tuplas, onde a i-ésima tupla contém o i-ésimo elemento de cada um dos argumentos. exemplo

▶ Listagem completa de funções internas



Exemplos



Método clear e extend

```
1 lista1 = [2, 1, 3, 5,['a','b']]
2 lista2, lista3 = [[4, 3, 9],6, "texto"], [1,2,3]
3 lista1.extend(lista2+lista3)
4
5 print ("Nova lista1 (elementos adicionados): ", end="")
6 for i in range(0, len(lista1)):
7     print(lista1[i], end=" ")
8 print ("\r")
9
10 lista1.clear() # Deletando os elementos da lista1
11
12 print("Nova lista1 (elementos deletados): ", end="")
13 for i in range(0, len(lista1)):
14     print(lista1[i], end=" ")
```



Método append e insert

```
1  municipios = ['Serra', 'Vila Velha', 'Cariacica']
2  municipios.append('Vitória') # append adiciona um elemento ao final da
   ↪ lista
3  municipios
4  # append não permite mais de um argumento. Nesse caso utilize o método
   ↪ extend
5  #municipios.append('Castelo','Iconha', 'Guarapari','Viana')
6  municipios.extend(['Castelo','Iconha', 'Guarapari','Viana'])
7  municipios
8
9  # insert permite inserir um elemento na posição especificada
10 municipios.insert(3, 'Alegre')
11 for i in range(0, len(municipios)):
12     print(municipios[i], end=" ")
```



Método copy

```
1  lista1, lista2 = ['a', 'b', 'c'], lista1
2  print("lista1= ", lista1)
3  print("lista2= ", lista2)
4
5  lista2.append('d') # adicionando um elemento na lista 2
6  print("lista1= {}".format(lista1))
7  print("lista2= {}".format(lista2))
8
9  # Não use o operador '=' para copiar listas, utilize o método copy
10 lista1 = ['a', 'b', 'c']
11 lista2 = lista1.copy()
12 lista2.append('d')
13 print("lista1= {}".format(lista1))
14 print("lista2= {}".format(lista2))
```



Método sort, reverse e index

```
1  municipios = ['Serra', 'Vila Velha', 'Cariacica', 'Alegre', 'Vitória',  
    ↪ 'Castelo', 'Iconha', 'Guarapari', 'Viana']  
2  municipios.sort()  
3  print(municipios)  
4  municipios.reverse()  
5  print(municipios)  
6  print("O índice do município 'Castelo' é  
    ↪ {}".format(municipios.index("Castelo")))
```

Outros métodos



Método count

```
1 from random import randint
2 n,b=10,9
3 lista = list((randint(0,b) for i in range(n)))
4 contagem = [0]*n
5 print("lista de números aleatórios entre 0 e {0} = {1}".format(n,lista))
6 contagem = list(enumerate([lista.count(i) for i in range(n)]))
7 print("Valor\tfrequência")
8 for i in range(b+1):
9     print(contagem[i][0], "\t", contagem[i][1])
```

Outros métodos



Funções min, max, sum e enumerate

Funções internas

```
1  from random import randint
2  import statistics
3  a, b, n = 1, 7, 10
4  numeros = list((randint(a,b) for i in range(n)))
5  numeros.sort()
6  print("lista de numeros gerados aleatoriamente (ordenados)=
    ↳ {}".format(numeros))
7  print("Mínimo = {0} e Máximo = {1}".format(min(numeros),max(numeros)))
8  print("Média aritmética= %2.2f" %(sum(numeros)/n)+"\tMediana= %2.2f"
    ↳ %statistics.median(numeros))
9
10 contagem = list(enumerate([numeros.count(i) for i in range(a,b+1)]))
11 max_freq= max([contagem[i][1] for i in range(b-a+1)])
12 print("Valor(es) com maior ocorrência (moda): ", end= " ")
13 for i in range(b-a+1):
14     if(contagem[i][1]==max_freq):
15         print(contagem[i][0]+a, end=" ")
```



Funções filter e map Funções internas

```
1 import math
2 from random import randint
3
4 a, b, n = 1, 7, 10
5 numeros = list((randint(a,b) for i in range(n)))
6 numeros.extend([None]*6); random.shuffle(numeros)
7 print("Sequência com valores 'None': ", numeros)
8
9 res1 = list(filter(None, numeros)); print("Sequência sem 'None': ", res1)
10 pares = [i for i in res1 if i%2 == 0]; print("Números pares: ", pares)
11 maiores_dois = filter(lambda x: x > 2, res1); print("Maiores que 2:
    ↳ {}".format(list(maiores_dois)))
12
13 raiz_quad = list(map(math.sqrt, res1))
14 print("Raiz quadrada= {}".format([round(x,2) for x in raiz_quad]))
15 print("Quadrado= {}".format([round(math.pow(x,2)) for x in raiz_quad]))
```



Funções filter e map

Funções internas

```
1  from random import randint
2
3  a, b, n, valor = 10, 70, 20, b
4  numeros = list((randint(a,b) for i in range(n)))
5  print("Sequência de valores: ", numeros)
6
7  def Teste_Numero_Menor(lista, valor):
8      return(all(i < valor for i in lista))
9
10 if (Teste_Numero_Menor(numeros, valor)): print("Sim, todos o valores são
    ↳ menores que {0}. Valor máximo é {1}.".format(valor,max(numeros)))
11 else: print("Não, existe pelo menos um valor {} na
    ↳ sequência.".format(valor))
```




Funções

Funções em Python

- ▶ Uma função é um bloco de código organizado capaz de realizar uma determinada ação;
- ▶ Funções nos ajudam a deixar o código mais *modular*, trazendo a possibilidade de reusabilidade.
- ▶ **Python** traz várias funções pré-construídas, como por exemplo `print()`, `range()`, `type()`, entre outras. Funções internas



Funções

Funções em Python

- ▶ Uma função é um bloco de código organizado capaz de realizar uma determinada ação;
- ▶ Funções nos ajudam a deixar o código mais *modular*, trazendo a possibilidade de reusabilidade.
- ▶ **Python** traz várias funções pré-construídas, como por exemplo `print()`, `range()`, `type()`, entre outras. Funções internas

Também é possível criarmos nossas próprias funções!



Definindo funções

Para definir uma função é necessário levar em conta:

- ▶ Funções devem começar com a palavra-chave `def` seguido do nome da função;
- ▶ Os parâmetros de entrada (ou argumentos da função) devem ser colocados entre parênteses;
- ▶ Para retornar valores da função podemos usar `return`. O comando de retorno sem argumentos é o mesmo que `return None`.

```
1 def NOME_DA_FUNCAO(LISTA DE PARAMETROS):  
2     COMANDOS
```



Definindo funções

Para definir uma função é necessário levar em conta:

- Opcionalmente podemos usar strings de documentação (doc-strings) para descrever o que nossa função faz. Por exemplo

```
1  def bemvindo(nome):  
2      """  
3      A função 'bemvindo' é usada para dar as boas vindas a uma pessoa  
4      """  
5      print("Olá {0} Seja bem-vindo".format(nome))  
6  
7  bemvindo("Fabio") # Invoca a função  
8  print(bemvindo.__doc__) # Mostra a descrição da nossa função
```



Mais sobre funções

Importante

- ▶ Não esqueça de colocar os dois pontos após definir a função;
- ▶ Respeite a indentação na hora de definir a função;



Mais sobre funções

Importante

- ▶ Não esqueça de colocar os dois pontos após definir a função;
- ▶ Respeite a indentação na hora de definir a função;
- ▶ A *docstring*, é uma string usada para especificar a funcionalidade da nossa função;
- ▶ O uso de *docstring* é **opcional**, mas recomenda-se documentar sempre as funções. Essa importante prática de programação permite que você lembre o que a função faz;



Mais sobre funções

Importante

- ▶ Não esqueça de colocar os dois pontos após definir a função;
- ▶ Respeite a indentação na hora de definir a função;
- ▶ A *docstring*, é uma string usada para especificar a funcionalidade da nossa função;
- ▶ O uso de *docstring* é **opcional**, mas recomenda-se documentar sempre as funções. Essa importante prática de programação permite que você lembre o que a função faz;
- ▶ `return` ou `print()`? O que usar para retornar valores de uma função?



return ou print() ?

- ▶ A função `print`, mostra na tela uma string ou um número, mesmo quando o valor retornado é atribuído a uma variável;
- ▶ A instrução `return` não imprime o valor que retorna quando o mesmo é atribuído a uma variável;



return ou print() ?

- ▶ A função `print`, mostra na tela uma string ou um número, mesmo quando o valor retornado é atribuído a uma variável;
- ▶ A instrução `return` não imprime o valor que retorna quando o mesmo é atribuído a uma variável;

```
1 def soma1():  
2     print(1 + 1)  
3 a = soma1()
```

```
1 def soma2():  
2     return 1 + 1  
3 b=soma2()
```



return ou print() ?

- ▶ A função `print`, permite a execução de todas as instruções no função;
- ▶ A instrução `return` faz com que a função termine imediatamente a execução, mesmo que não seja a última instrução da função.



return ou print() ?

- ▶ A função `print`, permite a execução de todas as instruções no função;
- ▶ A instrução `return` faz com que a função termine imediatamente a execução, mesmo que não seja a última instrução da função.

```
1 def soma1():  
2     print(1 + 1)  
3     print("Imprime")  
4 a = soma1()
```

```
1 def soma2():  
2     return 1 + 1  
3     print("Imprime")  
4 b = 'soma2()'
```



return ou print()?

E o que acontece quando a função conta com argumentos?

```
1 def soma1(x,y):  
2     print(x + y)  
3 a = soma1(soma1(1,2),soma1(3,4))  
4 print(a)
```

```
1 def soma2(x,y):  
2     return x + y  
3 b=soma2(soma2(1,2),soma2(3,4))  
4 print(b)
```



return ou print() ?

E o que acontece quando a função conta com argumentos?

```
1 def soma1(x,y):  
2     print(x + y)  
3 a = soma1(soma1(1,2),soma1(3,4))  
4 print(a)
```

```
1 def soma2(x,y):  
2     return x + y  
3 b=soma2(soma2(1,2),soma2(3,4))  
4 print(b)
```

TypeError: unsupported operand type(s) for +: 'NoneType' and 'NoneType'



Retornando múltiplos valores

Uma das formas mais simples é usando objetos de tipo `list`:

```
1 import math
2 def fun():
3     texto = "Aprender python é divertido"
4     x = math.pi
5     a = ['a']*3
6     return [texto, x, a];
7 lista = fun(); print(lista)
8 um_texto, uma_constante, uma_lista = fun()
9 print(um_texto, uma_constante, uma_lista)
```

Estudaremos outras formas para retornar múltiplos valores mais adiante.