

Python: Instalação e Comandos Básicos



Eduardo Corrêa
24/01/2020

Este texto¹ explica como instalar o ambiente Python e como criar e executar os seus primeiros programas utilizando os comandos básicos da linguagem.



Lição 1 – O que é Python?

Python é uma linguagem de programação de **propósito geral**, o que significa que ela pode ser empregada nos mais diferentes tipos de projetos, variando desde aplicações Web até sistemas de inteligência artificial. A linguagem foi criada no ano de 1991, tendo como principal filosofia priorizar a construção de programas simples e legíveis (aquilo que os *pythonistas* chamam de programas “bonitos”) sobre a construção de programas que, ainda que velozes, sejam complicados e pouco legíveis (os ditos programas “feios”). No decorrer dos dez anos seguintes, a linguagem alcançou grande popularidade tanto em ambiente acadêmico como corporativo. Isto motivou o surgimento da *Python Software Foundation* no ano de 2001, uma instituição independente e sem fins lucrativos que tornou-se responsável pelo desenvolvimento de novas versões da linguagem (no momento da elaboração deste texto, a versão mais recente era a 3.8.1).

Nos últimos anos, Python consolidou-se como uma das tecnologias mais difundidas na área ciência de dados, apesar de não ter sido originalmente projetada para este fim. Essa conquista não se deveu apenas ao fato de a linguagem facilitar a criação de programas de “rosto bonito”! Na realidade, o sucesso do Python para ciência de dados está relacionado a outras de suas características, conforme apresenta-se a seguir:

- Como Python é uma linguagem **interpretada**, os iniciantes podem aprender alguns comandos e começar a fazer coisas legais (ex.: aplicar funções matemáticas e estatísticas sobre conjuntos de dados) quase que imediatamente, sem esbarrar em problemas relacionados à compilação de código. E para tornar a coisa ainda melhor, o interpretador Python pode ser utilizado de forma **interativa**, onde cada comando digitado é imediatamente traduzido e executado. Isto oferece aos programadores uma forma ágil e simples para examinar em tempo real os resultados intermediários obtidos em qualquer passo de um processo de análise de dados.
- Python é uma **linguagem livre** (*open source*). No Web site da *Python Software Foundation*² é possível baixar gratuitamente o arquivo que instala o interpretador Python e a sua biblioteca padrão (a famosa **standard library**). Juntos, estes componentes formam o “coração” do ambiente Python, oferecendo um rico conjunto de estruturas de dados (como listas e dicionários) e centenas de módulos voltados para a execução dos mais diversos tipos de tarefas, desde o uso de funções matemáticas e estatísticas até o processamento de arquivos texto em diferentes formatos (CSV, JSON, etc.).
- A linguagem Python pode ser facilmente estendida através da incorporação de outros **pacotes**. Atualmente, existem milhares de pacotes disponíveis no repositório central do Python (*Python Package Index* – PyPI). Muitos e muitos deles são voltados para ciência de dados, tais como, ‘NumPy’ (manipulação de vetores e matrizes), ‘SciPy’ (rotinas numéricas para resolver problemas de integração, equações algébricas e cálculo numérico, entre outras

1 Texto retirado do livro “Meu primeiro livro de Python” (Número ISBN: 978-65-900095-0-0), autor Eduardo Corrêa.

2 <https://www.python.org/>

coisas), ‘pandas’ (importação e transformação de bases de dados), ‘Matplotlib’ (geração de gráficos) e ‘scikit-learn’ (algoritmos de mineração de dados e aprendizado de máquina).

- Por fim, Python é uma linguagem **multiparadigma**, que permite tanto a elaboração de programas no estilo **procedural** como no **orientado a objetos**.
 - O paradigma procedural é mais simples, porém perfeito para o desenvolvimento de *scripts* de ciência de dados. Isso porque ele não exige do programador o domínio de certos conceitos que são muito específicos da área de engenharia de software³ (como, por exemplo, a definição de classes e métodos e a elaboração rotinas detalhadas para tratamento de exceções). Esta característica do estilo procedural é muito importante na área de ciência de dados, onde frequentemente os programadores não são pessoas com formação em computação, mas sim biólogos, estatísticos, químicos, etc., que desejam apenas criar um programa bem enxuto e direto para explorar uma determinada base de dados.
 - Por outro lado, o paradigma orientado a objetos é sofisticado e complexo, sendo, no entanto, o mais indicado para a construção de aplicativos e sistemas de informação. Em programas deste tipo, o desenvolvedor precisa se preocupar com uma série de questões que nem sempre fazem parte do mundo da análise de dados, tais como o gerenciamento de sessões, o controle de acesso e a criação de menus, apenas para citar algumas poucas. É exatamente no paradigma orientado a objetos que são oferecidas as ferramentas mais adequadas para lidar com estes problemas. Porém, vale a pena observar que um pouco de conhecimento sobre orientação a objetos é importante para qualquer cientista de dados, pelo fato de este paradigma ser o mais recomendado para a construção de **pacotes**.

Muito Prazer, Linguagem Python

- Python é uma linguagem de propósito geral, interpretada, interativa, gratuita e multiparadigma.
- Trata-se de uma linguagem extremamente versátil e poderosa que possui um grande número de pacotes para ciência de dados (pacotes de estatística, matemática, mineração de dados, inteligência artificial, aprendizado de máquina, cálculo numérico, etc.). Por isso, tem sido adotada por um número cada vez maior de empresas.



Lição 2 – Distribuições do Python

Por ser uma “linguagem-tudo” (de propósito geral), gratuita e muito popular, Python possui um impressionante **ecossistema** de pacotes que já atingiu a casa das dezenas de milhares! Eles são destinados aos mais diferentes tipos de problemas: alguns servem para a criação de jogos, muitos deles são para o desenvolvimento de aplicações Web, alguns outros para o desenvolvimento de sistemas embarcados, há um bom número que oferece algoritmos de aprendizado de máquina, e por

³ Engenharia de software é a subárea da ciência computação que trata da pesquisa e desenvolvimento de técnicas que visem garantir alta qualidade e produtividade no processo de criação de softwares.

aí vai. Não há nem como tentar fazer uma lista completa, pois o número de categorias de problemas distintos cobertos pelos pacotes Python é realmente enorme!

Esta situação motivou o surgimento de diferentes **distribuições** do ambiente Python. Uma distribuição (ou *distro*), nada mais é do que uma coleção de pacotes e aplicativos selecionados, que são reunidos em um único **arquivo instalador** por serem considerados os “mais adequados” para uma determinada finalidade. A Figura 1 apresenta dois exemplos de distribuições muito conhecidas, CPython e WinPython.

Distribuição CPython	Distribuição WinPython
<ul style="list-style-type: none">• Interpretador Python• Standard Library• Utilitários básicos (IPython, pip, ...)	<ul style="list-style-type: none">• Interpretador Python• Standard Library• Utilitários básicos (IPython, pip, ...)• 'NumPy'• 'pandas'• 'Matplotlib'• 'scikit-learn'• 'SciPy'• 'Spyder'• 'Jupyter Notebook'• ...

Figura 1. Duas diferentes distribuições do ambiente Python: CPython e WinPython

CPython é a **distribuição oficial** do Python, gerenciada pela *Python Software Foundation*, cujo arquivo instalador para diferentes plataformas – Windows, Mac, Linux – pode ser obtido a partir de <https://www.python.org/>. Trata-se da versão de referência do Python, que vem basicamente com a dupla interpretador + *standard library* e um pequeno conjunto de utilitários básicos. Pode-se dizer que é uma distribuição “neutra”. Nenhum pacote específico para ciência de dados é instalado por esta distribuição. No entanto, o CPython disponibiliza um aplicativo chamado “**pip**” que possibilita a posterior instalação de qualquer pacote que faça parte do repositório PyPI. Informações sobre o CPython e o “pip” são apresentadas no Anexo B.

Por sua vez, a distribuição **WinPython** foi criada exclusivamente para o ambiente Windows, sendo destinada aos que desejam trabalhar com Python para ciência de dados. A sua característica mais atraente é fato de que ela não precisa ser instalada; na verdade, basta descompactar o arquivo instalador em alguma pasta de seu computador e, pronto, você já pode começar programar! A WinPython já vem com mais de 300 pacotes e aplicativos para ciência de dados: ‘NumPy’, ‘pandas’, ‘Matplotlib’, ‘Spyder’, etc., que podem ser usados imediatamente, sem a necessidade de nenhuma etapa de instalação adicional.

Devido à simplicidade de seu processo de configuração e pelo fato de ser uma distribuição voltada para ciência de dados, decidimos adotar a WinPython como o ambiente Python padrão para a execução dos exemplos apresentados neste texto. Na próxima lição, são apresentados os passos necessários para efetuar o seu download e instalação. No entanto, é importante observar que esta não é a melhor distribuição do Python, ela é apenas um bom ponto de partida para quem está começando a trabalhar com esta linguagem. Outro problema da WinPython é que ela não está disponível para Linux e Mac. Caso você utilize estes sistemas operacionais, uma alternativa interessante é utilizar a distribuição Anaconda (<https://www.anaconda.com/download/>), que também é voltada para ciência de dados, sendo inclusive bem mais abrangente (vem como mais de 700 pacotes). Outra opção é montar um ambiente a partir da distribuição CPython, porém instalando também os principais

pacotes para ciência de dados com o uso do “pip”. No Anexo B, são fornecidas instruções para a instalação do CPython e sobre a forma de utilização do “pip”.

Versões do Python – Python 2 versus Python 3

- As versões do Python são numeradas como x.y.z (ex.: Python 3.8.1). Neste esquema de numeração, tem-se que:
 - “x” representa o número principal da versão, incrementado apenas quando ocorrem mudanças extremamente significativas na linguagem.
 - “y” é o número secundário, incrementado quando ocorrem mudanças menos significativas.
 - “z” é o nível micro, normalmente incrementado quando ocorrem *bugfixes* (correções de problemas).
- No ano de 2008, a *Python Software Foundation* decidiu promover uma mudança radical na linguagem, lançando a série de versões Python 3. Esta nova série não é completamente compatível com a série Python 2, o que significa que muitos programas implementados no Python versão 2.a.b podem não rodar no Python versão 3.c.d.
- Sendo assim, quando você baixar uma distribuição, deverá sempre optar pelo instalador referente ao Python 3. Não vale mais a pena trabalhar com o Python 2, visto que esta versão é considerada obsoleta e deverá ser descontinuada ao longo de 2020 (ao menos, esta é a previsão dada pela *Python Software Foundation*).



Lição 3 – Instalação do WinPython

A seguir são apresentados os passos para a instalação do WinPython em uma máquina com sistema operacional Windows 7 ou superior.

3.1 Instalação em sistemas Windows de 64 bits:

- **3.1.1 Acessar o repositório do WinPython:** <https://sourceforge.net/projects/winpython/> (Figura 2).

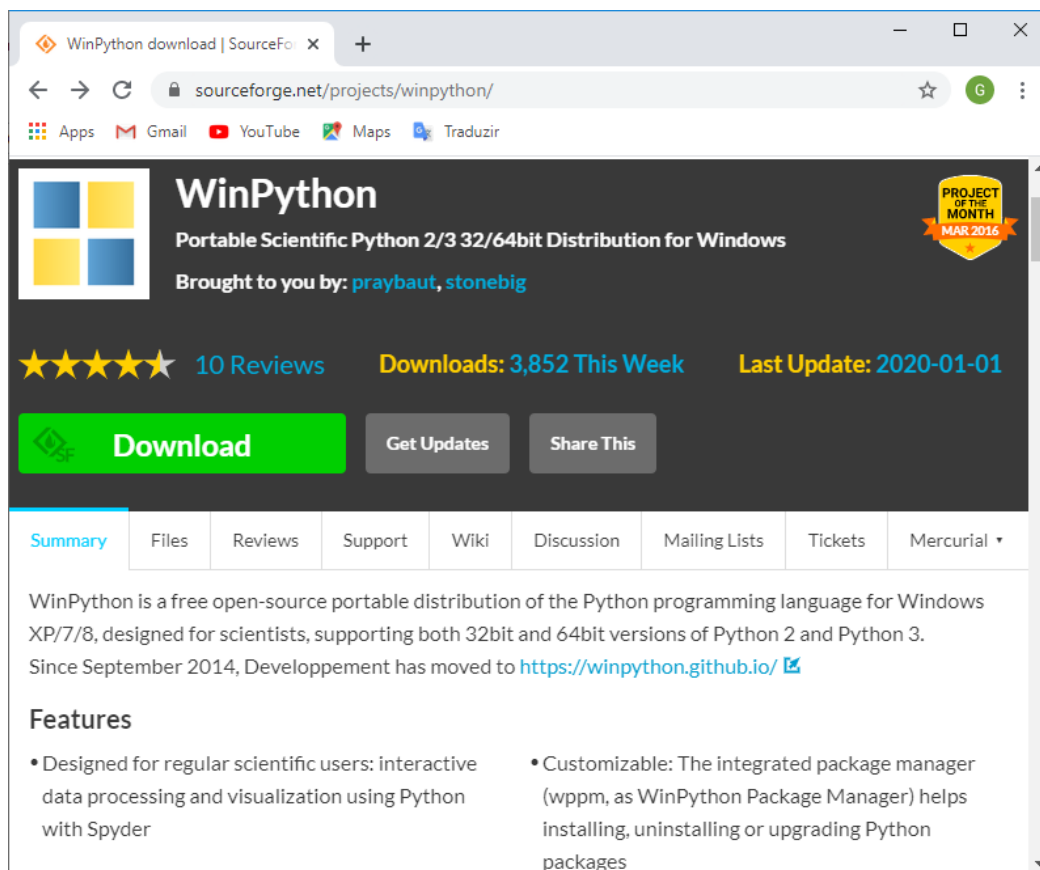


Figura 2. Web site para download do WinPython

- **3.1.2 Efetuar o download do arquivo de instalação (versão 64 bits):** caso o seu sistema operacional seja de **64 bits**, clique no botão **Download**. Após, alguns segundos, o arquivo de instalação começará a ser automaticamente baixado. No momento da elaboração deste texto, a última **versão estável** do *software* era a de número 3.7.6.0 (por isso, o arquivo tinha o nome de “WinPython64-3.7.6.0.exe”). O processo de *download* levará algum tempo até ser concluído, pois o arquivo de instalação possui mais de 648MB.
- **3.1.3 Instalar o WinPython:** ao término do *download*, execute o arquivo baixado para iniciar o processo de instalação do WinPython em sua máquina (que na verdade consiste em uma simples descompactação). A instalação é demorada, mas extremamente simples, consistindo apenas em indicar o local em que a pasta do WinPython (denominada “WPy64-3760”) será descompactada e então clicar no botão Extract. Na instalação que realizei em minha máquina, optei por trocar a sugestão inicial (pasta Downloads) para um caminho mais simples (“C:\”), pois assim o WinPython pôde ser descompactado diretamente para a pasta C:\WPy64-3760. Com relação ao espaço ocupado em disco, o WinPython é um pouco guloso, exigindo mais de 3GB de espaço livre no seu HD.

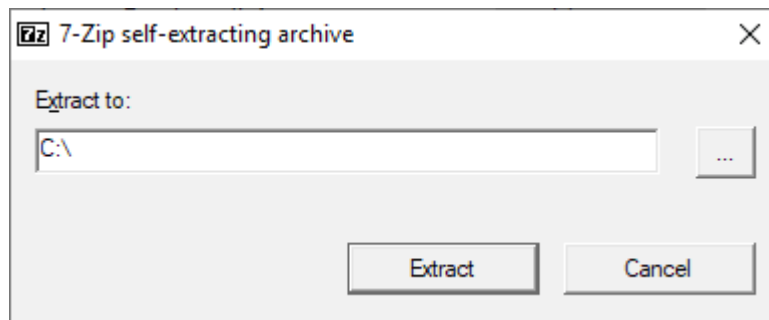


Figura 3. Instalação do WinPython: modificando a pasta destino

3.2 Instalação em sistemas Windows de 32 bits:

- **3.2.1 Navegar até a pasta “Files”:** a instalação do WinPython em sistemas de 32 bits começa um pouquinho diferente, pois, antes de tudo, é preciso navegar até a página que contém as diferentes versões da distribuição. Sendo assim, depois de acessar o repositório do WinPython (<https://sourceforge.net/projects/winpython/>), você deverá clicar em **Files**, conforme destacado na Figura 4 (não é para clicar no botão Download, mas sim em Files!!!).

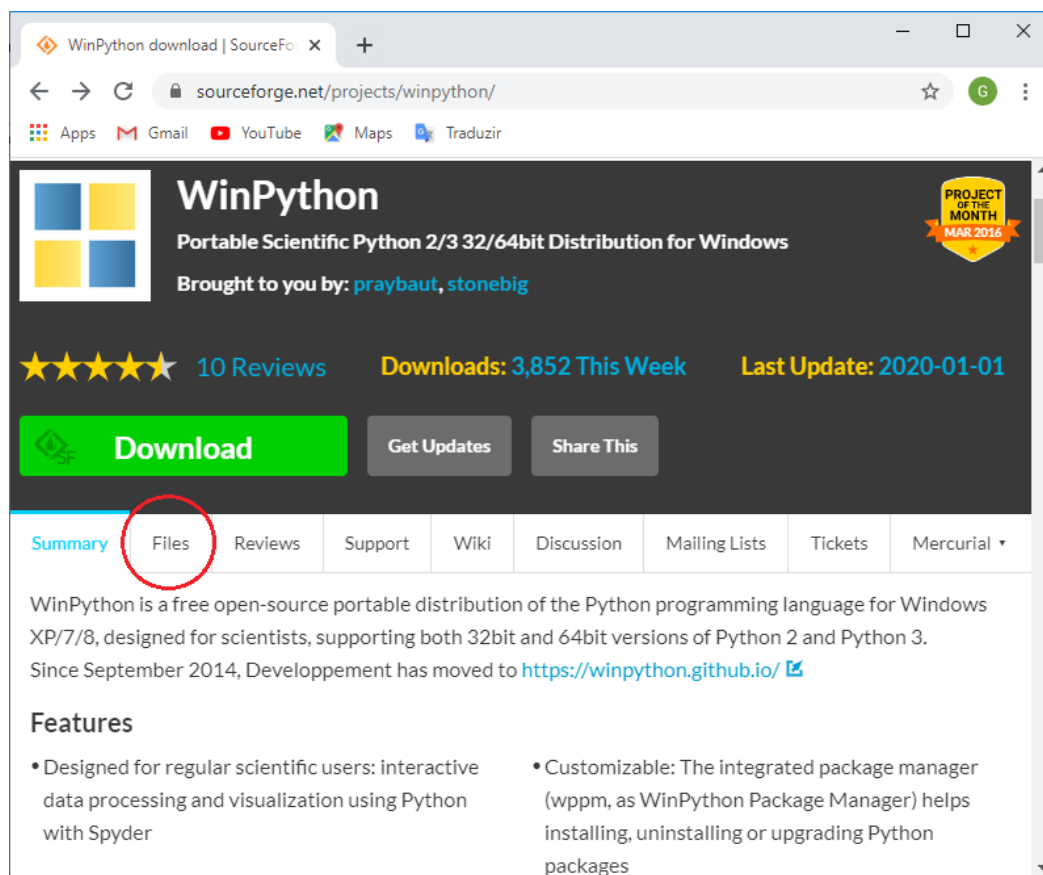


Figura 4. Download do instalador do WinPython de 32 bits (parte 1)

- **3.2.2 Efetuar o download do arquivo de instalação (versão 32 bits):** aparecerá a tela da Figura 5, onde as diversas versões da distribuição são listadas, cada qual com o seu link para *download*. Observe que há um botão verde informando que a última **versão estável** é

“WinPython64-3.7.6.0” (o botão em que está escrito “Download Latest Version”). **Não** clique nesse botão, pois ele acarretará no download da versão de 64 bits. Em vez disso, clique no link **WinPython_3.7** (destacado na Figura 5) para que seja possível acessar a versão 32 bits do WinPython 3.7.6.0. Na janela que será aberta, clique então em **3.7.6.0**. Por fim, na nova página que abriu, role a tela para baixo e clique no arquivo “WinPython32-3.7.6.0.exe”. Pronto! O arquivo começará a ser baixado.

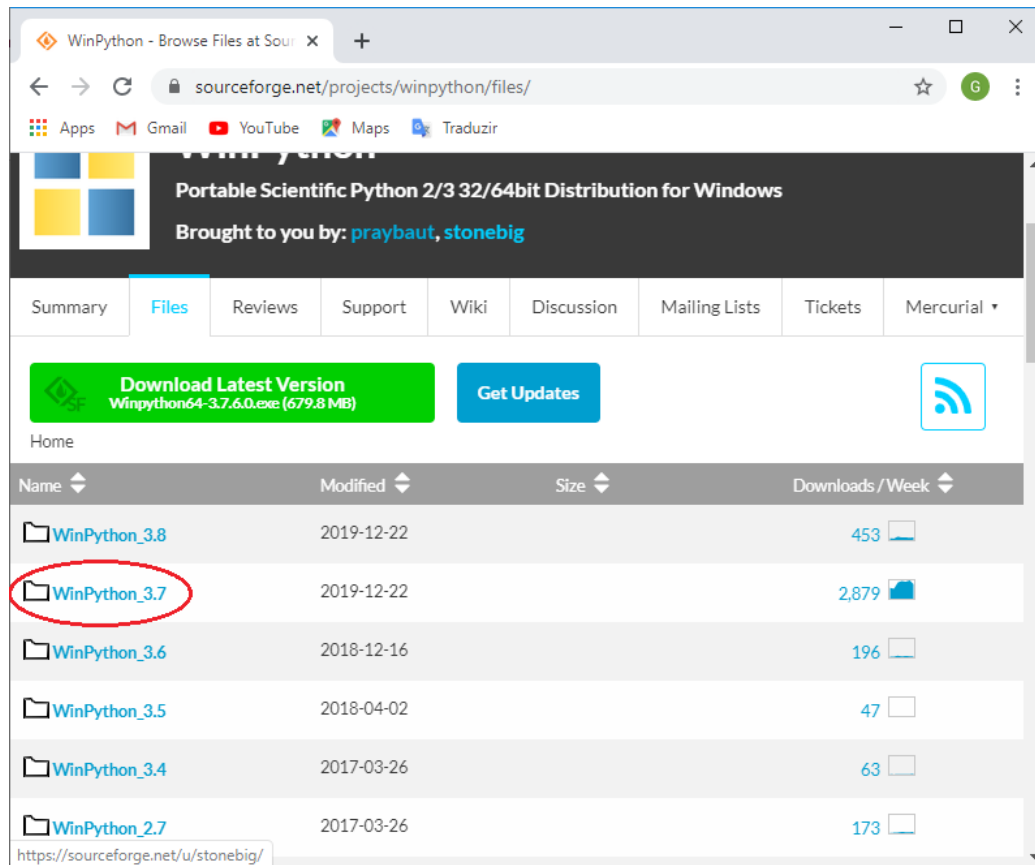


Figura 5. Download do instalador do WinPython de 32 bits (parte 2)

- **Importante:** o grande macete na instalação de 32 bits é primeiro identificar qual é o número da última versão estável (isto é feito observando-se o botão verde, onde está escrito “Download Latest Version”). Feita a identificação, você deverá clicar no link que possui o número da versão para poder ter acesso ao instalador de 32 bits.
- **3.2.3 Instalar o WinPython:** Terminado o *download*, basta seguir as orientações apresentadas no item 3.1.3 para instalar o WinPython.



Lição 4 – Utilizando o WinPython no Modo Interativo

Os programas Python podem ser executados em dois modos: **normal** e **interativo**. No modo normal, o programa é primeiro digitado por completo, em seguida salvo e, por fim, executado “de cabo a rabo” (isto é, por inteiro, sem pausas) pelo interpretador Python. De maneira oposta, quando

estamos trabalhando no modo interativo, o Python interpreta e executa comando por comando, conforme eles vão sendo digitados. A utilização do WinPython no modo normal será tratada na próxima lição. Nesta, apresentaremos a receita para trabalhar com o WinPython no modo interativo. Basta seguir os passos abaixo:

- **4.1 Iniciar o QtConsole:** abra o Windows Explorer e acesse a pasta que você escolheu para descompactar o WinPython. Então, efetue o duplo clique sobre o programa “**IPython Qt Console.exe**” (Figura 6). Trata-se de uma versão um pouco mais “simpática” do IPython, o mais conhecido terminal interativo do Python.

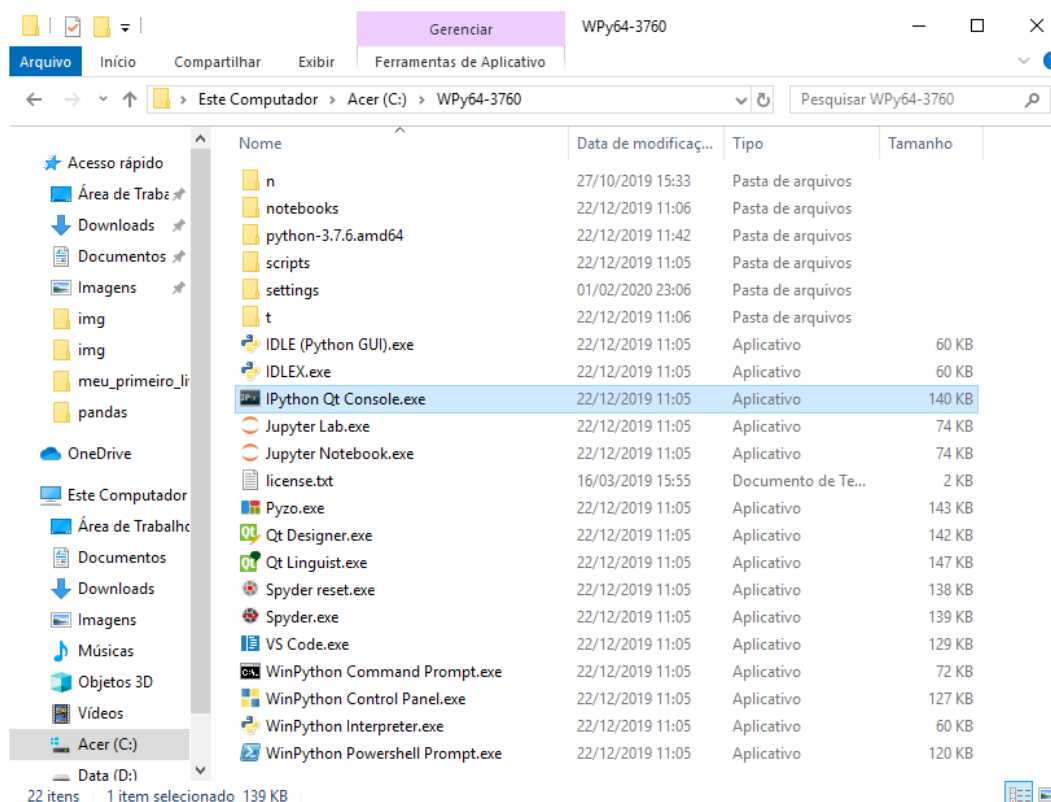


Figura 6. Localizando o aplicativo “QtConsole”

- Uma tela similar à mostrada na Figura 7 será aberta.

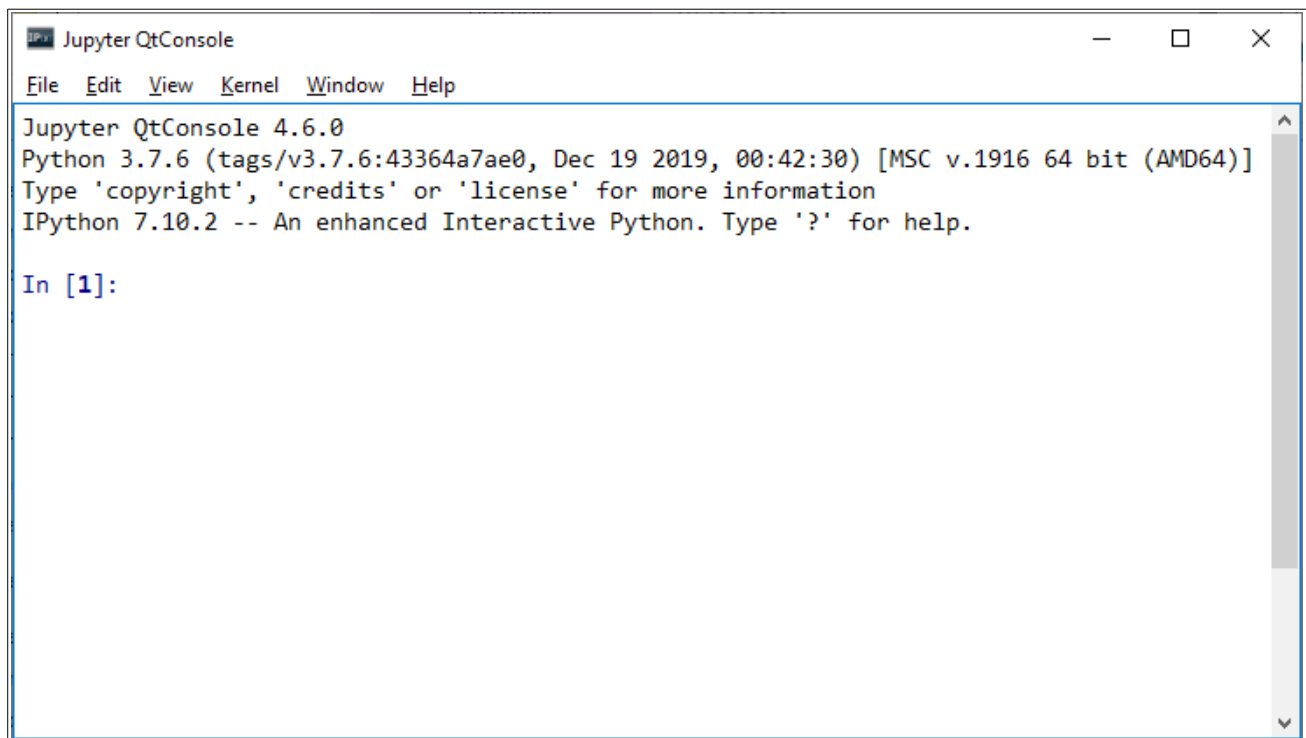


Figura 7. Tela inicial do QtConsole

- **4.2 Digitando o seu primeiro comando:** podemos testar o WinPython digitando simplesmente $1 + 1$. Você vai digitar na parte da tela que exibe “In [1]:” (chamada de *prompt* de entrada). Basta fazer como indicado abaixo:

```
In [1]: 1 + 1
Out[1]: 2
In [2]:
```

- O valor 2 será retornado após a indicação “Out[1]:” e, logo abaixo, vai aparecer um novo *prompt* para entrada de dados, desta vez representado por “In [2]:”. O modo interativo funciona sempre desta maneira: digitamos uma linha de comando e o Python a processa imediatamente, ficando logo em seguida pronto para que possamos digitar um outro comando.
- **4.3 Digitando mais um comando:** dessa vez, podemos arriscar uma coisa mais clássica: o comando para escrever “Olá Python!” na tela. Basta utilizar a função `print()` e escrever a mensagem entre aspas simples, dentro de parênteses:

```
In [1]: 1 + 1
Out [1]: 2

In [2]: print('Olá Python!')
Olá Python!

In [3]:
```

- **4.4 Encerrando a conversa com o Python:** quando quiser fechar o QtConsole digite `quit()` ou simplesmente feche a janela do aplicativo.

```
In [1]: 1 + 1
Out [1]: 2

In [2]: print('Olá Python!')
Olá Python!

In [3]: quit()
```



Lição 5 – Utilizando o WinPython no Modo Normal

Ambiente integrado de desenvolvimento (*Integrated Development Environment* – IDE) é um jargão utilizado na área de engenharia de software para rotular aplicativos destinados a servir como ambientes para a criação, depuração e execução de programas em uma determinada linguagem. A grande vantagem de utilizar uma IDE é que normalmente este tipo de software oferece uma série de recursos capazes de aumentar a eficiência do processo de programação. Alguns exemplos: **autocomplete** (possibilita com que um comando seja automaticamente completado durante a digitação), **inline help** (exibe um texto de ajuda sobre uma palavra reservada quando o mouse é colocado sobre a mesma) e **formatação automática** de programas (realiza o alinhamento automático de comandos que estejam colocados entre o início e o fim de um bloco de instruções). Apenas para apresentar alguns exemplos bem conhecidos de IDEs, podemos citar o aplicativo Eclipse, para a criação de programas na linguagem Java, e o RStudio, para o desenvolvimento em R.

Existem muitas e muitas IDEs disponíveis para Python, como PyCharm, Rodeo, Jupyter Notebook e Spyder. Na Internet, você encontrará bons artigos que discutem as vantagens e desvantagens de cada uma delas. Neste texto, adotaremos a **Spyder**, uma vez que esta IDE é otimizada para projetos de ciência de dados, além de ser muito intuitiva e permitir o uso do Python em ambos os modos, normal e interativo. Para quem está acostumado a programar em R, eu devo admitir que a Spyder não é tão bacana quanto o RStudio, além de ser um software um pouco pesado, que exige uma quantidade considerável de memória para ser executado. Mas, de qualquer forma, é bem mais agradável do que o QtConsole. O exemplo a seguir mostra o passo a passo para você utilizar o Python no modo normal por meio da Spyder.

- **5.1 Iniciar a IDE Spyder:** abra o Windows Explorer e acesse a pasta que você escolheu para descompactar o WinPython. Em seguida, efetue o duplo-clique sobre o programa “Spyder.exe” (Figura 8). Após algum tempinho, a tela principal da Spyder será mostrada (Figura 9).

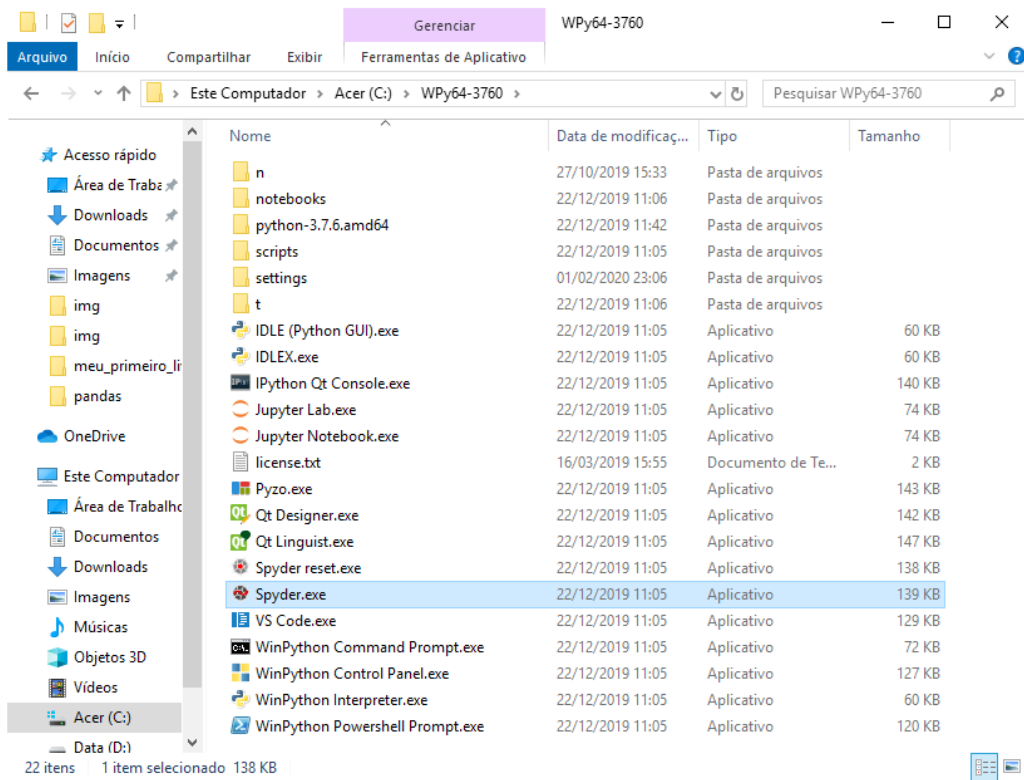


Figura 8. Localizando o aplicativo "Spyder"

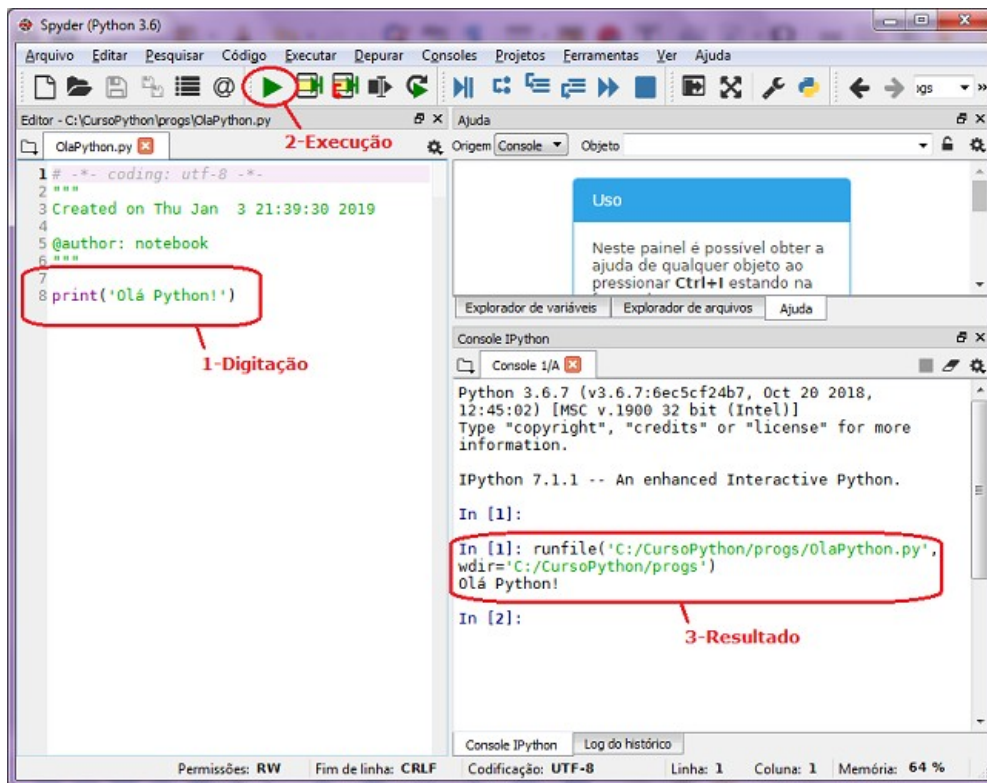


Figura 9. Tela principal do aplicativo “Spyder”. Possui três áreas principais: (1)-Editor de Código, onde os programas são digitados; (2) – Menu, com destaque para o botão “Run”; (3) – Console IPython, onde as saídas são exibidas

- **5.2 Digitar o programa:** digite na área do editor de código um programa com uma única linha: `print('Olá Python!')`. Não se assuste com o fato de a Spyder já colocar de forma automática algumas linhas de cabeçalho (linhas compreendidas entre `#-*- coding: utf-8 -*-` e `"""`). Basta digitar o seu programa logo abaixo das mesmas, a partir da linha 8.
- **5.3 Salvar o programa:** o programa está digitado, mas ainda não foi salvo. Há duas formas de fazer isto: com o uso do menu **Arquivo > Salvar** ou digitando **CTRL + S**. Em seguida, escolha uma pasta e dê um nome para o programa. No exemplo mostrado na Figura 9, a pasta escolhida foi “C:\CursoPython” e o nome do programa “OlaPython.py”. A extensão “.py” foi utilizada porque é a extensão padrão de programas Python (assim como “.java” é usada por programas Java, “.c” por programas C, etc.).
- **5.4 Executar o programa:** existem algumas formas distintas:
 - Selecione a opção de menu **Executar > Executar**; ou
 - Clique no botão “executar arquivo” (destacado na Figura 9);
 - Simplesmente pressione F5.
- **5.5 Verificar os resultados:** a saída do programa (resultado da execução) será apresentada no **Console IPython**, que consiste na área localizada no canto inferior direito da tela (indicada por “3-Resultados” na Figura 9). Este console também permite com que você utilize o Python no modo interativo.



Lição 6 – Programação Python: Primeiros Passos

6.1 Variáveis e Tipos

Uma variável pode ser entendida como o nome de um local onde se pode colocar qualquer valor numérico ou não-numérico. O código a seguir exemplifica o uso de variáveis em programas Python e apresenta os tipos de dado básicos oferecidos pela linguagem. Neste exemplo tanto como nos demais apresentados ao longo no livro, mostra-se primeiro o código do programa e, logo abaixo, o resultado da execução. O resultado estará sempre listado após o prompt “Saída [n]:”, onde *n* é o número do programa.

Programa 1 – Utilização de variáveis para armazenar informações de uma pessoa.

```
#P001: variáveis e tipos; funções type() e print()

#PARTE 1: declaração de variáveis
nome = 'Jane'
sobrenome = "Austen"
idade = 41
nota = 9.9
aprovado = True

#PARTE 2: imprimindo o conteúdo das variáveis e os tipos das mesmas
print(nome, sobrenome, idade, nota, aprovado)
print(type(nome))
print(type(sobrenome))
print(type(idade))
print(type(nota))
print(type(aprovado))

#PARTE 3: mudando o valor e o tipo da variável "nota"
nota = 'A'
print('mudei o valor e o tipo de "nota" para: ', nota, ", ", type(nota))
```

Saída [1]:

```
Jane Austen 41 9.9 True
<class 'str'>
<class 'str'>
<class 'int'>
<class 'float'>
<class 'bool'>
mudei o valor e o tipo de "nota" para: A , <class 'str'>
```

Para começar, é importante mencionar que este é um típico exemplo de programa procedural, cujo **fluxo de execução** (ordem em que os comandos são executados pelo interpretador Python) é de cima para baixo, em sequência. O programa foi dividido em três partes. Vamos às explicações sobre cada uma:

- Na PARTE 1, encontram-se as declarações das variáveis “nome”, “sobrenome”, “idade”, “nota” e “aprovado”. Estas variáveis são declaradas, respectivamente, com valores do tipo **str** (abreviação de **string**, isto é, alfanumérica, tanto “nome” quanto “sobrenome”), **int** (valores

inteiros), **float** (valores reais) e **bool** (valores lógicos: `True` ou `False`). Estes são os quatro **tipos de dados primitivos** do Python. O nome “primitivo” refere-se ao fato de que estes são os tipos de valores mais simples com os quais a linguagem trabalha.

- O **operador de atribuição**, representado pelo símbolo “=”, é usado para atribuir um valor (lado direito da equação) a uma variável (lado esquerdo). Por exemplo: `nota = 9.9`. O valor associado não precisa ser sempre fixo; ele pode também ser outra variável (por exemplo, `a = b`) ou uma expressão (`perimetro_do_quadrado = 4 * lado`).
- Em Python, a declaração de variáveis pode ser realizada em qualquer linha do programa. O tipo da variável será automaticamente determinado de acordo com o valor que lhe for atribuído, podendo mudar durante a execução do programa caso o valor da variável seja alterado.
- Os valores de strings podem ser definidos entre aspas simples (`nome = 'Jane'`) ou aspas duplas (`sobrenome = "Austen"`).
- A PARTE 2 cobre o uso de **`print()`**, a função padrão para saída de dados. Para imprimir diversas variáveis, basta separá-las por vírgula. Observe que valores de qualquer tipo podem ser impressos com esse comando. Ainda nesta parte, utilizamos a função **`type()`**, uma função muito útil que retorna o tipo de dado armazenado em uma variável.
- Na PARTE 3, mostra-se como uma variável pode ter não apenas o seu valor, mas também seu tipo modificado durante o fluxo de execução. Neste trecho de código, o valor de “nota” é mudado de 9.9 (tipo float) para ‘A’ (tipo str).

6.2 Comentários

Comentários são textos ou frases definidos com o uso do símbolo `#`. No programa anterior, observe que a primeira linha contém um texto com comentários:

```
#P001: variáveis e tipos; funções type() e print()
```

Os comentários são **ignorados** pelo interpretador Python na hora em que o programa é processado. Eles servem apenas para **documentação**, isto é, para serem lidos pelos humanos que estiverem analisando ou alterando o programa. Na prática, são muito úteis para que o entendimento de uma **rotina** (conjunto de instruções) complexa possa ser facilitado.

Nomes de Variáveis

- Como foi introduzido na Lição 1, Python é uma linguagem que prioriza a legibilidade do código. Por este motivo, os *pythonistas* são incentivados a escolher nomes que consigam expressar com clareza o significado das variáveis (mesmo que eles fiquem grandes). Existe até mesmo um guia de estilo denominado PEP-8, responsável por oferecer uma série de regras que o programador pode aplicar para deixar o seu código mais legível. Este guia recomenda que todas as variáveis devam ter o nome especificado em minúsculo e que o caractere *underscore* (`_`) seja utilizado para separar nomes compostos. Alguns exemplos:
 - `idade`
 - `renda_media_anual`
 - `codigo_ocupacao`

- produtos_2019

- Os nomes podem conter letras, números e o caractere *underscore*, no entanto não podem ser iniciados por um número. Há diferenciação entre letras maiúsculas e minúsculas, o que significa que se você nomear uma variável como “x”, **não** poderá referenciá-la como “X” (mas lembre-se de que a recomendação é usar minúsculo nos nomes de todas as variáveis).
- O nome de uma variável também não pode ser igual ao de uma **palavra reservada** (*keyword*) da linguagem Python, cuja lista completa é apresentada abaixo:

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

6.3 Expressões

O programa a seguir cobre as operações matemáticas básicas. Os operadores +, -, *, /, e ** são usados, respectivamente, para realizar a adição, subtração, multiplicação, divisão e exponenciação. O operador // retorna o quociente de uma divisão inteira, enquanto % retorna o módulo (resto). Qualquer operação segue a ordem básica conhecida como PEMDAS (parênteses, exponenciação, multiplicação, divisão, adição e subtração).

Programa 2 – Operações matemáticas básicas.

```
#P002: matemática básica

#Adição, subtração, multiplicação e divisão
x=5; y=2
print(x+y, x-y, x*y, x/y)
print(y**x)

#quociente e resto (ou módulo)
quociente = x // y
modulo = x % y
print("O quociente da divisão de", x, "por", y, "é: ", quociente)
print("O módulo da divisão de", x, "por", y, "é: ", modulo)

#expressão com parênteses
minha_expressao = (1 + 2) * 5**2 / ((5-3) + 1)
print("O valor da expressão 'minha_expressao' é: ", minha_expressao)

#a divisão de dois inteiros sempre gera um float
#(mesmo que a divisão seja exata)
a=10; b=5; c=a/b
print(a, b, c)
print(type(a), type(b), type(c))
```


Saída [2]:

```
7 3 10 2.5
32
O quociente da divisão de 5 por 2 é: 2
O módulo da divisão de 5 por 2 é: 1
O valor da expressão 'minha_expressao' é: 25.0
10 5 2.0
<class 'int'> <class 'int'> <class 'float'>
```

Duas observações importantes:

- É possível escrever mais de um comando em uma única linha, bastando separá-los por ponto e vírgula. Por exemplo: `a=10; b=5; c=a/b`. Entretanto, de acordo com as “boas práticas de programação” (PEP-8), você deve evitar produzir uma linha que possua mais de 72 caracteres.
- Conforme mostrado nas três últimas linhas do programa, no Python 3, a divisão de dois inteiros sempre resultará em um valor do tipo real (float).

6.4 Entrada de Dados via Teclado

A linguagem Python possui uma função chamada `input()` que possibilita a entrada de dados via teclado.

Programa 3 – Recebe o nome de uma pessoa via teclado.

```
#P003: Entrada de dados
nome = input("Qual o seu nome?\n")
print("Hmmm... então você é o famoso " + nome)
```

Saída [3]:

```
Qual é o seu nome?
> Nelson Cavaquinho
Hmmm... então você é o famoso Nelson Cavaquinho
```

Explicando:

- Quando a função `input()` é chamada, o programa “dá uma paradinha” e fica esperando o usuário digitar alguma coisa. Após o usuário terminar de digitar e teclar Enter, a função retornará o valor introduzido via teclado como uma string. Este valor poderá então ser armazenado em uma variável. Em nosso exemplo, foi armazenado na variável “nome”.
- Na primeira linha do programa, a estranha sequência “`\n`” foi utilizada para forçar uma quebra de linha após a pergunta “Qual é o seu nome?”. Essa sequência também é comumente utilizada dentro da função `print()`. Na realidade, ela é muito empregada em programas escritos em diferentes linguagens de programação, sendo intitulada “formato *newline*”.
- Na segunda linha do programa, o operador `+` é empregado para concatenar strings, ou seja, para juntar a frase “Hmmm... então você é o famoso” com o valor da variável “nome”.

- Na ciência de dados, os programas costumam receber como entrada apenas arquivos ou bases de dados e, por isso, a função `input()` praticamente não é utilizada.



Lição 7 – Instruções de Desvio Condicional: *if*, *else*, *elif*

Em qualquer *script* de análise de dados, sempre ocorrerão situações em que será necessário analisar condições para, só então, definir o comportamento do programa. As instruções `if` e `else` são as utilizadas para esta finalidade. Elas são chamadas de instruções de **desvio condicional**.

Programa 4 – Desvio condicional com `if`: - `else`: Neste exemplo, a estrutura `if-else` é empregada para verificar a idade de uma pessoa que deseja entrar em uma festa realizando uma comparação com o valor inteiro 18 (a idade da pessoa está armazenada na variável “idade”). Se (`if`) a idade for maior ou igual a 18, então uma mensagem que a convida a entrar é exibida na tela. Senão (`else`), uma mensagem informando que o cidadão ou cidadã está devidamente barrado(a) é exibida na tela (não adianta nem a pessoa argumentar que tem 17 anos e 11 meses, pois “idade” é do tipo inteiro!!!). Se você tiver alguma dúvida, teste o programa diferentes vezes modificando o valor da variável “idade”.

```
#P004: desvio condicional com if: - else:
idade = 17
if (idade >= 18) :
    print("Pode entrar, a festa está bombando!")
    print("Temos muita música e drinks especiais!!!")
else :
    print("Você é jovem demais para este clube! Volte apenas quando fizer 18.")
```

Saída [4]:

Você é jovem demais para este clube! Volte apenas quando fizer 18.

Observe que os comandos `if` e `else` terminam com um sinal de dois pontos “:” e que as **linhas subordinadas** a cada comando estão **indentadas**. Neste exemplo, o `if` tem dois comandos subordinados e o `else` apenas um. No jargão da computação, um conjunto de comandos subordinados é chamado de **bloco de código**. A seção a seguir descreve em detalhes o esquema utilizado pelo Python para a definição de blocos de código.

7.1 Blocos de Código

Para começar, um aviso: preste atenção, muita atenção mesmo, pois será apresentada uma característica muito peculiar do Python. Ao contrário do que ocorre em qualquer outra linguagem de programação, os blocos de código que estão subordinados ao `if`, `else` e a outros comandos que ainda serão apresentados, como `for` e `while`, são definidos através da **indentação de comandos**⁴. Enquanto o Pascal usa as palavras *begin* e *end* para marcar um bloco e linguagens como C, Java e R utilizam os símbolos “{” e “}”, o Python utiliza espaços em branco ou tabulações. É isso mesmo:

4 A palavra “indentação” é muito feia, mas infelizmente existe!

espaços em branco ou tabulações. Isto significa que os comandos precisam estar alinhados da mesma forma para que o Python os reconheça como parte de um bloco. A convenção é utilizar 4 espaços para indentação⁵ (esta é mais uma recomendação da PEP-8). Entretanto, a linguagem permite qualquer padrão de alinhamento (por exemplo, um tab, três espaços, etc.), contanto que este seja repetido para todos os comandos pertencentes a um mesmo bloco. Para que o conceito fique claro, observe o exemplo abaixo:

Erro de indentação: <pre>if (x > 0) : a=1</pre> <p><code>IndentationError: expected an indented block</code></p>	Indentação correta: <pre>if (x > 0) : a=1</pre>
---	--

Cuidado com o Copiar e Colar!

- Se você tentar selecionar e copiar o código dos programas a partir deste documento PDF, perceberá que eles perderão a indentação ao serem colados em seu destino. Desta forma, o ideal é que você digite os exemplos (ou sempre corrija a indentação depois de colar).

7.2 Operadores Relacionais e Lógicos

A instrução **if** toma uma decisão a partir da avaliação de uma **condição**. Uma condição representa uma comparação ou um conjunto de comparações que irá resultar sempre em um valor do tipo lógico (boolean), isto é, VERDADEIRO (`True`) ou FALSO (`False`). Na linguagem Python, os operadores utilizados nas condições – denominados **operadores relacionais** – são os seguintes:

- `x == y` *# o valor de x é igual ao de y?*
- `x != y` *# o valor de x é diferente do valor de y?*
- `x > y` *# o valor de x é maior que o de y?*
- `x < y` *# o valor de x é menor que o de y?*
- `x >= y` *# o valor de x é maior ou igual ao de y?*
- `x <= y` *# o valor de x é menor ou igual ao de y?*

Os operadores lógicos **and** (e), **or** (ou) e **not** (não) podem ser utilizados para unir diferentes condições, aumentando assim, a complexidade dos testes. Eles funcionam da seguinte maneira:

- `and` *# a sentença é verdadeira se TODAS as condições forem verdadeiras*
- `or` *# a sentença é verdadeira se UMA das condições for verdadeira*
- `not` *# inverte o valor lógico de uma sentença (True → False, False → True)*

Programa 5 – Miscelânea de exemplos envolvendo operadores relacionais e lógicos.

⁵ <https://stackoverflow.blog/2017/06/15/developers-use-spaces-make-money-use-tabs/>

```

#P005: operadores relacionais e lógicos
print('* * * * parte 1 * * * * ')
print(4 * 2 == 8) #True.
print(9 ** 2 == 81) #True.
print(5 + 2 < 7) #False.
print(5 + 2 >= 7) #True.
print('Portela' == 'Mangueira') #False.
print(5 / 2 > 3) #False.
print(7 % 2 != 0) #True.

print('\n\n* * * * parte 2 * * * * ')
pais = 'Brasil'
print(pais == 'Brasil') #True.
print(pais == 'BRASIL') #False.

media_final = 7.0
if (media_final >= 7.0):
    print('com a média', media_final, 'você está aprovado')
else:
    print('reprovado')

a=10; b=100
if (a >= b):
    print('o valor de "a" é maior ou igual ao de "b"')
else:
    print('o valor de "a" é menor do que o de "b"')

print('\n\n* * * * parte 3 * * * * ')
print((4 * 2 == 8) and (9**2 >= 81)) #True.
print((1 + 1 < 3) and (9**2 > 81)) #False.
print((1 + 1 < 3) or ('cruzeiro' == 'atletico')) #True.

```

Saída [5]:

* * * * parte 1 * * * *

True
True
False
True
False
False
True

* * * * parte 2 * * * *

True
False
com a média 7.0 você está aprovado
o valor de "a" é menor do que o de "b"

* * * * parte 3 * * * *

True
False
True

Em situações onde mais de dois operadores lógicos tenham que ser utilizados para a montagem de um teste lógico, a ordem de precedência apresentada no Quadro 1 será adotada.

Quadro 1. Regras de precedência para operadores lógicos

Operador	Ordem de avaliação
not	1
and	2
or	3

O operador **not** tem maior prioridade (é o primeiro a ser avaliado), seguido do **and** e do **or**, respectivamente. Estas regras de precedência podem ser sobrepostas através do uso de parênteses. Para que o conceito fique claro, considere o cenário descrito a seguir. Suponha que você está desenvolvendo um programa para processar uma base de dados de produtos e que, nesta base, existam cinco diferentes categorias de produtos: 'A', 'B', 'C', 'D' e 'E'. Imagine que você deseja produzir um relatório que listará apenas os produtos das categorias 'A' e 'B' que custem menos de R\$ 500,00. Em uma situação como esta, caso você monte a sua estrutura de desvio da forma apresentada abaixo, acabará por selecionar produtos que não deveriam pertencer à listagem:

```
preco = 999.99; categoria = 'B'

if categoria=='B' or categoria=='A' and preco < 500:
    print('selecionado')
else:
    print('não selecionado')
```

Saída [1]:
selecionado

Neste exemplo, temos um produto da categoria 'B' com preço igual a R\$ 999,99, ou seja, acima de R\$ 500,00. Ele não deveria ser selecionado, mas a instrução **if** possui um erro de lógica: como o operador **and** tem precedência sobre o operador **or**, o Python testa a condição especificada em negrito primeiro. Ou seja, o teste será interpretado pelo Python da seguinte maneira: “selecione o produto caso a categoria seja igual a 'A' e o preço for menor do que 500; ou se a categoria for 'B'”. Observe agora a correção do programa, onde os parênteses são empregados para alterar a precedência do teste e, desta forma, pegar apenas os produtos corretos:

```
preco = 999.99; categoria = 'B'

if (categoria=='B' or categoria=='A') and preco < 500:
    print('selecionado')
else:
    print('não selecionado')
```

Saída [1]:
não selecionado

7.3 Instrução **elif**

Em muitas situações práticas, existe a necessidade de avaliar mais de duas possibilidades em um teste condicional. Nesta situação, os testes poderão ser logicamente agrupados com o uso da instrução **elif** (não é “else if” e tampouco “elsif”, o nome da instrução é “elif”). No Programa 6, a estrutura if-elif-else é utilizada para determinar a categoria de um valor numérico.

Programa 6 – Retorna a faixa etária de uma pessoa em função de sua idade.

```
#P006: if, elif, else
idade = 25

if (idade < 18):
    faixa_etaria = '<18'
elif (idade >= 18 and idade < 30):
    faixa_etaria = '18-29'
elif (idade >= 30 and idade < 40):
    faixa_etaria = '30-39'
else:
    faixa_etaria = '>=40'

print('Se a idade é', idade, 'então a faixa etária é :', faixa_etaria)
```

Saída [6]:

Se a idade é 25 então a faixa etária é : 18-29

Python **não** possui uma estrutura do tipo case ou switch presente em outras linguagens de programação. Ou seja: o que você implementaria com case-switch em outra linguagem, você fará com if-elif-else em Python.

7.4 Instruções Condicionais Aninhadas (*Nested Conditionals*)

Para encerrar a lição, a seguir apresentamos um exemplo de código que contém um if dentro de um else. Para implementar uma rotina deste tipo – if dentro de else, if dentro de if, ou qualquer coisa parecida – também é preciso fazer uso da indentação.

Programa 7 – Dados dois números, verifica qual é o maior ou se ambos são iguais.

```
#P007: instruções condicionais aninhadas
a=5; b=10
if (a == b):
    print("a e b são iguais")
else:
    if (a > b):
        print("a é maior do que b")
    else:
        print("a é menor do que b")
```

Saída [7]:

a é menor do que b

O Quadro 2 resume a sintaxe dos comandos de desvio condicional.

Quadro 2. Sintaxe da estruturas: (1) if (sozinho); (2) if – else; (3) if – elif – else;

- **if sozinho (podendo ter um ou mais comandos subordinados)**

```
if <condição> :
    comando1
    ...
    comandon
```

- **if com else (ambos podendo ter um ou mais comandos subordinados)**

```
if <condição> :
    comando1
    ...
    comandon
else :
    comando1
    ...
comandon
```

- **if + elif + else (cada qual podendo ter um ou mais comandos subordinados)**

```
if <condição1> :
    comando1
    ...
    comandon

elif <condição2> :
    comando1
    ...
    comandon
...
elif <condiçãom> :
    comando1
    ...
    comandon
else :
    comando1
    ...
    comandon
```



Lição 8 – Instruções de Repetição (1): while

Uma estrutura de **repetição** (ou **laço**) permite com que um bloco de instruções possa ser executado diversas vezes até que uma condição seja satisfeita. Na linguagem Python, há dois tipos de estrutura de repetição: **while** e **for**.

O comando **while** funciona da mesma forma que em outras linguagens de programação: enquanto o valor da condição especificada ao lado da palavra **while** for verdadeira, o bloco de código subordinado ao comando é executado. Quando for falso, o comando é abandonado. Se no primeiro teste o resultado é falso, os comandos não são executados nenhuma vez. A sintaxe do comando é apresentada no Quadro 3.

Quadro 3. Sintaxe do comando while

```
while <condição> :  
    comando1  
    ...  
    comandon
```

Programa 8 – Tabela de equivalência entre graus Celsius e graus Fahrenheit. No programa a seguir, gera-se uma tabela em que -20°C é o valor de temperatura inicial e 100°C o final. A escala da tabela em graus Celsius varia de 10 em 10.

```
#P008: repetição com o comando while (primeiro exemplo)  
c = -20  
print('* * * Tabela de conversão de graus Celsius para graus Fahrenheit')  
while c <= 100:  
    f=c*1.8 + 32  
    print(c, '°C ----> ', f, '°F')  
    c=c+10  
  
print('\nFIM!!!')
```

Saída [8]:

```
* * * Tabela de conversão de graus Celsius para graus Fahrenheit  
-20 °C ----> -4.0 °F  
-10 °C ----> 14.0 °F  
0 °C ----> 32.0 °F  
10 °C ----> 50.0 °F  
20 °C ----> 68.0 °F  
30 °C ----> 86.0 °F  
40 °C ----> 104.0 °F  
50 °C ----> 122.0 °F  
60 °C ----> 140.0 °F  
70 °C ----> 158.0 °F  
80 °C ----> 176.0 °F  
90 °C ----> 194.0 °F  
100 °C ----> 212.0 °F
```

FIM!!!

A variável “c” armazena a temperatura em graus Celsius e também atua como **variável de controle**, pois ela é utilizada para controlar o laço, ou seja, para mantê-lo em execução durante o número de repetições desejadas. Ela é inicializada fora do laço e é atualizada dentro do mesmo, como deve ocorrer sempre que utilizarmos o **while**. Cada repetição executada dentro de um laço é chamada de **iteração**. Desta forma, o programa acima realizou 13 iterações, pois as três instruções subordinadas ao **while** foram executadas 13 vezes.

Programa 9 – Computa o valor da série $H = 1 + (1 / 2) + (1 / 3) + \dots + (1 / N)$.

```
#P009: repetição com o comando while (segundo exemplo)
N = 5
H = 1
i = 1 #variável de controle

print('* * * cálculo de H = 1 + (1 / 2) + (1 / 3) + ... + (1 / N), dado N =',N)

while (i !=N):
    i = i + 1
    H = H + (1 / i)

print('* * * resposta: H = ', H)
```

Saída [9]:

```
* * * cálculo de H = 1 + (1 / 2) + (1 / 3) + ... + (1 / N), dado N = 5
* * * resposta: H = 2.2833333333333333
```

O comando **break** pode ser utilizado **quebrar um laço “na marra”**, passando o fluxo de execução do programa para a linha que estiver localizada imediatamente depois do fim do bloco de comandos subordinados ao laço. Por sua vez, o comando **continue** serve para **quebrar uma iteração**, mas não o laço propriamente dito. Mais claramente: sempre que o **continue** é executado, o fluxo de execução do programa é automaticamente desviado para a linha que contém o comando **while**.

Programa 10 – Quebra de laço com break e quebra de iteração com continue.

```
#P010: while com break + while com continue
print('-----')
print('1-Exemplo de while com break:\n')
n=-1;
while (n < 21):
    n=n+1;
    if n%2 != 0: break #quebra o laço se n for ímpar...
    print(n)

print('fim do while com break...\n')

print('-----')
print('2-Exemplo de while com continue:\n')
n=-1;
while (n < 21):
    n=n+1;
    if n%2 != 0: continue #quebra a iteração se n for ímpar...
    print(n)

print('fim do while com continue...\n')
```

Saída [10]:

```
-----
1-Exemplo de while com break:
```

```
0
fim do while com break...
```

2-Exemplo de while com continue:

```
0
2
4
6
8
10
12
14
16
18
20
fim do while com continue...
```

Vetorização

- Os principais pacotes para ciência de dados permitem a execução de operações sobre conjuntos de dados sem a necessidade da implementação de laços. Esse processo é conhecido como **computação vetorizada** (*vectorization*), apresentado em detalhes no livro “Pandas Python: Data Wrangling para Ciência de Dados”. Por esta razão, embora o comando **while** seja largamente utilizado na programação de sistemas convencionais, ele é bem menos empregado na ciência de dados.



Lição 9 – Instruções de Repetição (2): for – range()

Assim como o **while**, o comando **for** também serve para implementar a repetição de blocos de código. Ele existe em praticamente todas as linguagens de programação e, por ser mais prático do que o **while**, é normalmente escolhido pelos programadores em situações onde deseja-se executar um conjunto de comandos por um número fixo de vezes. Na linguagem Python o **for** possui uma característica bastante singular (e relevante!): ele pode iterar apenas sobre **sequências**. Nesta lição, mostraremos como utilizar o **for** para iterar sobre sequências criadas com a função **range()**. No livro “Pandas Python: Data Wrangling para Ciência de Dados”, veremos como é possível utilizá-lo para iterar sobre sequências mais complexas, como Series e DataFrames.

Programa 11 – Receita básica para implementar laços utilizando a estrutura for-range().
Atenção: conforme mostra o programa, o limite final definido em um **range()** não fará parte da sequência gerada (comentaremos mais sobre isso no texto após o código do programa).

```
#P011: repetição com for-range()
print('\n* * imprimindo de 0 a 9')
for i in range(10):
    print(i)

print('\n* * imprimindo de 100 a 105')
for i in range(100, 106):
    print(i)

print('\n* * 0 a 15, usando 5 como incremento')
for i in range(0, 16, 5):
    print(i)

print('\n* * ordem reversa: 5, 4, 3, 2, 1')
for i in range(5, 0, -1):
    print(i)
```

Saída [11]:

```
* * imprimindo de 0 a 9
0
1
2
3
4
5
6
7
8
9

* * imprimindo de 100 a 105
100
101
102
103
104
105

* * usando 5 como incremento
0
5
10
15

* * ordem reversa
5
4
3
2
1
```

A estrutura **for-range()** executa um laço por um número fixo de vezes. Os comandos subordinados ao **for** devem estar indentados. É necessário especificar:

- Uma **variável de controle** (ou **contador**) esta é a popular “variável do **for**”, que em nosso programa foi chamada de “i” nos quatro laços implementados.
- Os **limites inferior** e **superior** e o **valor do incremento**: esses valores determinam o número de repetições que o laço conterà e são definidos com o uso da função **range()**. A explicação sobre esta função é apresentada no tópico a seguir.

9.1 Conhecendo a Função **range()**

A função **range()** serve para gerar uma sequência de números inteiros que é geralmente utilizada para ser percorrida por um comando **for**. O Quadro 4 apresenta informações detalhadas sobre a sua sintaxe e parâmetros.

Quadro 4. Sintaxe da função **range()**

- Sintaxe: **range([início], fim, [incremento])**
- Parâmetros:
 - **início**: número inicial da sequência (opcional). Caso seja omitido, o valor 0 é assumido.
 - **fim**: a sequência será gerada até, mas sem incluir, o número especificado neste parâmetro (único parâmetro obrigatório).
 - **incremento**: diferença entre cada número na sequência (opcional). Se omitido, o valor 1 será utilizado.
- Observações:
 - Todos os parâmetros devem ser do tipo inteiro;
 - Todos os parâmetros podem ser positivos ou negativos;
 - Na função **range()** e na linguagem Python de uma maneira geral, conjuntos de valores são indexados a partir do valor 0 e não do valor 1. Em outras palavras: em Python, o primeiro elemento de uma sequência, lista, vetor, etc., sempre possuirá o índice 0. É por isso que o comando **range(n)** produz uma lista cujo primeiro elemento é 0 e o último $n-1$,
- Exemplos:

◦ range(3)	# [0, 1, 2]
◦ range(1, 4)	# [1, 2, 3]
◦ range(0, 10, 2)	# [0, 2, 4, 6, 8]

Para finalizar a lição, o Quadro 5 apresenta um resumo com a "receita" para implementar os casos mais comuns de utilização da estrutura **for-range()**.

Quadro 5. Sintaxe da estrutura for-range() para as situações práticas mais comuns

- **for simples:** iteração de 0 a $n-1$ com incremento de 1

```
for (contador) in range(n):  
    comando1  
    ...  
    comandon
```

- **for básico:** iteração de 1 a n , com incremento de 1

```
for (contador) in range(1, n+1):  
    comando1  
    ...  
    comandon
```

- **for crescente:** vai de m até n (onde $n \geq m$), com incremento de k

```
for (contador) in range(m, n+1, k):  
    comando1  
    ...  
    comandon
```

- **for decrescente:** vai de n decrescendo até m (onde $n \geq m$), com decremento de k

```
for (contador) in range(n, m-1, -k):  
    comando1  
    ...  
    comandon
```

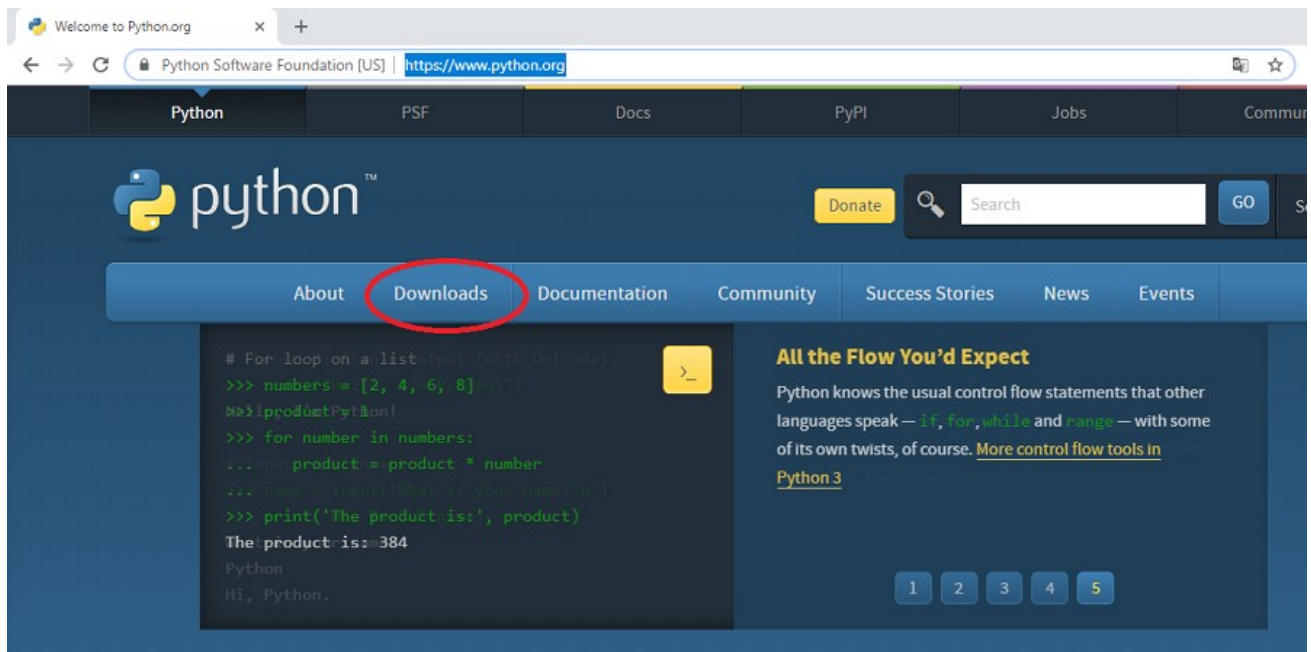
Anexo A - CPython

O texto a seguir apresenta os passos necessários para instalar e utilizar o ambiente Python padrão (conhecido como CPython), obtendo o instalador a partir do Website oficial do Python. Depois de instalar essa distribuição, você poderá instalar os pacotes de interesse separadamente, utilizando o aplicativo "pip". A distribuição oficial é bem mais leve do que a distribuição WinPython e, além disso, está disponível para qualquer sistema operacional.

A.1- Instalação do Python

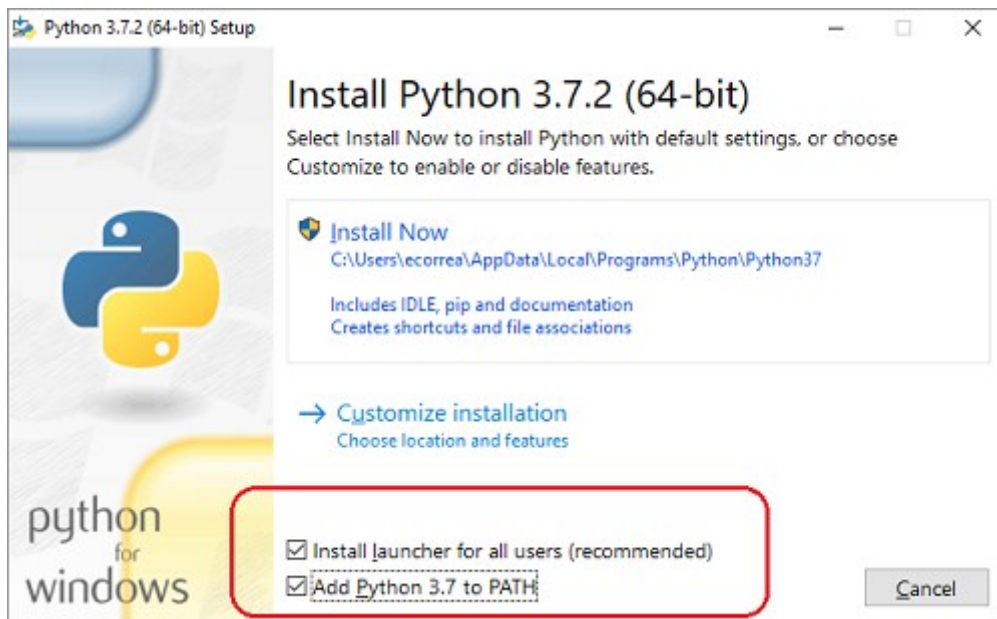
PASSO 1: DOWNLOAD

- Acesse o site <https://www.python.org/>, selecione Downloads (destacado na figura abaixo) e clique no botão referente à **versão 3** do Python que corresponda ao seu sistema operacional (Windows, Linux ou Mac).

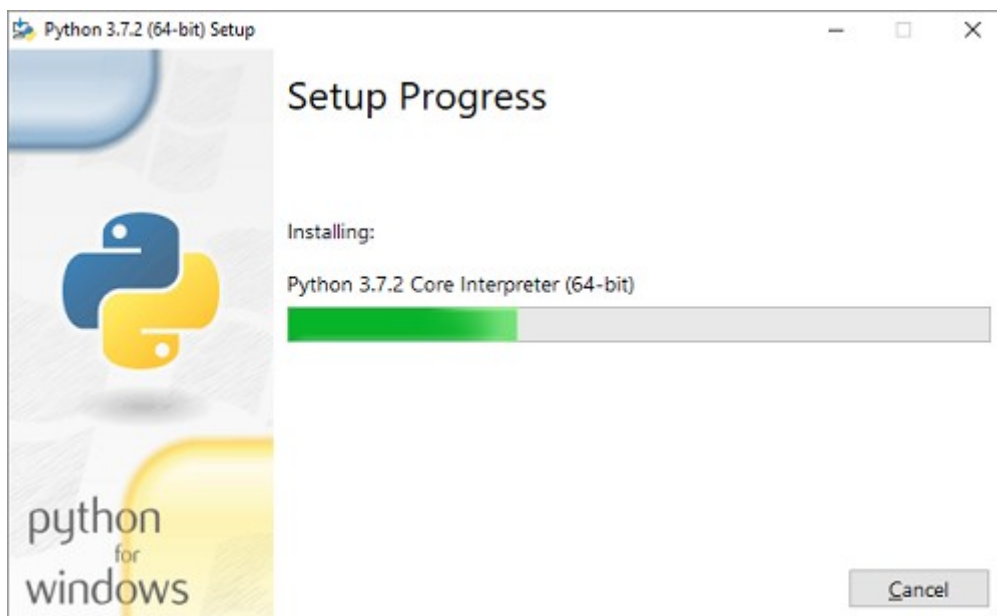


PASSO 2: INSTALAÇÃO

- Para realizar a instalação, basta executar o arquivo baixado. * * * **Importante!!!** se o seu sistema operacional for **Windows**, é interessante que você **marque as duas opções mostradas** figura da página a seguir (a opção "Add Python 3.7 to PATH" vem desmarcada, mas o ideal é que você a marque).



- A razão para a segunda opção não estar pré-selecionada por default é o fato de que muitas pessoas podem estar realizando a instalação com uma conta que não possui direito administrativo na máquina. Se você não marcá-la, o Python não será adicionado ao PATH do Windows e isto vai acabar atrapalhando a futura instalação de pacotes e a execução de programas através do prompt de comandos.
- De resto, é só prosseguir normalmente com a instalação, clicando nos botões "Next" e aguardar até o processo ser finalizado.

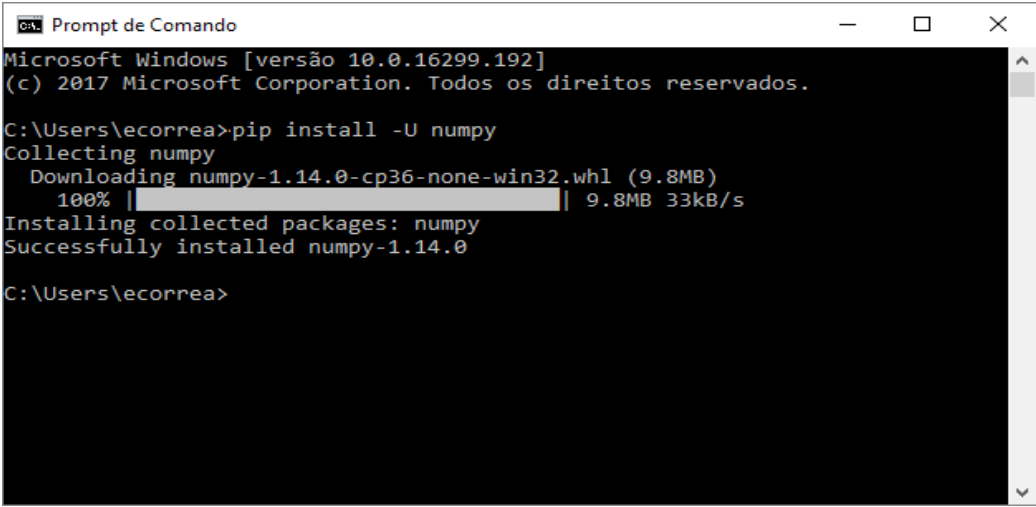


A.2- Instalação de Pacotes com o "pip"

- A forma mais utilizada para instalar pacotes é através do utilitário “pip”. Para usar este utilitário, é preciso abrir uma janela de terminal (no caso do Windows, "prompt de comandos") e digitar:

```
pip install -U nome_do_pacote
```

- O pacote especificado será então baixado e instalado. Caso ele já esteja instalado, será atualizado por uma versão mais nova (se existir). O exemplo a seguir, mostra a instalação do pacote 'Numpy':



```
cmd Prompt de Comando
Microsoft Windows [versão 10.0.16299.192]
(c) 2017 Microsoft Corporation. Todos os direitos reservados.

C:\Users\ecorrea>pip install -U numpy
Collecting numpy
  Downloading numpy-1.14.0-cp36-none-win32.whl (9.8MB)
    100% |#####| 9.8MB 33kB/s
Installing collected packages: numpy
Successfully installed numpy-1.14.0

C:\Users\ecorrea>
```

- Sendo assim, para instalar todos os pacotes que não fazem parte do Python padrão e que foram apresentados no livro “Pandas Python: Data Wrangling para Ciência de Dados”, você deve utilizar:

```
pip install -U numpy
pip install -U pandas
pip install -U scikit-learn
```

A.3- Instalação e Execução da IDE Spyder

- A seguir apresenta-se a forma de instalar e executar a IDE Spyder. A instalação também pode ser feita através do “pip” (essa instalação é bem mais demorada do que a instalação dos pacotes):

```
pip install -U spyder
```

- Após a instalação, você poderá executar o aplicativo digitando “spyder3” a partir da janela de terminal:


```
C:\Users\ecorreia>spyder3
C:\Users\ecorreia>_
```

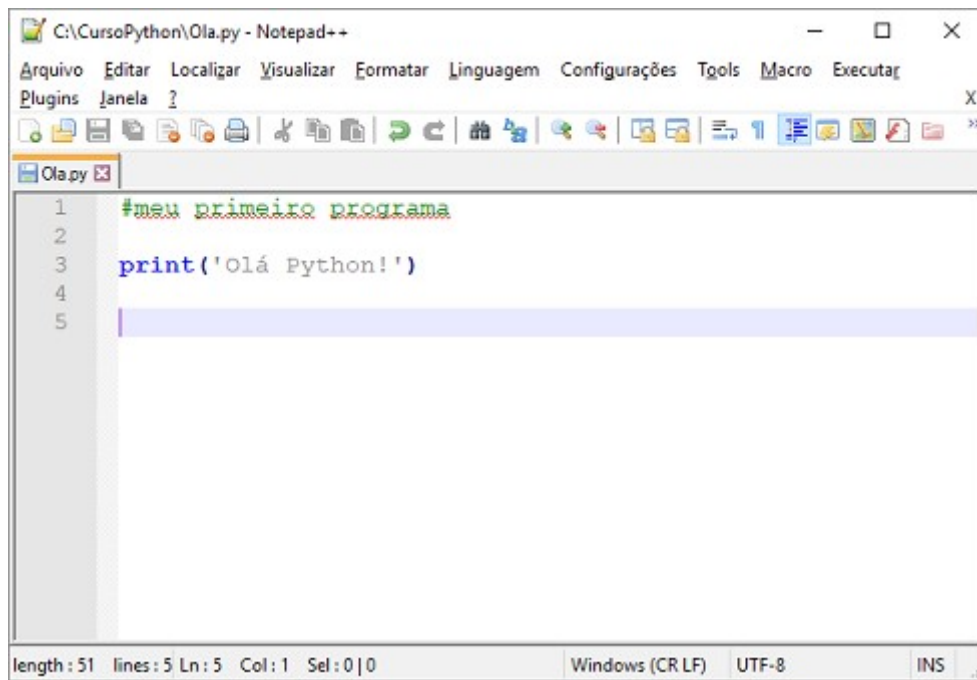
- Após alguns segundos, o aplicativo será carregado.



A.4- Criando e Executando Programas

O texto a seguir, mostra como criar, salvar e executar programas Python sem precisar usar a Spyder ou o IPython.

- Abra um editor de textos qualquer, como o Notepad++ e digite o programa. Ao terminar, salve com algum nome e a extensão “.py”. No exemplo da página a seguir, o programa foi salvo na pasta "C:\CursoPython" com o nome “Ola.py”.



- Após salvar, abra a janela de terminal ou prompt de comando. Daí, basta digitar:
`python nome_do_programa.py`
- Em nosso exemplo, como o programa está gravado na pasta "C:\CursoPython", o caminho completo foi especificado: `python c:\cursopython\ola.py` (como mostra a figura abaixo).

