

CC8210 – NCA210

Programação Avançada I

---

Prof. Reinaldo A. C. Bianchi

Prof. Leandro Alves da Silva

Prof. Isaac Jesus da Silva

Prof. Danilo H. Perico

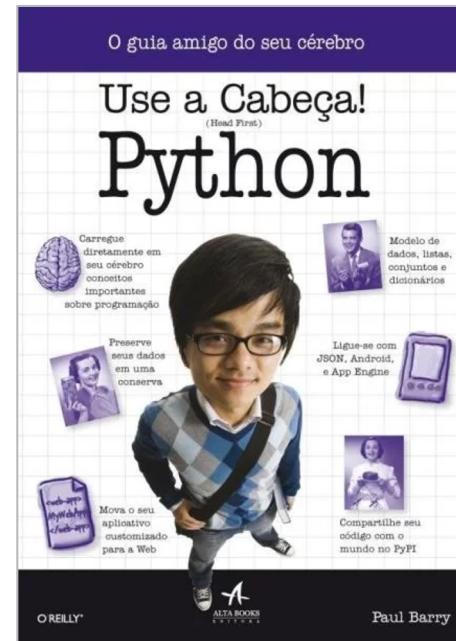
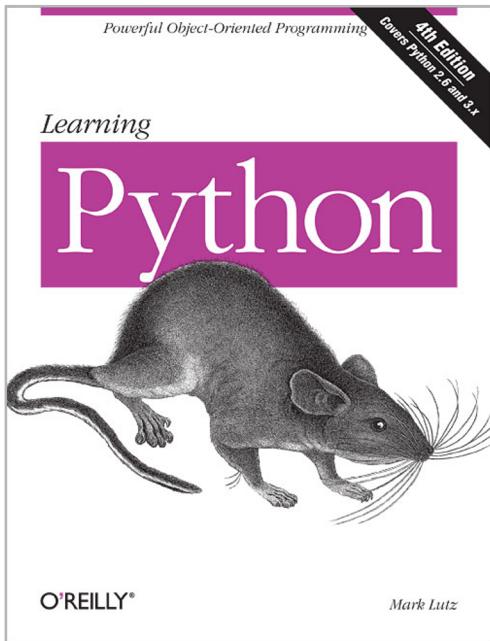
# Programação Avançada I

- Ementa:
  - Fundamentos de ambiente de programação, estudo de procedimento e função avançados, manipulação de arquivos, gráficos múltiplos e de barra à partir de dados de uma planilha, fundamentos de linguagens orientadas a objeto.
- Objetivo da disciplina:
  - Capacitar o aluno com a formação básica em programação de computadores, permitindo que possa converter e resolver problemas utilizando linguagem algorítmica, colaborando com o desenvolvimento do raciocínio lógico

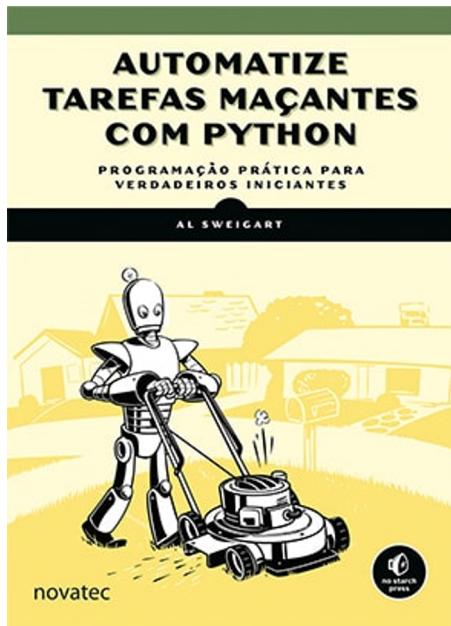
# Conteúdo Programático

AULA	DATA	TEORIA
1	10/08/20	Introdução a disciplina. Introdução a Python :Estrutura Sequencial, Condicional e Estruturas de Repetição;
2	17/08/20	Vetores ou Listas;
3	24/08/20	Listas Aninhadas ou Matrizes;
4	31/08/20	Modularização (Funções com/sem passagem de parâmetro, com/sem retorno);
5	14/09/20	Manipulação de Strings
6	21/09/20	Manipulação de Arquivos
7	28/09/20	Dicionários e Tuplas
	05/10/20	P1
8	19/10/20	Introdução à POO - Classes e Objetos
9	26/10/20	Introdução à biblioteca Pandas
10	09/11/20	Introdução à bibliotecas matemáticas e gráficas numpy e matplotlib. Manipulação de gráficos múltiplos e de barras;
11	16/11/20	Interface Gráfica do Usuário (GUI)
12	23/11/20	P2

# Bibliografia Básica



# Bibliografia Complementar



# Também disponível: python.fei.edu.br



The screenshot shows a web browser window with the URL [python.fei.edu.br](http://python.fei.edu.br) in the address bar. The page has a dark blue header with the text "Python FEI". Below the header, there are two main sections: "Sobre o projeto" and "Progressão dos módulos". The "Sobre o projeto" section contains text about the project's goal of providing free online Python classes and its structure into four modules. The "Progressão dos módulos" section provides an overview of the four modules: Python Básico, Python Orientado a Objetos, Interfaces Gráficas com Python, and Python para Backend.

## Nossos módulos



### Python Básico

Aprenda o básico sobre a linguagem de programação Python: operadores matemáticos, controle de fluxo, laços de repetição (loops) entre outros.



### Python Orientado a Objetos

Aprenda a estruturar seu código.



### Interfaces Gráficas com Python

Aprenda a criar interfaces para suas aplicações, deixando-as mais eficientes e fáceis de utilizar.



### Python para backend

Aprenda a criar e manusear servidores utilizando o Python.

# Critérios de Avaliação

$$NF = (LAB + 2*PJ + 3*PV) / 6$$

- LAB = atividades realizadas no Laboratório
- PJ = Projeto
- PV = Prova
- SUB = Substitutiva - caso a média não seja alcançada, atividade ou prova que substitui a pior nota entre PJ e PV

# Algoritmos

---

# Algoritmos

- Algoritmo é uma sequência de passos que visa atingir um objetivo bem definido (FORBELLONE, 1999).
- Algoritmo é uma sequência finita de instruções bem definidas e não ambíguas
- O conceito de algoritmo existe há séculos e o uso do conceito pode ser atribuído aos matemáticos gregos: Eratóstenes e Euclides, por exemplo.

# Algoritmos

- Executamos vários algoritmos no dia-a-dia, como por exemplo:
- Somar dois números inteiros
  - Precisamos receber os dois números
  - Então somamos o primeiro número com o segundo
  - Então, exibimos o resultado

# Algoritmos - Exercícios

- Faça um algoritmo para trocar um pneu furado do carro.
- Faça um algoritmo para trocar uma lâmpada.
- Faça um algoritmo para sacar dinheiro no banco 24 horas.

# Algoritmos

- Um algoritmo não representa, necessariamente, um programa de computador, e sim os passos necessários para realizar uma tarefa.
  - Sua implementação pode ser feita por um computador, por outro tipo de autômato ou mesmo por um ser humano.
- O conceito de algoritmo foi formalizado em 1936 pela Máquina de Turing de Alan Turing e pelo cálculo lambda de Alonzo Church, que formaram as primeiras fundações da Ciência da Computação.

# Como representar um algoritmo?

- Descrição Narrativa: escrever em linguagem natural os passos que devem ser seguidos para solucionar um problema.
- Fluxograma: escrever os passos que devem ser seguidos utilizando símbolos gráficos predefinidos.
- Pseudocódigo: escrever por meio de regras predefinidas os passos necessários para se solucionar o problema.

# Python



# Python

- É uma Linguagem de Programação.
- Linguagens de programação são usadas como um meio de comunicação entre os computadores e os humanos.
- Codificam os algoritmos para uma linguagem que o computador pode entender.
- Línguagem de alto nível.
- Interpretada.

# Tipos de Linguagem de Programação

- Baixo nível:
  - São mais próximas da maneira que o computador trabalha.
  - São mais rápidas, porém mais difíceis de se trabalhar / programar.
  - Exemplo: Assembly
- Alto nível:
  - São mais próximas da nossa linguagem;
  - Ações são representadas por palavras de ordem (faça, imprima etc).
  - Mais fáceis de se trabalhar; mais lentas do que as de baixo nível.
  - Exemplo: Python

# Linguagem Compilada vs. Interpretada

- Compilada:
  - Executam o código diretamente no Sistema Operacional ou Processador.
  - São rápidas
  - Ex.: C, C++, Fortran, Pascal
- Interpretada:
  - Necessitam de outro programa (interpretador) capaz de traduzir e executar o código fonte e executar os comandos no SO ou Processador.
  - Mais lentas
  - Ex.: JavaScript, Python, C#

# Python - História

- A Linguagem Python foi concebida no fim dos anos 80.
- A primeira ideia de implementar o Python surgiu mais especificamente em 1982 por Guido Van Rossum
- 1991: lançada a primeira versão do Python, então denominada de v0.9.0.



# Python - Origem do nome

- Guido sabia que não queria siglas;
- Ele queria que o nome da linguagem fosse marcante e forte
- Não fazia questão que o nome possuísse um significado profundo
- Foi então que Guido usou a primeira coisa que veio a sua cabeça: Monty Python's.



*Algumas vezes conhecido como os Pythons, foram um grupo de comédia britânico.*

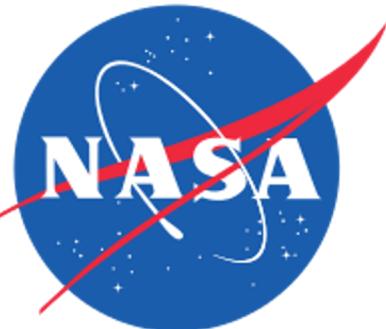
# Python Software Foundation

- Em 2001 foi criada a Python Software Foundation (PSF):
  - uma organização sem fins lucrativos
  - tem como objetivo ser dona de qualquer propriedade intelectual relacionada ao Python
  - e como missão promover e proteger o avanço da linguagem Python, além suportar e auxiliar o crescimento de comunidades de programadores Python.

# Por que usar Python???

- Vantagens:
  - Fácil de programar e aprender a programar;
  - É portável a quase todos os sistemas operacionais;
  - Rápida prototipagem;
  - Pode fazer integração com outras linguagens;
  - Produtividade.

# Quem usa Python?



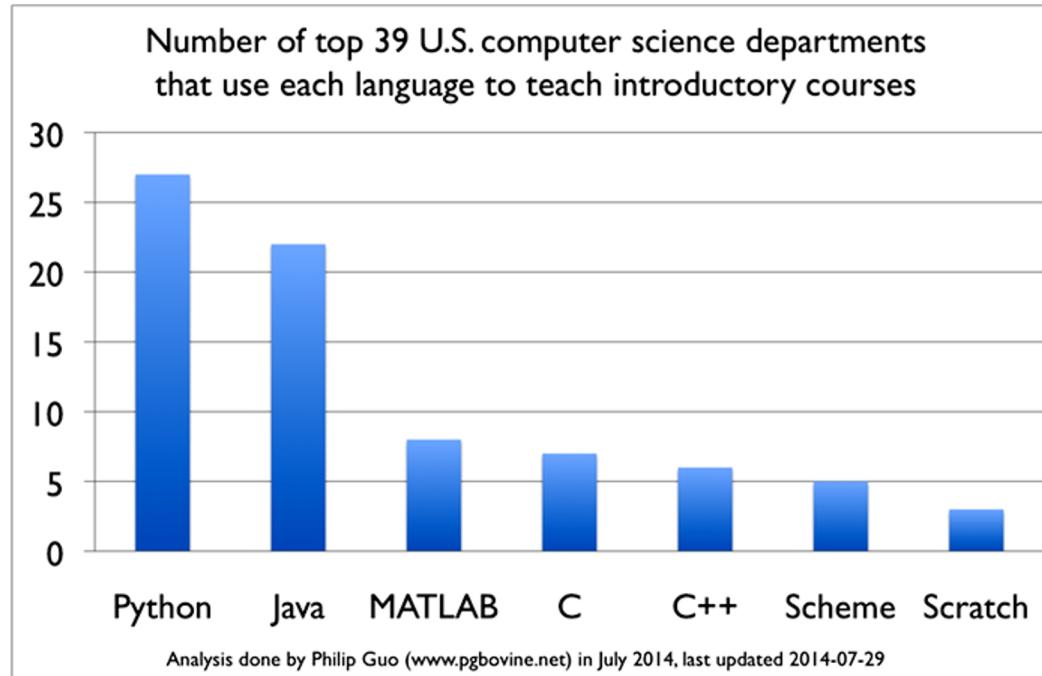
NETFLIX



Google

*iRobot*®

# Quem usa Python?

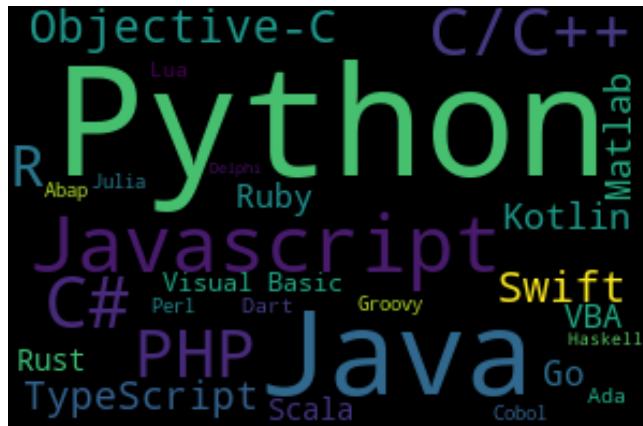


Fonte :

<http://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-us-universities/fulltext>

Comparado com as outras línguas, como está o Python???

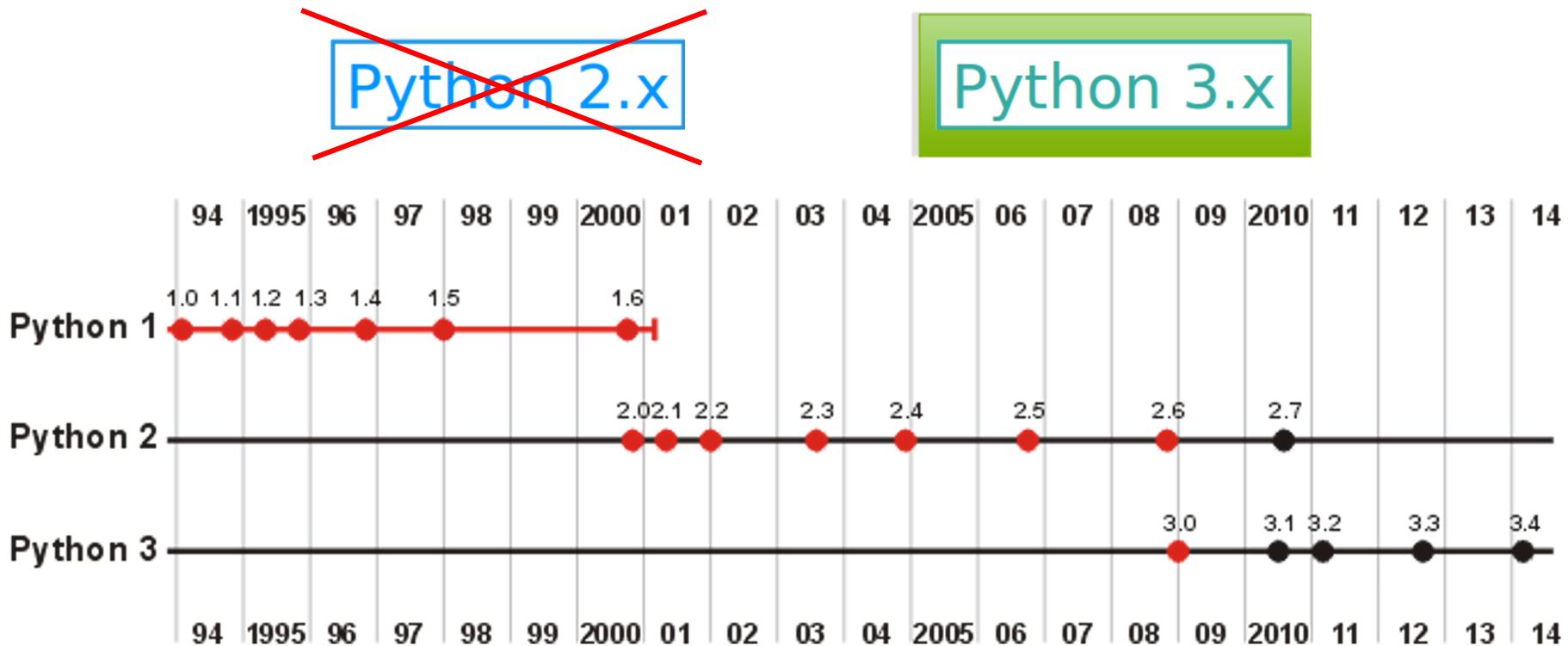
# IEEE e Github 2020



<http://pypl.github.io/PYPL.html>

Rank	Language	Type	Score
1	Python ▾	🌐💻⚙️	100.0
2	Java ▾	🌐📱💻	95.3
3	C ▾	📱💻⚙️	94.6
4	C++ ▾	📱💻⚙️	87.0
5	JavaScript ▾	🌐	79.5
6	R ▾	💻	78.6
7	Arduino ▾	⚙️	73.2
8	Go ▾	🌐💻	73.1
9	Swift ▾	📱💻	70.5
10	Matlab ▾	💻	68.4

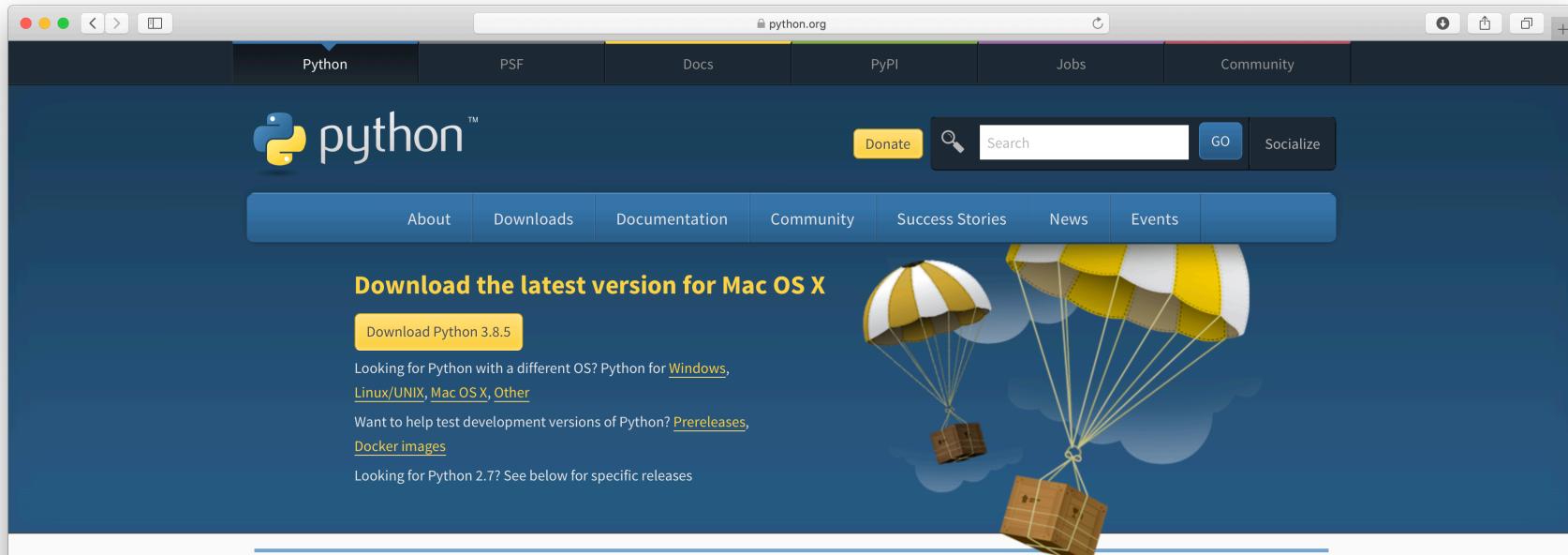
# Versões do Python



# Como programar em Python?

- Você pode baixar o programa Python no site:  
<https://www.python.org>
- Ou baixar o Python já com vários pacotes no site:  
<https://www.anaconda.com>

# Python Site: <https://www.python.org/downloads/>



python™

Donate  Search GO Socialize

About Downloads Documentation Community Success Stories News Events

**Download the latest version for Mac OS X**

[Download Python 3.8.5](#)

Looking for Python with a different OS? Python for [Windows](#), [Linux/UNIX](#), [Mac OS X](#), [Other](#)

Want to help test development versions of Python? [Prereleases](#), [Docker images](#)

Looking for Python 2.7? See below for specific releases

Active Python Releases

For more information visit the [Python Developer's Guide](#).

Python version	Maintenance status	First released	End of support	Release schedule
3.8	bugfix	2019-10-14	2024-10	<a href="#">PEP 569</a>
3.7	security	2018-06-27	2023-06-27	<a href="#">PEP 537</a>
3.6	security	2016-12-23	2021-12-23	<a href="#">PEP 494</a>
3.5	security	2015-09-13	2020-09-13	<a href="#">PEP 478</a>

Anaconda Site: <https://www.anaconda.com>

The screenshot shows a web browser window displaying the Anaconda website at <https://www.anaconda.com>. The page features a navigation bar with links for Products, Pricing, Solutions, Resources, Blog, and Company, along with a prominent 'Get Started' button. The main headline reads 'Data science technology for human sensemaking.' Below it, a subtext states: 'A movement that brings together millions of data science practitioners, data-driven enterprises, and the open source community.' The bottom half of the page is decorated with a grid of circular profile pictures of diverse individuals.

anaconda.com

ANACONDA. Products Pricing Solutions Resources Blog Company Get Started

# Data science technology for human sensemaking.

A movement that brings together millions of data science practitioners, data-driven enterprises, and the open source community.

Get Started

29

# Anaconda Site: <https://www.anaconda.com>

The screenshot shows a web browser displaying the Anaconda website at <https://www.anaconda.com>. The page features a navigation bar with links for Products, Pricing, Solutions, Resources, Blog, and Company, along with a prominent 'Get Started' button. A large central text area reads 'Data technology for ensemblemaking.' Below this, a subtext states 'Whether millions of data science practitioners, data-driven enterprises, and the open source community.' To the left, a sidebar highlights four product editions: Individual Edition (Open Source Distribution), Team Edition (Package Manager), Enterprise Edition (Full Data Science Platform), and Professional Services (Data Experts Work Together). The bottom half of the page is decorated with a grid of circular profile pictures of diverse individuals, suggesting a community of users.

Individual Edition  
Open Source Distribution

Team Edition  
Package Manager

Enterprise Edition  
Full Data Science Platform

Professional Services  
Data Experts Work Together

Get Started

Data technology for ensemblemaking.

Whether millions of data science practitioners, data-driven enterprises, and the open source community.

Anaconda Site: <https://www.anaconda.com>

The screenshot shows the Anaconda website homepage. At the top, there's a navigation bar with links for Products (underlined), Pricing, Solutions, Resources, Blog, and Company. A "Get Started" button is located on the right side of the bar. The main content area features a large green "Q" logo followed by the text "Individual Edition". Below this, a large heading reads "Your data science toolkit". A descriptive paragraph follows, stating: "With over 20 million users worldwide, the open-source Individual Edition (Distribution) is the easiest way to perform Python/R data science and machine learning on a single machine. Developed for solo practitioners, it is the toolkit that equips you to work with thousands of open-source packages and libraries." At the bottom left, there's a "Download" button.

anaconda.com

ANACONDA. Products Pricing Solutions Resources Blog Company Get Started

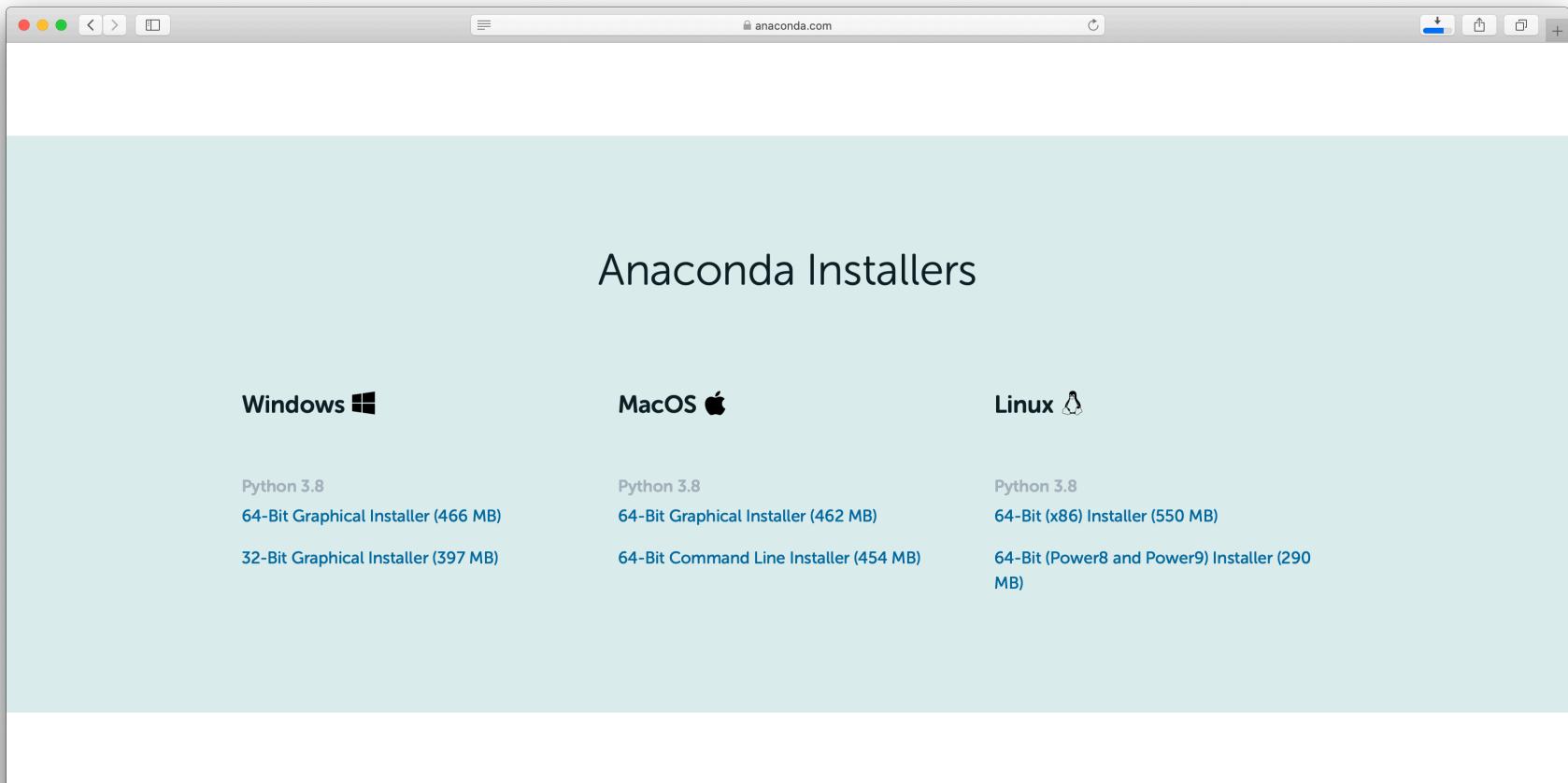
Individual Edition

# Your data science toolkit

With over 20 million users worldwide, the open-source Individual Edition (Distribution) is the easiest way to perform Python/R data science and machine learning on a single machine. Developed for solo practitioners, it is the toolkit that equips you to work with thousands of open-source packages and libraries.

Download

Anaconda Site: <https://www.anaconda.com>



The screenshot shows a web browser window with the URL "anaconda.com" in the address bar. The main content area displays the "Anaconda Installers" section. It features three columns for different operating systems: Windows, macOS, and Linux. Each column includes the system logo and a heading for Python 3.8, followed by links for 64-Bit Graphical Installer and 64-Bit Command Line Installer, along with their respective file sizes.

Platform	Python Version	Installer Type	File Size
Windows	Python 3.8	64-Bit Graphical Installer	466 MB
Windows	Python 3.8	32-Bit Graphical Installer	397 MB
macOS	Python 3.8	64-Bit Graphical Installer	462 MB
macOS	Python 3.8	64-Bit Command Line Installer	454 MB
Linux	Python 3.8	64-Bit (x86) Installer	550 MB
Linux	Python 3.8	64-Bit (Power8 and Power9) Installer	290 MB

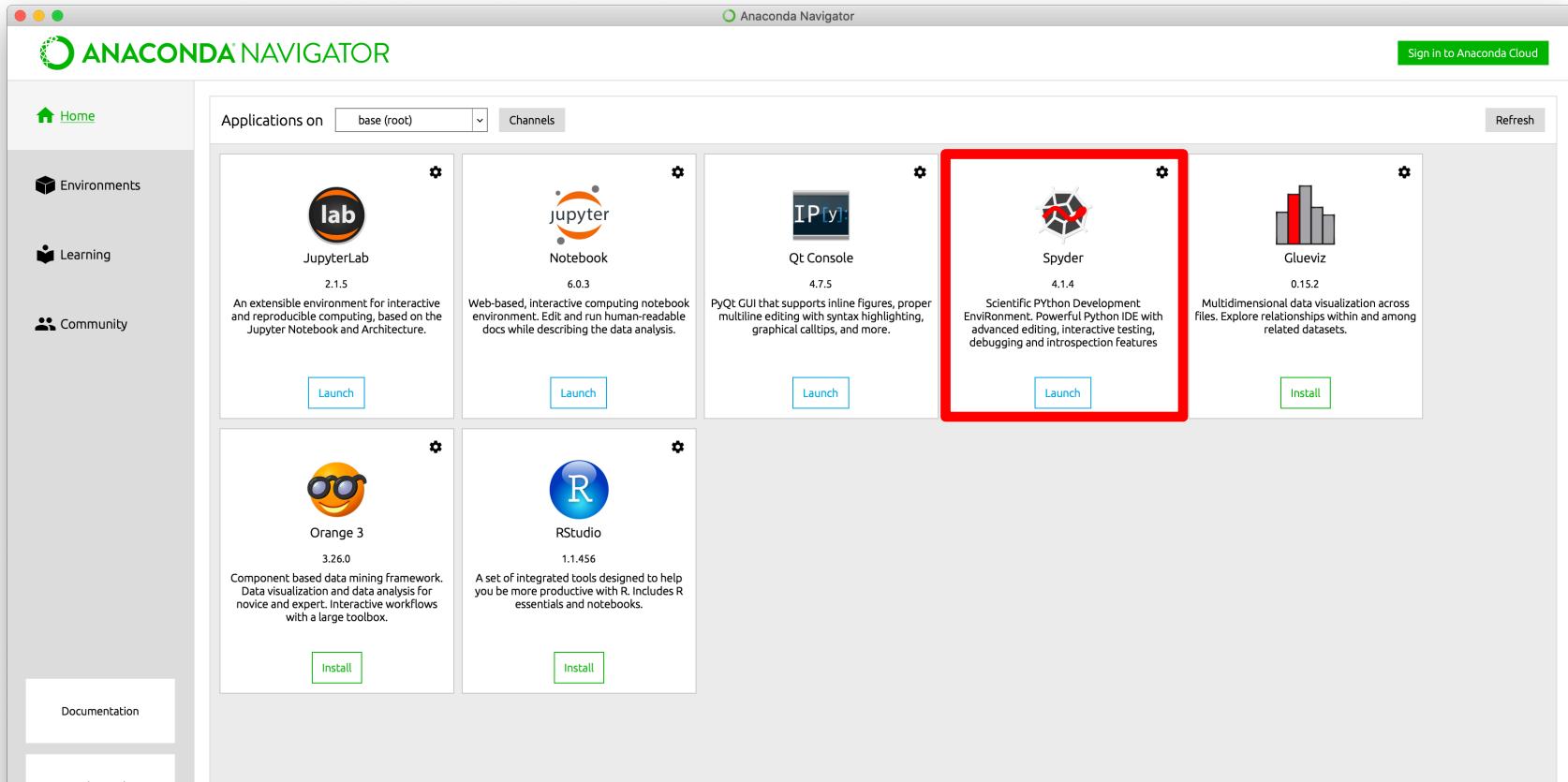
# Navegador Anaconda – Diversos aplicativos...

The screenshot shows the Anaconda Navigator application window. The left sidebar has links for Home, Environments, Learning, Community, and Documentation. The main area displays a grid of application cards:

- JupyterLab**: Version 2.1.5. An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture. Includes a **Launch** button.
- Notebook**: Version 6.0.3. Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis. Includes a **Launch** button.
- Qt Console**: Version 4.7.5. PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more. Includes a **Launch** button.
- Spyder**: Version 4.1.4. Scientific Python Development Environment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features. Includes a **Launch** button.
- Glueviz**: Version 0.15.2. Multidimensional data visualization across files. Explore relationships within and among related datasets. Includes an **Install** button.
- Orange 3**: Version 3.26.0. Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox. Includes an **Install** button.
- RStudio**: Version 1.1.456. A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks. Includes an **Install** button.

At the top right, there is a "Sign in to Anaconda Cloud" button and a "Refresh" button. The top bar also shows the title "Anaconda Navigator".

# Navegador Anaconda – IDE Spider



The screenshot shows the Anaconda Navigator interface with a sidebar and a main application grid.

**Left Sidebar:**

- Home** (selected)
- Environments
- Learning
- Community
- Documentation

**Main Content Area:**

Applications on **base (root)** | Channels | Refresh

Application	Version	Description	Action Buttons
JupyterLab	2.1.5	An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.	<a href="#">Launch</a>
Notebook	6.0.3	Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.	<a href="#">Launch</a>
Qt Console	4.7.5	PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.	<a href="#">Launch</a>
<b>Spyder</b>	4.1.4	Scientific Python Development Environment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features	<a href="#">Launch</a>
Glueviz	0.15.2	Multidimensional data visualization across files. Explore relationships within and among related datasets.	<a href="#">Install</a>
Orange 3	3.26.0	Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.	<a href="#">Install</a>
RStudio	1.1.456	A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.	<a href="#">Install</a>

# Navegador Anaconda – IDE Spider

The screenshot shows the Anaconda Navigator interface with the following applications listed:

- JupyterLab**: Version 2.1.5. An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture. Includes a **Launch** button.
- Notebook**: Version 6.0.3. Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis. Includes a **Launch** button.
- Qt Console**: Version 4.7.5. PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more. Includes a **Launch** button.
- Spyder**: Version 4.1.4. Scientific Python Development Environment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features. Includes a **Launch** button.
- Glueviz**: Version 0.15.2. Multidimensional data visualization across files. Explore relationships within and among related datasets. Includes a **Launch** button.
- Orange 3**: Version 3.26.0. Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox. Includes an **Install** button.
- RStudio**: Version 1.1.456. A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks. Includes an **Install** button.

A red box highlights the **Spyder** application. A large grey arrow points to it from the right, labeled **CLIQUE**.

/Users/rbianchi/Documents/TMECH-R1/humanoid\_field\_detection-master/train-OTHER-CLASSIFIERS-FFT.py

temp.py train-OTHER-CLASSIFIERS-FFT.py\*

```

1 import time
2 import numpy as np
3 from sklearn.metrics import classification_report
4 from sklearn.preprocessing import StandardScaler
5 from sklearn import svm
6 from sklearn.metrics import confusion_matrix
7 from sklearn.neighbors import KNeighborsClassifier
8 from sklearn.neighbors import NearestCentroid
9 from sklearn.tree import DecisionTreeClassifier
10 from sklearn.ensemble import AdaBoostClassifier
11 from sklearn.linear_model import SGDClassifier
12 from sklearn.ensemble import RandomForestClassifier
13
14 def get_one_hot(targets, nb_classes):
15     res = np.eye(nb_classes)[np.array(targets).reshape(-1)]
16     return res.reshape(list(targets.shape)+[nb_classes])
17
18 def calc_accuracy(mat):
19     c = 0
20     s = np.zeros(6)
21     for m in mat:
22         acc = m[c]/np.sum(m,dtype=np.float32)
23         s[c] = acc
24         c = c+1
25     print ("Mean: " + str(np.mean(s)) + " Variance: " + str(np.std(s)))
26
27
28 field_order = ['blanket','grass','rubber','carpet','mdf','tile']
29 sensor_order = ['angX','angY','accX','accY','accZ','gyroX','gyroY','gyroZ','torque3','torque4']
30
31 # Total: 800 + 200 = 1000
32 num_train = 700
33 num_valid = 150
34 num_test = 150
35 num_total = num_train + num_valid + num_test
36
37 num_impact = 28
38 num_sensors = len(sensor_order)
39 num_fields = len(field_order)
40
41
42 all_data = np.zeros(num_fields*num_sensors*num_impact*num_total).reshape([num_fields, num_sensors, num_impact]
43 all_output = np.zeros(num_fields*num_fields*num_total).reshape([num_fields, num_total, num_fields])
44
45
46 # Fill the input data
47 for sensor in sensor_order:
48     #all_data[:,sensor_order.index(sensor),:] = np.loadtxt("sim/" + sensor + ".csv", delimiter=",")
49
50     all_data[:,sensor_order.index(sensor),:] = np.loadtxt("real/" + sensor + ".csv", delimiter=",")
51
52 all_data = all_data.reshape([num_fields, num_sensors, num_total, num_impact])
53
54 # Shuffle data for each sensor/field
55 # Fill one-hot expected outputs
56 for sensor in sensor_order:
57     for field in field_order:
58         np.random.shuffle(all_data[field_order.index(field),sensor_order.index(sensor),:,:])
59         all_output[field_order.index(field),:,:] = get_one_hot(np.array([field_order.index(field)]),num_fields)
60
61
62

```

Source Console Object

Usage

Here you can get help of any object by pressing Cmd+I in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in Preferences > Help.

New to Spyder? Read our [tutorial](#)

Variable explorer Help Plots Files

Console 1/A

Python 3.8.1 (default, Jan 8 2020, 16:15:59)  
Type "copyright", "credits" or "license" for more information.  
IPython 7.16.1 -- An enhanced Interactive Python.

In [1]:

LSP Python: ready conda: base: Python 3.8.3 Line 4, Col 49 ASCII LF RW Mem 59%

# Python - Primeiros Programas - Saída de dados

- Saída de dados:

```
print("Olá mundo!")
```

- A função print informa que vamos exibir algo na tela.

# Python - Primeiros Programas - Entrada de Dados

- Entrada de dados:

```
input("Entre com o dado: ")
```

- A função input informa que vamos realizar entrada de dados.

# Python - Primeiros Programas - Entrada de Dados

- O problema do código de entrada exibido é que a entrada realizada pelo usuário não é salva em nenhum lugar!
- Precisamos salvar a entrada na Memória Principal (RAM).
- Como?
- Com variáveis!

# Python - Primeiros Programas - Variáveis

- Variáveis são usadas para criar uma região de memória em que vamos armazenar dados:

```
nome = input("Entre com o seu nome:")
```

- nome é uma variável que vai armazenar a entrada que o usuário digitar

# Python - Primeiros Programas - Exemplo

- Código:

```
1 nome = input("Entre com o seu nome: ")  
2 print(nome)
```

Spyder (Python 3.8)

/Users/rbianchi

/Users/rbianchi/Documents/AULAS/CC8210 - PROGRAMAÇÃO AVANÇADA I/Semana 01/exemplo01.py

x exemplo01.py\*

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Created on Sun Aug  9 16:26:51 2020
5
6 @author: rbianchi
7 """
8
9 nome = input ("Entre com o seu nome: ")
print(nome)
10
```

Source Console Object

Usage

Here you can get help of any object by pressing **Cmd+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our [tutorial](#)

Variable explorer Help Plots Files

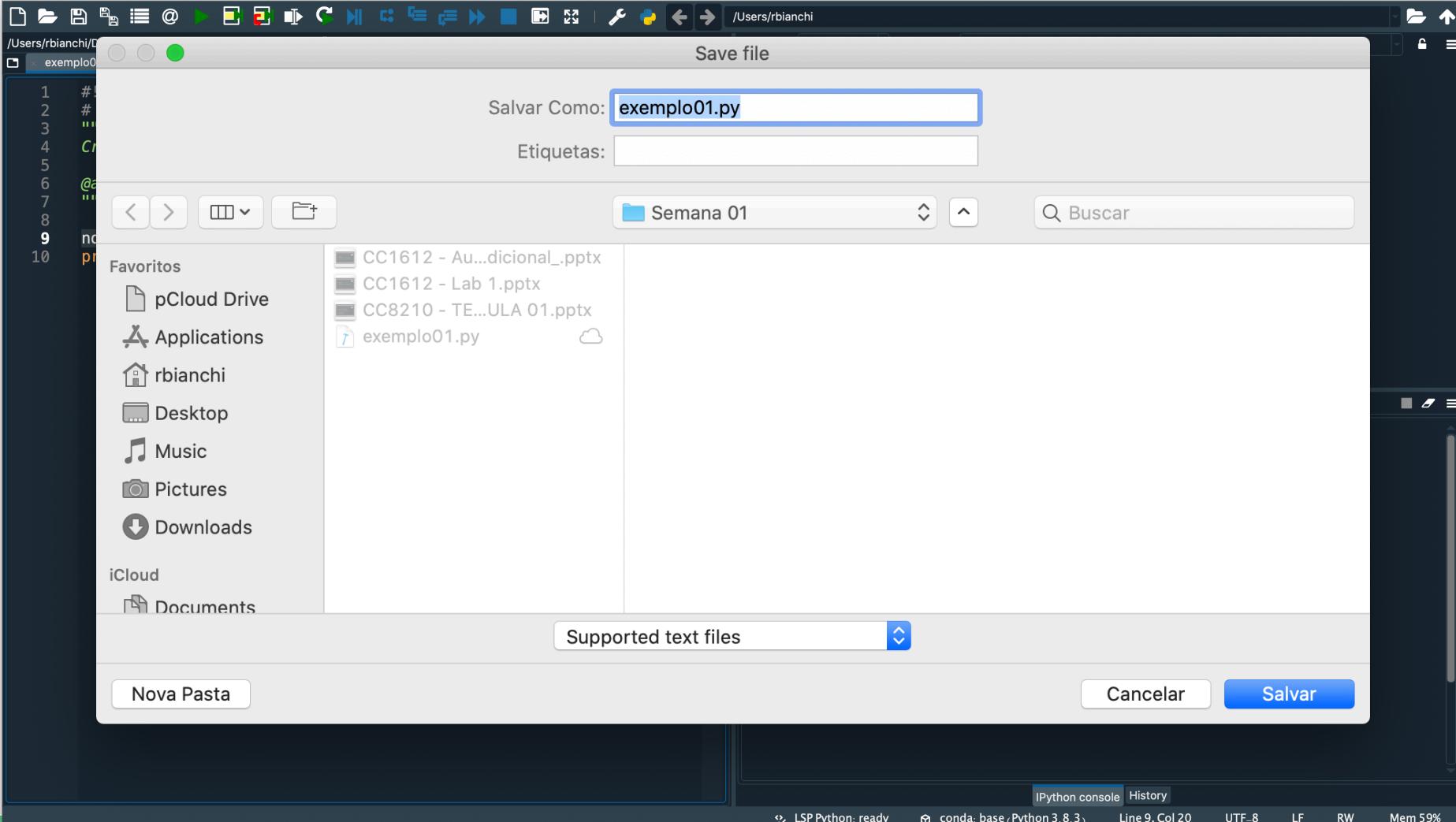
Console 1/A

```
Python 3.8.1 (default, Jan  8 2020, 16:15:59)
Type "copyright", "credits" or "license" for more information.

IPython 7.16.1 -- An enhanced Interactive Python.

In [1]:
```

LSP Python: ready conda: base / Python 3.8.3 Line 9, Col 20 UTF-8 LF RW Mem 59%



Spyder (Python 3.8)

/Users/rbianchi/Documents/AULAS/CC8210 ->

exemplo01.py

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Created on Sun Aug  9 16:26:51 2020
5
6 @author: rbianchi
7 """
8
9 nome = input ("Entre com o seu nome: ")
print(nome)
10
```

Usage

Here you can get help of any object by pressing **Cmd+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our [tutorial](#)

Variable explorer Help Plots Files

Console 1/A

```
Python 3.8.1 (default, Jan  8 2020, 16:15:59)
Type "copyright", "credits" or "license" for more information.

IPython 7.16.1 -- An enhanced Interactive Python.

In [1]:
```

LSP Python: ready conda: base (Python 3.8.3) Line 9, Col 1 UTF-8 LF RW Mem 62%

Spyder (Python 3.8)

/Users/rbianchi/Documents/AULAS/CC8210 - PROGRAMAÇÃO AVANÇADA I/Semana 01/exemplo01.py

Source Console Object

exemplo01.py

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Created on Sun Aug  9 16:26:51 2020
5
6 @author: rbianchi
7 """
8
9 nome = input ("Entre com o seu nome: ")
print(nome)
10
```

Usage

Here you can get help of any object by pressing **Cmd+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our [tutorial](#)

Variable explorer Help Plots Files

Console 1/A

```
Python 3.8.1 (default, Jan  8 2020, 16:15:59)
Type "copyright", "credits" or "license" for more information.

IPython 7.16.1 -- An enhanced Interactive Python.

In [1]: runfile('/Users/rbianchi/Documents/AULAS/CC8210 - PROGRAMAÇÃO AVANÇADA I/Semana 01/exemplo01.py', wdir='/Users/rbianchi/Documents/AULAS/CC8210 - PROGRAMAÇÃO AVANÇADA I/Semana 01')

Entre com o seu nome: Reinaldo Bianchi
Reinaldo Bianchi

In [2]:
```

LSP Python: ready conda: base (Python 3.8.3) Line 9, Col 1 UTF-8 LF RW Mem 59%

Spyder (Python 3.8)

/Users/rbianchi/Documents/AULAS/CC8210 - PROGRAMAÇÃO AVANÇADA I/Semana 01/exemplo01.py

Source Console Object

Usage

Here you can get help of any object by pressing Cmd+I in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in Preferences > Help.

New to Spyder? Read our [tutorial](#)

CLIQUE

Variable explorer

Console 1/A

Python 3.8.1 (default, Jan 8 2020, 16:15:51)  
Type "copyright", "credits" or "license" for more information.

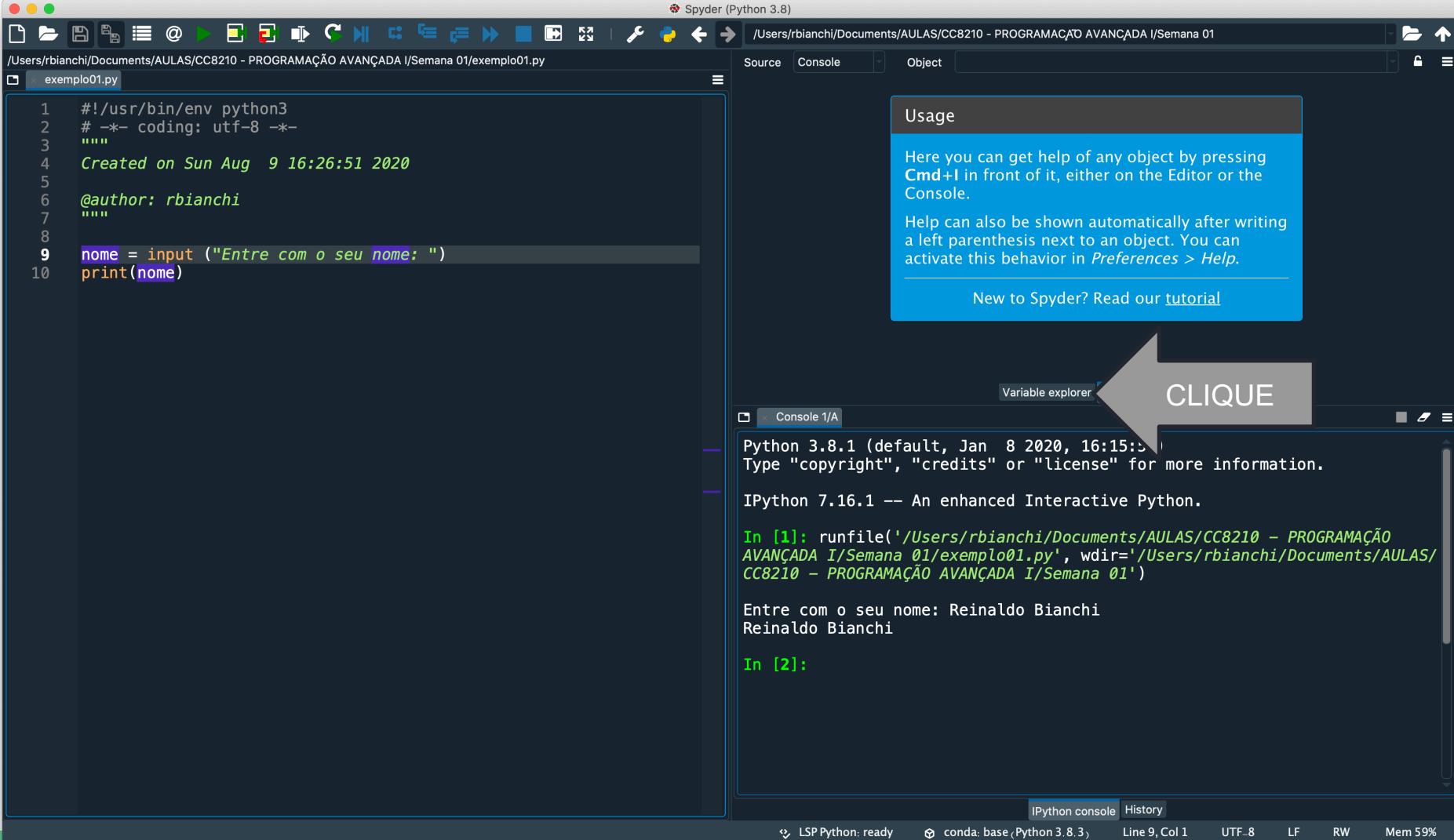
IPython 7.16.1 -- An enhanced Interactive Python.

In [1]: runfile('/Users/rbianchi/Documents/AULAS/CC8210 - PROGRAMAÇÃO AVANÇADA I/Semana 01/exemplo01.py', wdir='/Users/rbianchi/Documents/AULAS/CC8210 - PROGRAMAÇÃO AVANÇADA I/Semana 01')

Entre com o seu nome: Reinaldo Bianchi  
Reinaldo Bianchi

In [2]:

LSP Python: ready conda: base (Python 3.8.3) Line 9, Col 1 UTF\_8 LF RW Mem 59%



Spyder (Python 3.8)

/Users/rbianchi/Documents/AULAS/CC8210 - PROGRAMAÇÃO AVANÇADA I/Semana 01/exemplo01.py

exemplo01.py

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Created on Sun Aug  9 16:26:51 2020
5
6 @author: rbianchi
7 """
8
9 nome = input ("Entre com o seu nome: ")
print(nome)
10
```

Variable explorer Help Plots Files

Console 1/A

```
Python 3.8.1 (default, Jan  8 2020, 16:15:59)
Type "copyright", "credits" or "license" for more information.

IPython 7.16.1 -- An enhanced Interactive Python.

In [1]: runfile('/Users/rbianchi/Documents/AULAS/CC8210 - PROGRAMAÇÃO AVANÇADA I/Semana 01/exemplo01.py', wdir='/Users/rbianchi/Documents/AULAS/CC8210 - PROGRAMAÇÃO AVANÇADA I/Semana 01')

Entre com o seu nome: Reinaldo Bianchi
Reinaldo Bianchi

In [2]: |
```

LSP Python: ready conda: base (Python 3.8.3) Line 9, Col 1 UTF-8 LF RW Mem 59%

# Comandos básicos

---

# Variáveis

---

# Variáveis

**Uma variável é um nome que se refere a um dado ou valor**

# Variáveis

**Uma variável é um recurso utilizado nos programas para escrever e ler dados da memória do computador!**

# Variáveis

São simplesmente espaços na memória o qual reservamos e damos nomes.

# Variáveis - Tipos de Dados

- **Dados** são as entidades mais fundamentais que um programa manipula!
- Os dados podem ser de diferentes **tipos**

# Variáveis - Tipos de Dados

- Números:
  - Inteiro ( *int* ): 1 ; 2 ; -3 ; 0 ; 10
  - Real ( *float* ): 1.3 ; -3.63 ; 7.2 ; 16.42
  - Complexo ( *complex* ): 6 + 3j ; -2 + 4j
- Texto ( *string* ): “Olá” ; “Isto é uma string”
- Tipo Lógico ( *bool* ): True ; False

# Variáveis - Tipos de Dados

- Para descobrir qual é o tipo de determinado dado, podemos utilizar a função `type()` no Python.

```
In [1]: type(3.6)
```

```
Out[1]: float
```

```
In [2]: type(2)
```

```
Out[2]: int
```

```
In [3]: type("olá")
```

```
Out[3]: str
```

```
In [4]: type("2.8")
```

```
Out[4]: str
```

```
In [5]: type(2+6j)
```

```
Out[5]: complex
```

IPython 7.16.1 -- An enhanced Interactive Python.

In [1]: runfile('/Users/rbianchi/Documents/AULAS/CC8210 - PROGRAMAÇÃO AVANÇADA I/Semana 01/exemplo01.py', wdir='/Users/rbianchi/Documents/AULAS/CC8210 - PROGRAMAÇÃO AVANÇADA I/Semana 01')

Entre com o seu nome: Reinaldo Bianchi

Reinaldo Bianchi

In [2]: type (3)

Out[2]: int

In [3]: type (nome)

Out[3]: str

In [4]: type (3.1415926535897932384626433)

Out[4]: float

In [5]:

# Variáveis - Tipos de Dados - Aritmética com *float*

- Como já sabemos, o computador trabalha com números binários
- Porém, a maioria das frações decimais não pode ser representada exatamente como frações binárias.
- Uma consequência é que, em geral, os números decimais de ponto flutuante que você digita acabam sendo armazenados de *forma aproximada*.

# Variáveis - Tipos de Dados - Aritmética com *float*

- Assim, podemos ter diferenças na aritmética realizada com float

```
1 a = 0.1
2 b = 0.2
3
4 c = a+b
5
6 print(c)
```

0.3000000000000004

- Exemplo:
- Quase todas as máquinas atuais (julho de 2010) usam aritmética de ponto flutuante conforme a norma IEEE-754, que limita a quantidade de casas decimais

The screenshot shows a Jupyter Notebook interface with a dark theme. At the top, there is a toolbar with icons for file operations (New, Open, Save, etc.) and search. Below the toolbar is a variable explorer window displaying the following table:

Name	Type	Size	Value
a	float	1	0.1
b	float	1	0.2
c	float	1	0.3000000000000004
nome	str	1	Reinaldo Bianchi

Below the variable explorer is a large dark workspace area. At the bottom of the interface, there is a navigation bar with tabs: Variable explorer, Help, Plots, and Files. The Variable explorer tab is currently active. In the bottom-left corner of the workspace, there is a console window titled "Console 1/A" containing the following Python code and output:

```
In [9]: a = 0.1
In [10]: b = 0.2
In [11]: c = a+b
In [12]: print(c)
0.3000000000000004
In [13]:
```

# Variáveis - Nomes

- No Python os nomes das variáveis começam, obrigatoriamente, com uma letra
- Porém, o nome completo da variável pode conter números e o símbolo sublinhado ( \_ )
- Em programação, normalmente não se utiliza acentos nos nomes das variáveis, porém o Python 3 permite esta utilização
- O Python diferencia letras maiúsculas de letras minúsculas

Nota do BIANCHI

• NUNCA USE  
ACENTOS EM SEU  
PROGRAMA!!!!!!

# Variáveis - Nomes - Exemplos

nome	válido	Comentários
a1	sim	Começa com letra
velocidade	sim	letras
velocidade90	sim	Letras e números
salário_médio	sim	Letras e _
salário médio	não	Não pode conter espaços
_b	sim	_ é aceito no início do nome
1a	não	Não pode iniciar com números

# Variáveis - Nomes - Palavras Reservadas

- Os programadores geralmente escolhem os nomes das suas variáveis com base no que a variável vai armazenar
  - No entanto, algumas palavras são reservadas e não podem ser nomes de variáveis.

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

# Variáveis - Atribuição de Valores

- Para armazenar valores nas variáveis, utilizamos o símbolo de igual (`=`).
- Esta operação é chamada de **atribuição**, pois um valor é atribuído a uma variável
- Exemplo:

```
a = 5  
b = 6  
print(a + b)
```

11

- `a` recebe o valor 5
- `b` recebe o valor 6
- Imprimi o resultado da soma de `a` com `b`

# Variáveis - Conversão de Valores

- A conversão de valores é feita por meio das funções:
  - *int()*
  - *float()*
  - *str()*
- Exemplos:

```
In [9]: a = "3"  
b = int(a)  
type(b)
```

```
Out[9]: int
```

```
In [10]: a = 3  
b = str(a)  
type(b)
```

```
Out[10]: str
```

# Saída de dados

---

## Função *print()*

- Conforme já vimos, podemos imprimir *strings* ou o valor de variáveis com os seguintes comandos:

The image shows a screenshot of a Python code editor with two examples of the `print()` function.

The first example is:

```
print("Olá mundo!")
```

The output is:

Olá mundo!

The second example is:

```
a = 4  
print(a)
```

The output is:

4

## Função *print()*

- Para a impressão de dados na tela, podemos também combinar texto e o valor de alguma variável, utilizando vírgula ( , ) entre um dado e outro:

```
a = 3  
print("a vale", a)
```

```
a vale 3
```

## Função *print()* - composição

- Combinar várias *strings* com o valor de variáveis sem sempre é prático com o uso da vírgula.
- Por isso podemos usar composição de *strings* para combinar texto com o valor de variáveis.
- Exemplo:

```
anos = 30
print("João tem %d anos" % anos)
```

João tem 30 anos

## Função *print()* - composição

```
anos = 30  
print("João tem %d anos" % anos)
```

João tem 30 anos

- O símbolo *%d* é chamado de **marcador**.
- O marcador indica que naquela posição estaremos colocando o valor da variável *anos*, que deve ser um número inteiro neste exemplo.

# Função *print()* - composição - marcadores

marcador	tipo
%d	Número inteiro (int)
%f	Números decimais (float)
%s	Strings

# Função *print()* - composição - marcadores

- Exemplos:

```
a = "ola"  
dia = 16  
mes = 8  
print("%s hoje é dia %d do mês %d" % (a,dia,mes))
```

```
ola hoje é dia 16 do mês 8
```

```
pi = 3.141592  
print("O pi vale %f" % pi)
```

```
O pi vale 3.141592
```

## Função *print()* - composição - marcadores

- Limitando a quantidade de casas decimais na impressão:
  - Basta incluir `.x` entre o `%` e o `f`, onde x é o número desejado de casas decimais
  - Exemplo:

```
pi = 3.141592
print("O pi vale %.2f" % pi)
```

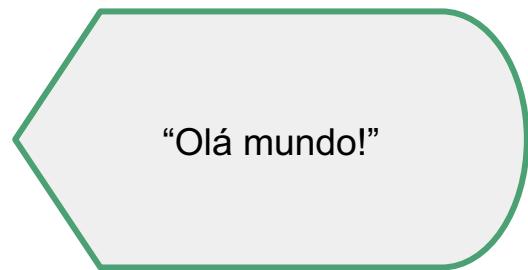
```
O pi vale 3.14
```

## Função *print()* - *format( )*

- O método ***format( )*** formata a *string* dada de acordo com o desejado para a saída de dados
  - Exemplo:

```
1 name = "Fulano"
2
3 peso = 78.51
4
5 print("0 {0} pesa {1:2.1f} Kg".format(name,peso))
6
7 0 Fulano pesa 78.5 Kg
```

# Símbolo no fluxograma - saída de dados



# Pseudocódigo

**ESCREVA** “Olá mundo!”

ou

**WRITE** “Olá mundo!”

# Entrada de dados

---

# Função input()

- A entrada de dados é feita pela função input()
  - input() aceita como parâmetro uma mensagem a ser exibida
- O valor recebido pela entrada de dados deve ser atribuído a uma variável
- Todo valor recebido pela função input() tem sempre o tipo string
- Se a ideia é utilizar o valor recebido em alguma conta ou cálculo, ele deve ser convertido para algum tipo numérico

# Função *input()*

- Exemplo:

- O valor digitado para pi será recebido como string; antes de ser atribuído a variável pi, ele é convertido em float:

```
pi = float(input("Digite o valor de pi: "))
print("O valor digitado é %.1f" % pi)
```

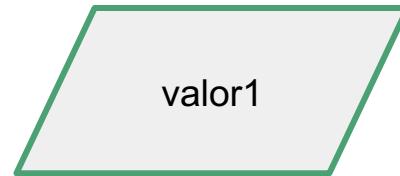
```
Digite o valor de pi: 3.14159
O valor digitado é 3.1
```

# Símbolo no fluxograma - entrada de dados



Entrada via teclado

Entrada genérica



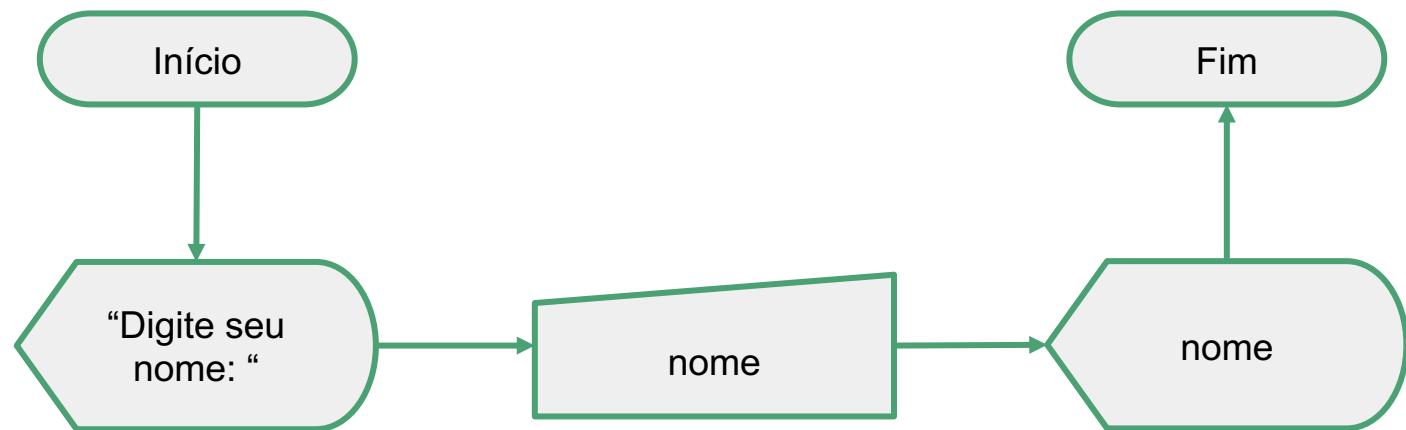
# Pseudocódigo

**LEIA** valor1

ou

**READ** valor1

# Exemplo Fluxograma



# Exemplo Pseudocódigo

**INÍCIO**

**ESCREVA** “Digite seu nome: ”

**LEIA** nome

**ESCREVA** nome

**FIM**

# Operadores Aritméticos

---

# Operadores Aritméticos

+	Soma
-	Subtração
*	Multiplicação
/	Divisão
//	Divisão com resultado inteiro
%	Módulo: retorna o resto da divisão
**	Potência: $x^{**} y$ ( $x$ elevado a $y$ )

# Operadores Aritméticos - Exemplos

```
a = 4
b = 5
c = a+b
print("Soma: ")
print(c)
c = a-b
print("Subtração: ")
print(c)
c = a*b
print("Multiplicação: ")
print(c)
c = b/a
print("Divisão: ")
print(c)
```

Soma:  
9  
Subtração:  
-1  
Multiplicação:  
20  
Divisão:  
1.25

```
a = 4
b = 5
c = b//a
print("Divisão (resultado parte inteira): ")
print(c)
c = a**b
print("Potência: ")
print(c)
c = b%a
print("Módulo: ")
print(c)
```

Divisão (resultado parte inteira):  
1  
Potência:  
1024  
Módulo:  
1

# Função Matemáticas

---

# Algumas das funções matemáticas mais usadas

- É necessário importar o módulo de matemática
- *from math import \**

<b>abs(x)</b>	Valor absoluto de x
<b>sqrt(x)</b>	Raiz quadrada de x
<b>log(x)</b>	Retorna o logaritmo natural de x
<b>log10(x)</b>	Retorna o logaritmo base-10 de x
<b>sin(x)</b>	Retorna o seno de x radianos
<b>cos(x)</b>	Retorna o cosseno de x radianos
<b>exp(x)</b>	Retorna $e^{**x}$
<b>round(x, n)</b>	Número x arredondado para n dígitos

# Exemplo

```
from math import *
print(sqrt(9))
print(log(3))
```

```
3.0
1.0986122886681098
```

# Operadores Relacionais

---

# Operadores Relacionais

- Operadores relacionais são utilizados para se realizar comparações entre valores;
- Estes valores podem ou não estar armazenados em variáveis.
- O resultado de toda comparação é um tipo lógico: **True** ou **False**

# Operadores Relacionais

operador	operação	Símbolo matemático
<code>==</code>	Igualdade	=
<code>&gt;</code>	Maior que	>
<code>&lt;</code>	Menor que	<
<code>!=</code>	diferente	≠
<code>&gt;=</code>	Maior ou igual	≥
<code>&lt;=</code>	Menor ou igual	≤

# Exemplos

```
In [26]: 5 > 9
```

```
Out[26]: False
```

```
In [27]: 9 == 9
```

```
Out[27]: True
```

```
In [28]: 7 > 3
```

```
Out[28]: True
```

```
In [29]: 4 != 6
```

```
Out[29]: True
```

```
In [30]: 2 >= 2
```

```
Out[30]: True
```

```
In [32]: a = 10  
b = -5
```

```
b >= a
```

```
Out[32]: False
```

```
In [33]: b <= a
```

```
Out[33]: True
```

```
In [34]: b != a
```

```
Out[34]: True
```

```
In [35]: b == a
```

```
Out[35]: False
```

# Estrutura Condicional

---

# Estrutura Condicional

- Nem sempre todas as linhas do código devem ser executadas!
- Normalmente, o programa deve decidir quais partes serão executadas com base em uma ou mais condições
- As condições são construídas com operadores relacionais: são feitas com base no resultado de comparações!

# Comando **if**

- Em Python, e em várias outras linguagens de programação, o comando principal para a realização de decisões é o **if**
- Sintaxe do **if** no Python:

```
if <condição>:  
    bloco verdadeiro
```

- **If** nada mais é do que nosso **se**
- Em português, podemos entender o comando if da seguinte forma:
  - **Se a condição for verdadeira, faça alguma coisa**

# Comando **if** - Exemplo

Ler dois valores e apresentar o maior deles:

```
a = int(input("Primeiro Valor: "))
b = int(input("Segundo Valor: "))

if a > b:
    print("O primeiro é o maior!")

if b > a:
    print("O segundo é o maior!")
```

```
Primeiro Valor: 87
Segundo Valor: 54
O primeiro é o maior!
```

# Comando **if** - indentação

- O bloco que será executado se a condição do if for verdadeira fica **indentado** com relação ao comando if
- **Indentação** é o **rekuo** (deslocamento do texto à direita)
- **Indentação:** neologismo derivado da palavra em inglês *indentation*
- **BLOCOS SÃO DEFINIDOS PELA IDENTAÇÃO!!!**

# Comando **if** - indentação

```
if a > b:  
    print("O primeiro é o maior!")  
    print(a)  
    print("fim do if")
```

↑  
indentação

# Comando if - indentação - Exemplo

```
a = int(input("Primeiro Valor: "))
b = int(input("Segundo Valor: "))

if a > b:
    print("O primeiro é o maior!")
    print(a)
    print("fim do if")

print("Este print() executa de forma independente com relação à condição a > b")
```

A blue curly brace is placed after the opening 'if' statement, spanning the four lines of code within the conditional block. To its right, the text 'Estes são os comandos que pertencem ao bloco do if' is written in blue.

## Comando **if** - Exemplo

Voltando ao exemplo do maior entre dois números, o que acontece se os valores digitados para os dois números forem iguais???

```
a = int(input("Primeiro Valor: "))
b = int(input("Segundo Valor: "))

if a > b:
    print("O primeiro é o maior!")

if b > a:
    print("O segundo é o maior!")
```

## Comando **if** - Exemplo

Voltando ao exemplo do maior entre dois números, o que acontece se os valores digitados para os dois números forem iguais???

Nenhuma das duas condições será verdadeira!

Nenhum print() será chamado!

## Comando **else**

- O comando **else** (senão) é utilizado nos casos em que a segunda condição é simplesmente o **contrário** da primeira.
- Sempre utilizado como uma sequência de um **if**
- Sintaxe:

```
if <condição>:  
    bloco verdadeiro  
else:  
    bloco contrário
```

Spyder (Python 3.8)

/Users/rbianchi/Documents/AULAS/CC8210 - PROGRAMAÇÃO AVANÇADA I/Semana 01/exemplo02-if-else.py

exemplo02-if-else.py

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Created on Sun Aug  9 16:26:51 2020
5
6 @author: rbianchi
7 """
8
9 a = int(input("Digite o primeiro valor: "))
10 b = int(input("Digite o segund valor: "))
11
12 if a > b:
13     print ("O primeiro numero é maior")
14
15 if b > a:
16     print ("O segundo numero é maior")
17
18 else:
19     print ("Os numeros são iguais!")
20
21
```

Nan Type Size Value

Nan	Type	Size	Value
a	int	1	10
b	int	1	10

Variable explorer Help Plots Files

Console 1/A

In [21]: runfile('/Users/rbianchi/Documents/AULAS/CC8210 - PROGRAMAÇÃO AVANÇADA I/Semana 01/exemplo02-if-else.py', wdir='/Users/rbianchi/Documents/AULAS/CC8210 - PROGRAMAÇÃO AVANÇADA I/Semana 01')

Digite o primeiro valor: 10

Digite o segund valor: 11

O segundo numero é maior

In [22]: runfile('/Users/rbianchi/Documents/AULAS/CC8210 - PROGRAMAÇÃO AVANÇADA I/Semana 01/exemplo02-if-else.py', wdir='/Users/rbianchi/Documents/AULAS/CC8210 - PROGRAMAÇÃO AVANÇADA I/Semana 01')

Digite o primeiro valor: 10

Digite o segund valor: 10

Os numeros são iguais!

In [23]:

LSP Python: ready conda: base (Python 3.8.3) Line 20, Col 5 UTF-8 LF RW Mem 58%

# Condições aninhadas

- Nem sempre nossos programas são tão simples!
- Precisamos, muitas vezes, aninhar vários **ifs** para obter o comportamento desejado no programa
- Aninhar significa colocar um **if** dentro do outro (ou um **if** dentro do **else**).

## Condições aninhadas - Exemplo

- Calcular o preço de uma conta de telefone que é formada por preços diferenciados: acima de 400 min, R\$ 0,15 por min; abaixo de 400 min, R\$ 0,18 por min; e abaixo de 200 min, R\$ 0,20 por min.

# Condições aninhadas - Exemplo

```
minutos = int(input("Quantos minutos foram utilizados este mês: "))

if minutos > 400:
    preco = 0.15
else:
    if minutos < 200:
        preco = 0.2
    else:
        preco = 0.18

print("O valor da sua conta é R$ %.2f" % (minutos*preco))
```

```
Quantos minutos foram utilizados este mês: 150
O valor da sua conta é R$ 30.00
```

## Comando **elif**

- O comando **elif** (*else if* - senão se) substitui, em muitos casos, a necessidade do aninhamento
- É sempre utilizado como uma sequência de um **if**
- Sintaxe:

```
if <condição 1>:  
    #bloco se a condição 1 for verdadeira  
elif <condição 2>:  
    #bloco se a condição 1 for falsa e a condição 2 verdadeira  
else:  
    #bloco contrário a todas outras condições
```

## Comando **elif** - Exemplo

- Exemplo da conta de telefone alterado para usar **elif**

```
minutos = int(input("Quantos minutos foram utilizados este mês: "))

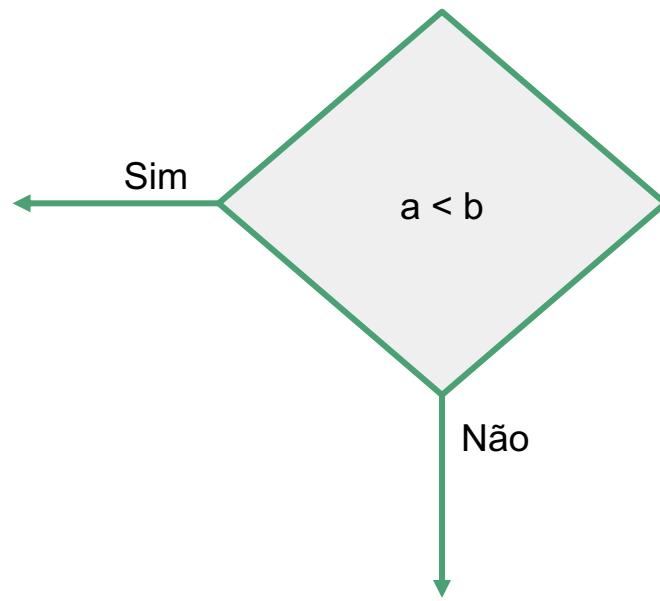
if minutos < 200:
    preco = 0.20
elif minutos < 400:
    preco = 0.18
else:
    preco = 0.15

print("O valor da sua conta é R$ %.2f" % (minutos*preco))
```

Quantos minutos foram utilizados este mês: 485

O valor da sua conta é R\$ 72.75

# Símbolo no fluxograma - condição



# Pseudocódigo

**SE** ( $a < b$ ) **ENTÃO**

Comandos para condição verdadeira

**SENÃO**

Comandos caso contrário

ou

**IF** ( $a < b$ ) **THEN**

Comandos para condição verdadeira

**ELSE**

Comandos caso contrário

# Comentários

---

# Comentários

- São textos que não são interpretados como código de programa
- Servem para documentar o programa
- No Python, uma linha de comentário começa com o símbolo # (padrão mais comum)
- Exemplo:

```
a = 8
#isso é um comentário
print(a)
```

# Operadores Lógicos

---

# Operadores Lógicos

- Podemos combinar condições para determinar como continuar o fluxo de um programa!
- O Python fornece operadores lógicos para permitir a construção de condições mais complexas.
- Os operadores lógicos mais utilizados são:
  - and (E condicional)
  - or (OU condicional)
  - not (NÃO lógico)

# Exemplo

- Faça um programa que lê um ano como entrada e verifica se esse ano é bissexto.
- Regras para definição de ano bissexto:
  - Se o ano for divisível por 400 ele é bissexto! Acaba aqui!
  - Se o ano não for divisível por 400, para ser bissexto ele deve:
    - Ser divisível por 4
    - Não ser divisível por 100
- Faça o programa com somente 1 if, 1 else, nenhum elif
- Alguns anos bissextos para verificação: 1904, 1920, 1932, 2016

## Exemplo

```
ano = int(input('Digite o ano: '))
if (ano % 400 == 0) or ((ano % 4 == 0) and (ano % 100 != 0)):
    print('É um ano bissexto')
else:
    print('Não é bissexto')
```

```
Digite o ano: 1906
Não é bissexto
```

# Operadores Lógicos

Lembre-se sempre:

- Operadores relacionais retornam sempre um valor Booleano:
  - True ou False

# Operadores Lógicos - *and*

- Operador *and*:
  - Também retorna **True** ou **False** na comparação das condições
  - Todas as condições devem ser verdadeiras para o *and* retornar **True**

# Operadores Lógicos - *or*

- Operador *or*:
  - Também retorna **True** ou **False** na comparação das condições
  - Basta que uma condição seja verdadeira para o *or* retornar **True**

# Operadores Lógicos - Exemplo Ano Bissesto

```
ano = 2000
```

```
if (ano % 400 == 0) or ((ano % 4 == 0) and (ano % 100 != 0)):
    print('É um ano bissexto')
else:
    print('Não é bissexto')
```

# Operadores Lógicos - Exemplo Ano Bissesto

ano = 2000

True

True

False

```
if (ano % 400 == 0) or ((ano % 4 == 0) and (ano % 100 != 0)):  
    print('É um ano bissexto')  
else:  
    print('Não é bissexto')
```

# Operadores Lógicos - Exemplo Ano Bissextos

ano = 2000

The diagram illustrates the execution flow of the following Python code:

```
if (ano % 400 == 0) or ((ano % 4 == 0) and (ano % 100 != 0)):  
    print('É um ano bissexto')  
else:  
    print('Não é bissexto')
```

The code checks if the year is a leap year based on the following logic:

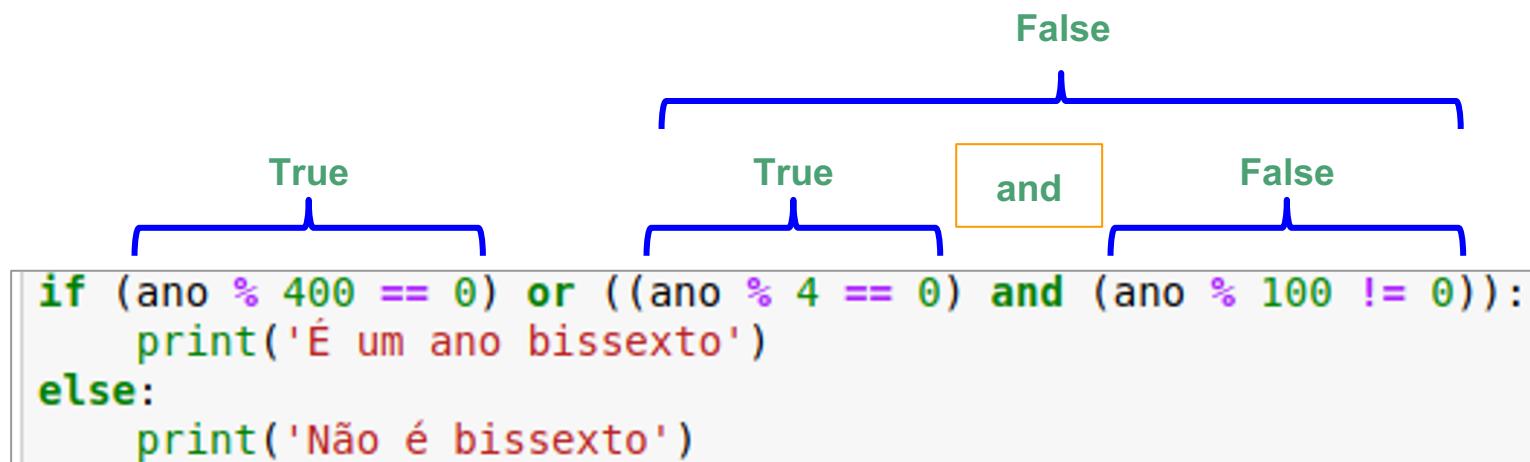
- If the year is divisible by 400, it's a leap year (True).
- Otherwise, it checks if the year is divisible by 4 and not by 100.
  - If the year is divisible by 4 (True), and not divisible by 100 (True), it's a leap year.
  - If the year is divisible by 4 (True), but divisible by 100 (False), it's not a leap year.

Annotations in the code highlight the conditions and the logical operators used:

- True**: Annotations above the first two conditions in the if-block.
- True**: Annotation above the second condition in the if-block.
- and**: A box highlights the word "and" in the expression `(ano % 100 != 0)`.
- False**: An annotation above the third condition in the if-block.

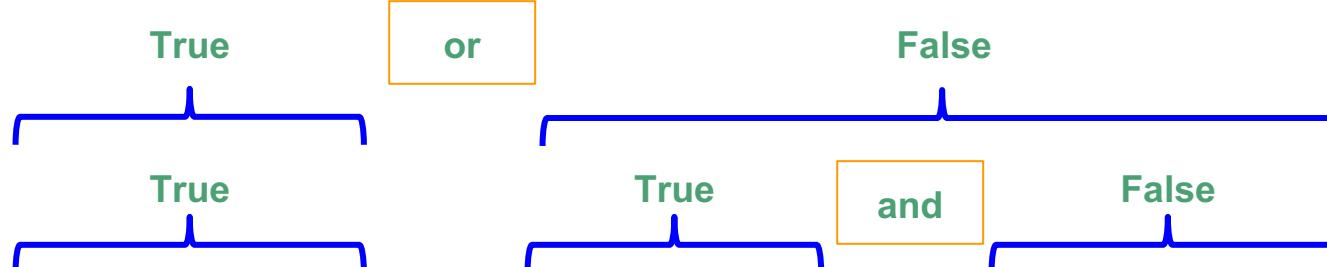
# Operadores Lógicos - Exemplo Ano Bissextos

ano = 2000



# Operadores Lógicos - Exemplo Ano Bissextos

ano = 2000



```
if (ano % 400 == 0) or ((ano % 4 == 0) and (ano % 100 != 0)):  
    print('É um ano bissexto')  
else:  
    print('Não é bissexto')
```

# Operadores Lógicos - Exemplo Ano Bissextos

ano = 2000

True

True

or

False

True

True

and

False

```
if (ano % 400 == 0) or ((ano % 4 == 0) and (ano % 100 != 0)):  
    print('É um ano bissexto')  
else:  
    print('Não é bissexto')
```

# Precedência de Operadores

primeiro



último

**	Exponencial
*, @, /, //, %	Multiplicação, multiplicação de matrizes, divisão, resto
+, -	Adição e Subtração
in, is, is not, <, <=, >, >=, !=, ==	Comparações
not x	NÃO booleano
and	E booleano
or	OU booleano
if - elif - else	Expressões condicionais

# Estruturas de Repetição

---

# Repetições

- São utilizadas para executar **várias vezes** a mesma parte do programa
- Normalmente dependem de uma condição
- Repetições são a base de vários programas!

# Exemplo

- Fazer um programa para imprimir 10 números sequenciais na tela, começando do número 1.

# Exemplo

- Esta é uma solução para o problema do exemplo:
- É uma solução boa?!
- Não: **Muitos comandos repetidos!**

```
print(1)
print(2)
print(3)
print(4)
print(5)
print(6)
print(7)
print(8)
print(9)
print(10)
```

```
1
2
3
4
5
6
7
8
9
10
```

# Estruturas de Repetição

## Comando *while*

---

# Comando **while**

- O comando **while** (enquanto) serve para executarmos alguma repetição **enquanto** uma condição for verdadeira (True)
- Sintaxe:

```
while <condição>:  
    #bloco que será repetido enquanto a condição for verdadeira
```

# Comando **while**

```
x = 1
while x <= 10:
    print(x)
    x = x + 1
```

```
1
2
3
4
5
6
7
8
9
10
```

# Exemplo

- Impressão do número 1 até o número digitado pelo usuário:

```
ultimo = int(input("Digite o último digito da contagem: "))

i = 1

while i <= ultimo:
    print(i)
    i += 1
```

Equivalente a  $i = i + 1$

Digite o último digito da contagem: 5

1  
2  
3  
4  
5

- O lado direito do sinal de igual ( $=$ ) é executado primeiro!
- O resultado é atribuído para a variável que estiver do lado esquerdo do sinal de igual ( $=$ )

# Exemplo

- Impressão do número 1 até o número digitado pelo usuário:

```
ultimo = int(input("Digite o último digito da contagem: "))

i = 1

while i <= ultimo:
    print(i)
    i += 1
```

```
Digite o último digito da contagem: 5
1
2
3
4
5
```

*i* é um contador

um contador é uma variável que conta o número de ocorrências de um evento: neste caso, o número de repetições!

## Exemplo - combinando repetição com **if**

- Programa que imprime todos os números pares de 0 até um número digitado pelo usuário.

```
ultimo = int(input("Digite o último digito da contagem: "))

i = 0

while i <= ultimo:
    if i % 2 == 0:
        print(i)
    i += 1
```

## Exemplo - combinando repetição com **if**

- Programa que imprime todos os números pares de 0 até um número digitado pelo usuário.

```
ultimo = int(input("Digite o último digito da contagem: "))

i = 0

while i <= ultimo:
    if i % 2 == 0:
        print(i)
    i += 1
```

Digite o último digito da contagem: 6

0

2

4

6

## while infinito

- Muitas vezes, queremos que nossos programas sejam executados infinitamente
- Nesses casos, podemos utilizar uma condição que nunca deixe de ser verdadeira (True)

# while infinito

**while True:**

*#bloco que sempre será executado,  
#nunca sai do loop de repetição*

## Comando **break**

- Porém, mesmo quando utilizamos um **while** infinito, é possível que em determinadas situações o programa precise sair do loop de repetição.
- Esta interrupção pode ser alcançada com o comando **break**
- O comando **break** pode ser utilizado para interromper o while, independentemente da condição

## Comando **break** - Exemplo

- Somatória de valores digitados pelo usuário até que o número 0 (zero) seja digitado; quando 0 for digitado o resultado da somatória é exibido:

```
somatoria = 0

while True:
    entrada = int(input("Digite um número a somar ou 0 para sair:"))
    if entrada == 0:
        break
    else:
        somatoria = somatoria + entrada

print("Somatória", somatoria)
```

```
Digite um número a somar ou 0 para sair:5
Digite um número a somar ou 0 para sair:6
Digite um número a somar ou 0 para sair:4
```

# Repetições aninhadas

- Podemos combinar vários **while**, um dentro do outro!
- Com isso, conseguimos alterar automaticamente o valor de mais do que somente uma variável

# Exemplo

- Programa para calcular as tabuadas do número 1 até o número

```
tabuada = 1

while tabuada <= 10:
    print()
    print("Tabuada do", tabuada)
    multiplicador = 0
    while multiplicador <= 10:
        print(tabuada, "x", multiplicador, "=", (tabuada*multiplicador))
        multiplicador += 1
    tabuada += 1
```

Tabuada do 1  
1 x 0 = 0  
1 x 1 = 1  
1 x 2 = 2  
1 x 3 = 3  
1 x 4 = 4  
1 x 5 = 5  
1 x 6 = 6  
1 x 7 = 7  
1 x 8 = 8  
1 x 9 = 9  
1 x 10 = 10

Tabuada do 2  
2 x 0 = 0  
2 x 1 = 2  
2 x 2 = 4  
2 x 3 = 6

# Estruturas de Repetição

## Comando *for*

---

# Comando for

- for é a estrutura de repetição mais utilizada
- Sintaxe:

```
for <referência> in <sequência>:  
    #bloco de código que será repetido  
    #a cada iteração
```
- Durante a execução, a cada iteração, a referência aponta para um elemento da sequência.
- Uma vantagem do for com relação ao while é que o contador não precisa ser explícito!

# Comando **for** - Exemplo

- Calcular a somatória dos números de 0 a 99

```
somatoria = 0  
  
for x in range(0,100):  
    somatoria = somatoria + x  
print(somatoria)  
  
4950
```

A função **range(*i, f, p*)** é bastante utilizada nos laços com **for**

Ela gera um conjunto de valores inteiros:

- Começando de *i*
- Até valores menores que *f*
- Com passo *p*

Se o passo *p* não for definido, o padrão de 1 será utilizado.

# Exercício

- Faça um programa que gera 100 vezes um número aleatório entre 1 e 100 e, então, exiba qual foi o maior número gerado e quantas vezes o maior número foi atualizado no seu código.

Para isso, você deve comparar o número gerado na iteração presente com o maior número armazenado até o momento.

A função `randrange(i, f)` gera números inteiros de `i` até valores menores que `f`



```
from random import randrange  
numero = randrange(1, 101)
```

## Comando `else` na repetição

- É possível a utilização do comando `else` nas estruturas de repetição
- Tanto no `while` quanto no `for`
- A cláusula `else` só é executada quando a condição do *loop* se torna falsa.
- Se você sair do *loop* com o comando `break`, por exemplo, ela não será executada.

# Comando **else** na repetição

```
i = 0
while i < 11:
    print(i)
    i+=2
else:
    print("Os números pares de 0 a 10 foram exibidos")
```

0  
2  
4  
6  
8  
10

Os números pares de 0 a 10 foram exibidos

```
for i in range(0,11,2):
    print(i)
else:
    print("Os números pares de 0 a 10 foram exibidos")
```

0  
2  
4  
6  
8  
10

Os números pares de 0 a 10 foram exibidos

# Comando **else** na repetição

- Exemplo com **break**: o **else** não é executado

```
1 i = 0
2
3 while i < 11:
4     print(i)
5     i+=2
6     if i == 8:
7         break
8 else:
9     print("Os números pares de 0 a 10 foram exibidos")
10
```

```
0
2
4
6
```

# Comando **continue** na repetição

- O comando **continue** funciona de maneira parecida com o **break**, porém o break interrompe e sai do *loop*;
- Já o **continue** faz com que a próxima iteração comece a ser executada, não importando se existem mais comandos depois dele ou não
- O **continue** não sai do *loop*
- O **continue** faz com que a próxima iteração seja executada imediatamente

# Comando **continue** na repetição

- Exemplo com **continue**

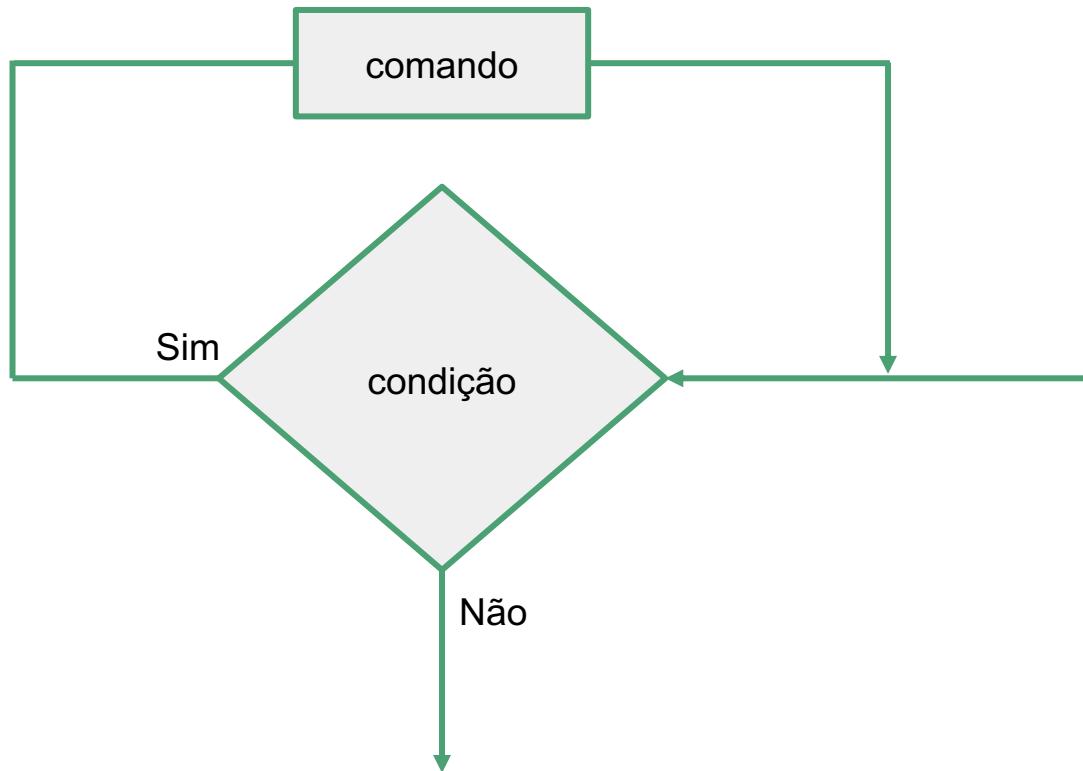
```
1 i = 0
2
3 while i < 12:
4     i+=2
5     if i == 8:
6         continue
7     print(i)
8 else:
9     print("Os números pares de 2 a 12 foram exibidos, com exceção do 8")
```

- Chama a próxima iteração
- Não executa os comandos da própria iteração: neste caso pula o *print()* com *i* = 8

```
2
4
6
10
12
```

```
Os números pares de 2 a 12 foram exibidos, com exceção do 8
```

# Símbolo no fluxograma - repetição



# Pseudocódigo

**ENQUANTO** ( $a < b$ ) **FAÇA**

    Comandos para condição verdadeira

**FIM-ENQUANTO**

ou

**WHILE** ( $a < b$ ) **DO**

    Comandos para condição verdadeira

**END-WHILE**

# Conclusão

- Vimos na aula de hoje todos os comandos básicos para programar em Python:
- Comandos de Decisão:
  - if, else, elif
- Comandos de Repetição:
  - while, for
- Já podemos começar a fazer exercícios!