

INTELIGÊNCIA COMPUTACIONAL APLICADA AO CONTROLE

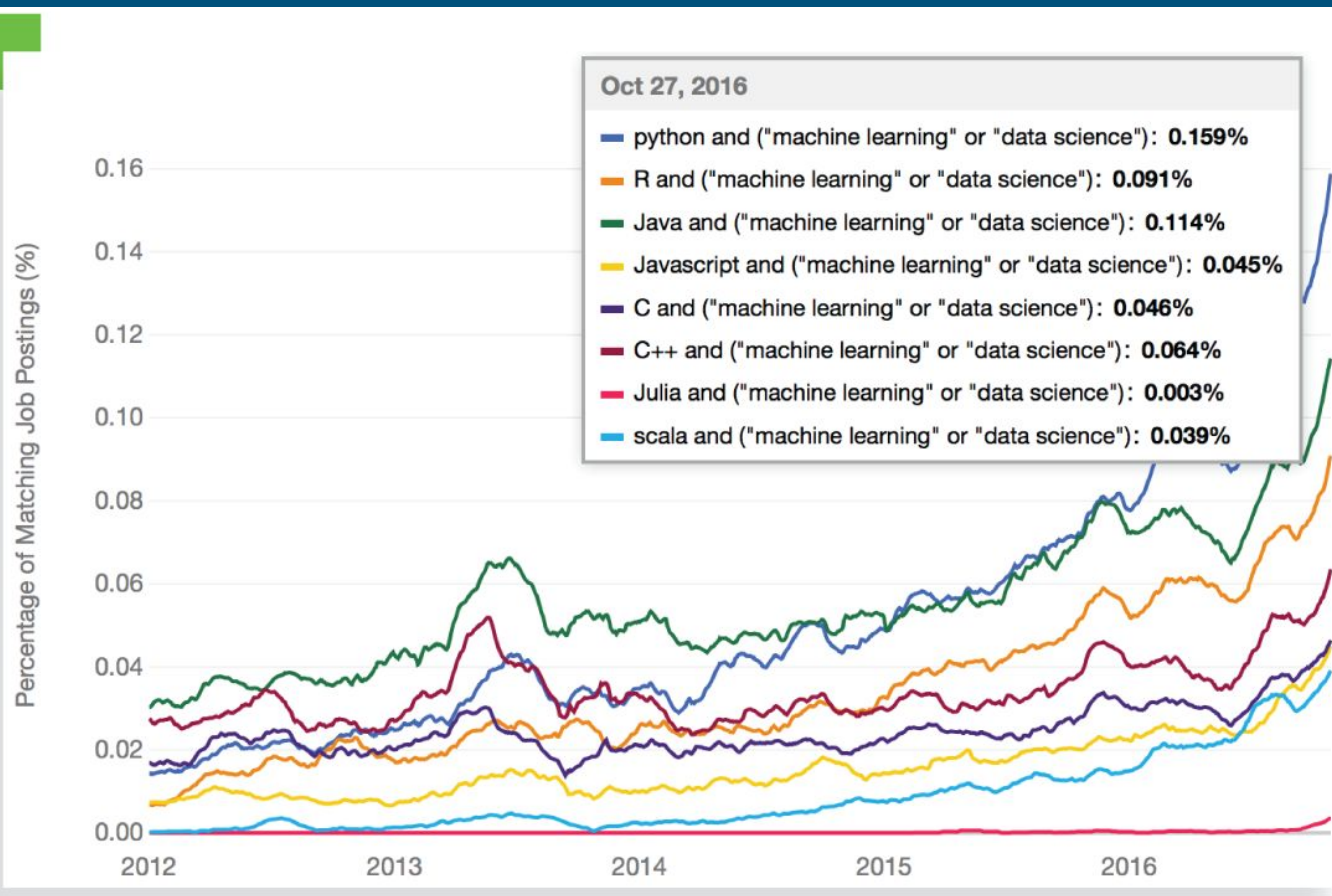
Prof. Dr. Danilo H.Perico

PYTHON



Por que Python é bom para IA?

- Um ótimo ecossistema de bibliotecas
- É fácil começar
- Flexibilidade
- Independência da plataforma
- Legibilidade
- Boas opções de visualização: gráficos etc.
- Bom suporte da comunidade
- Crescimento em popularidade

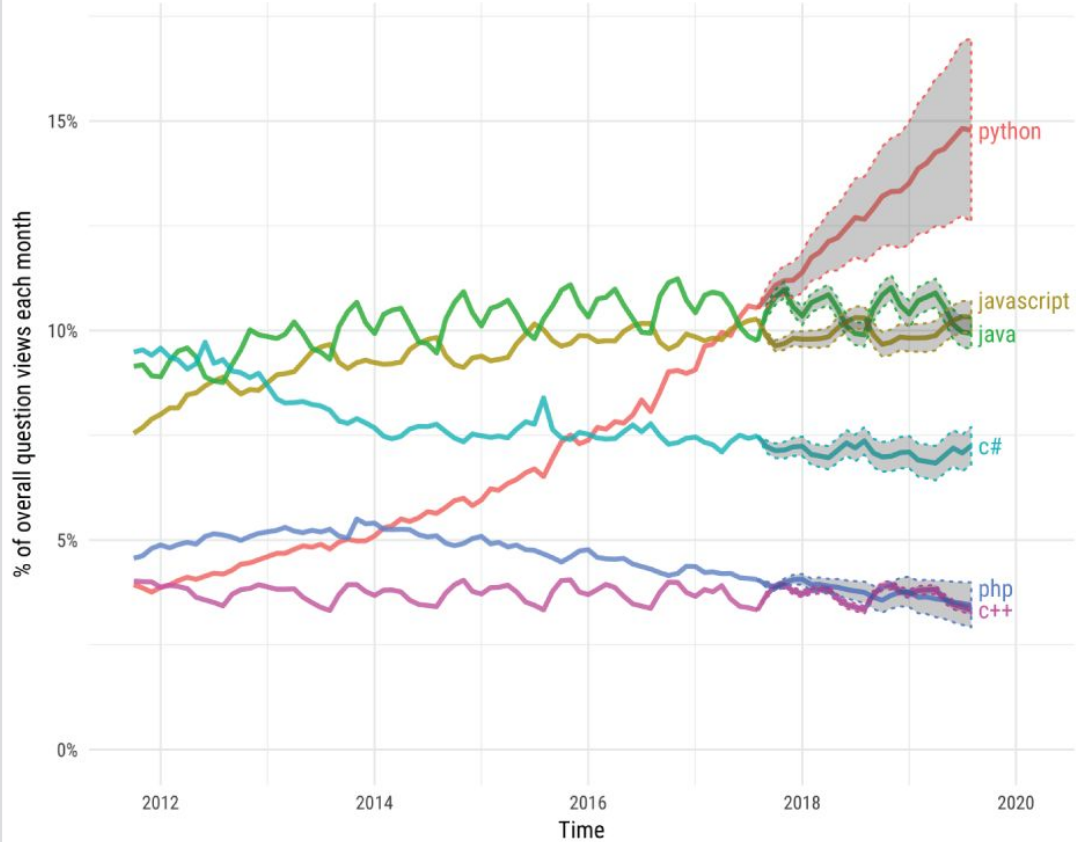


<https://djangostars.com/blog/why-python-is-good-for-artificial-intelligence-and-machine-learning/>



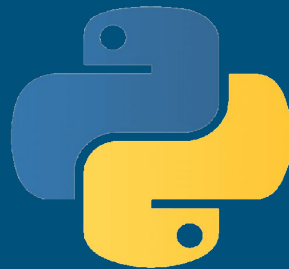
Projections of future traffic for major programming languages

Future traffic is predicted with an STL model, along with an 80% prediction interval.





História

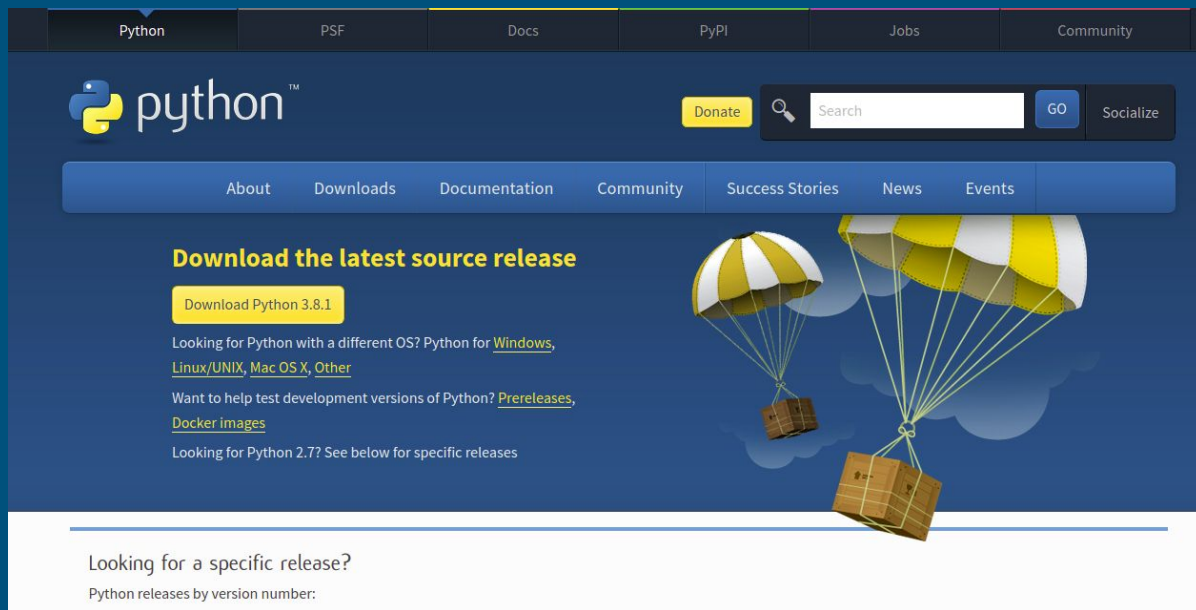


- A Linguagem Python foi concebida no fim dos anos 80
- 1991: lançada a primeira versão do Python por **Guido van Rossum**, então denominada de v0.9.0



Python

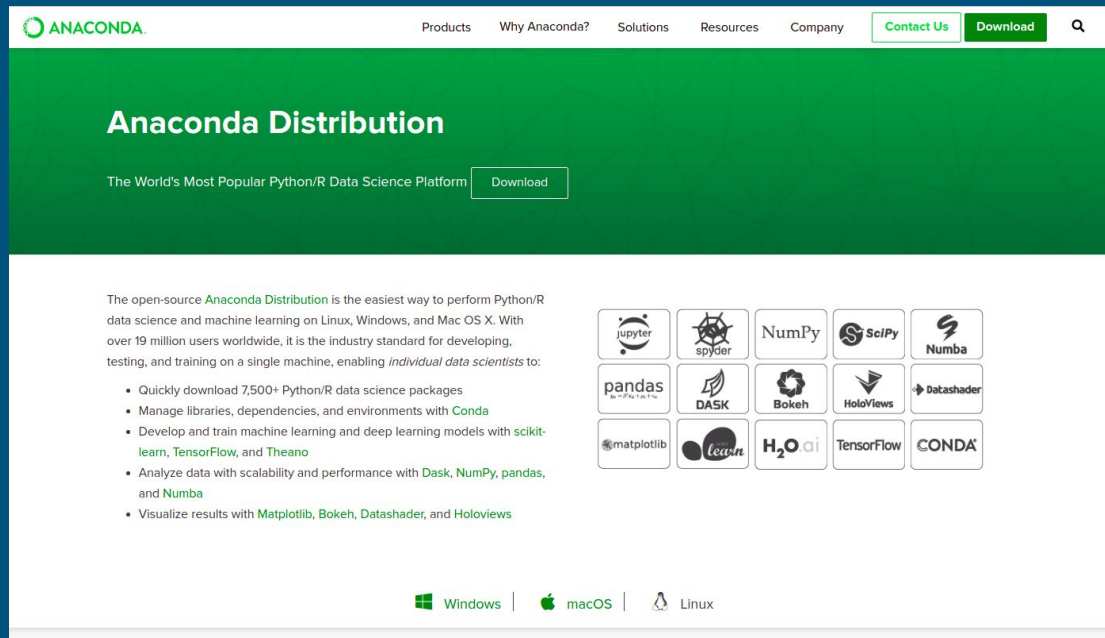
- Atualmente, o Python está na versão 3.8



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

Python - Anaconda

- Mas, em muitos casos é melhor instalar com o Anaconda



The screenshot shows the Anaconda Distribution website. The header features the Anaconda logo and navigation links: Products, Why Anaconda?, Solutions, Resources, Company, Contact Us, and Download. The main section is titled "Anaconda Distribution" with the tagline "The World's Most Popular Python/R Data Science Platform" and a prominent "Download" button. Below this, a paragraph describes the open-source distribution as the easiest way to perform Python/R data science and machine learning on Linux, Windows, and Mac OS X, citing over 19 million users. A bulleted list highlights key features: quick download of 7,500+ packages, management of libraries and environments with Conda, development and training of machine learning models with scikit-learn, TensorFlow, and Theano, analysis of data with Dask, NumPy, pandas, and Numba, and visualization with Matplotlib, Bokeh, Databricks, and Holoviews. To the right of the text is a grid of 16 logos for various data science libraries and tools, including Jupyter, Spyder, NumPy, SciPy, Numba, pandas, DASK, Bokeh, HoloViews, Databricks, matplotlib, sklearn, H2O.ai, TensorFlow, and CONDA. At the bottom, there are icons and labels for Windows, macOS, and Linux.

ANACONDA

Products Why Anaconda? Solutions Resources Company [Contact Us](#) [Download](#)

Anaconda Distribution

The World's Most Popular Python/R Data Science Platform [Download](#)

The open-source [Anaconda Distribution](#) is the easiest way to perform Python/R data science and machine learning on Linux, Windows, and Mac OS X. With over 19 million users worldwide, it is the industry standard for developing, testing, and training on a single machine, enabling *individual data scientists* to:

- Quickly download 7,500+ Python/R data science packages
- Manage libraries, dependencies, and environments with [Conda](#)
- Develop and train machine learning and deep learning models with [scikit-learn](#), [TensorFlow](#), and [Theano](#)
- Analyze data with scalability and performance with [Dask](#), [NumPy](#), [pandas](#), and [Numba](#)
- Visualize results with [Matplotlib](#), [Bokeh](#), [Databricks](#), and [Holoviews](#)

Windows | macOS | Linux

<https://www.anaconda.com/distribution/>

Porque usar Python?

Vantagens:

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

Quem usa python?





iRobot[®]



NETFLIX



IEEE - 2019 Top Programming Languages

Rank	Language	Type	Score
1	Python	  	100.0
2	Java	  	96.3
3	C	  	94.4
4	C++	  	87.5
5	R		81.5
6	JavaScript		79.4
7	C#	   	74.5
8	Matlab		70.6
9	Swift	 	69.1
10	Go	 	68.0

<https://spectrum.ieee.org/computing/software/the-top-programming-languages-2019>

Hello World

C++ (hello.cpp)

```
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello World!" << endl;
    return 0;
}
```

Python(hello.py)

```
print("Hello World!")
```

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

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("0 pi vale %f" % pi)
```

0 pi vale 3.141592

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
```


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 - 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")
```

Estes são os comandos que pertencem ao bloco do if

indentação

Comando *elif* - Exemplo

- Calcular o preço de uma conta de telefone: acima de 400 min, R\$ 0,15/min; abaixo de 400 min, R\$ 0,18/min; e abaixo de 200 min, R\$ 0,20/min.

```
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
```

Comando *while* - exemplo

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

```
ultimo = int(input("Digite o último dígito da contagem: "))  
  
i = 0  
  
while i <= ultimo:  
    if i % 2 == 0:  
        print(i)  
    i += 1
```

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***

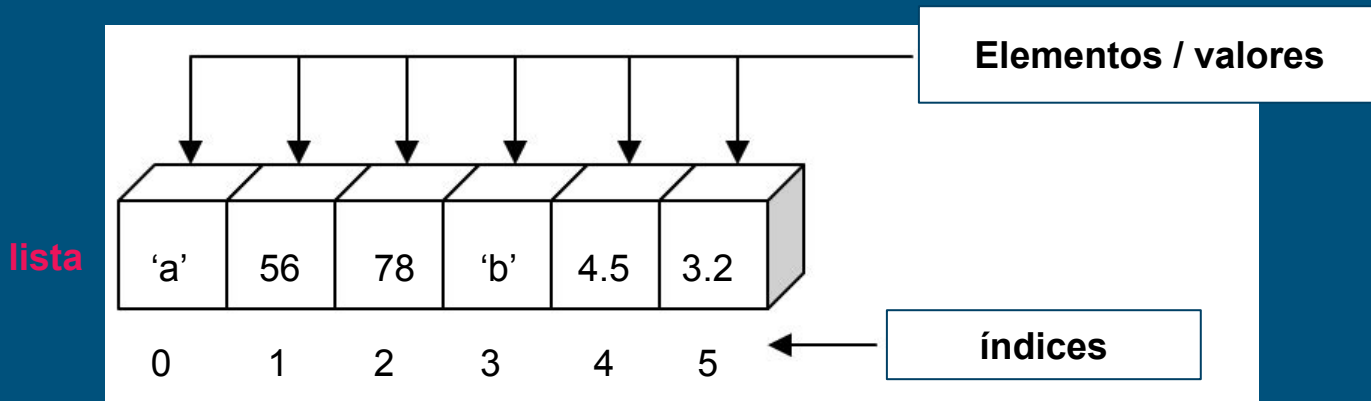
Ela gera um conjunto de valores inteiros:

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

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

Lista

- Lista é um tipo de **variável** que permite o armazenamento de valores com tipos homogêneos ou heterogêneos (do mesmo tipo ou de tipos diferentes)
- Uma lista armazena um conjunto de valores
- Os valores armazenados em uma lista são acessados por um **índice**



Lista - Acesso aos elementos

- Exemplo:

```
z = [5, 7, 1]
print(z)
```

[5, 7, 1]

```
z = [5, 7, 1]
print(z[0])
print(z[1])
print(z[2])
```

5
7
1

- Para acessarmos o primeiro número da lista *z*, utilizamos a notação: ***z[0]***
- Ou seja, da lista *z* queremos pegar o valor armazenado no índice 0.

Lista - Adicionando elementos no fim da lista

- Podemos ainda adicionar novos elementos no fim da lista
- Para isto, utilizamos o método *append(item)*
- Exemplo:

```
z = [32, 7, 1]  
print(z)
```

```
[32, 7, 1]
```

```
z.append("oi")  
print(z)
```

```
[32, 7, 1, 'oi']
```


Lista - Pesquisando na lista

- É necessário percorrer a lista toda com uma estrutura de repetição
- Exemplo:

```
z = ["a", "b", "c", "d", "e"]
for indice in range(len(z)):
    if z[indice] == "c":
        print("Elemento encontrado no índice %d" % indice)
        break
else:
    print("Elemento não encontrado!")
```

Elemento encontrado no índice 2

Matrizes

- Listas aninhadas podem ser utilizadas para representar matrizes.
- **Uma matriz é um caso específico de lista aninhada**

- Exemplo - matriz A:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

- No Python: `A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`

Dicionários - Acessando elementos

- Os valores são acessados por meio de suas chaves
- Utiliza-se o nome do dicionário e a chave dentro de colchetes []

```
ingles = {  
    'um' : 'one',  
    'dois' : 'two',  
    'tres' : 'three',  
    'quatro' : 'four',  
    'cinco' : 'five'  
}
```

```
ingles['um']
```

```
'one'
```

```
ingles['quatro']
```

```
'four'
```

```
ingles_num = {  
    1 : 'one',  
    2 : 'two',  
    3 : 'three',  
    4 : 'four',  
    5 : 'five'  
}
```

```
ingles_num[3]
```

```
'three'
```

```
ingles_num[2]
```

```
'two'
```

Importando pacotes

- Exemplo:

```
1 from datetime import datetime
2
3 agora = datetime.now()
4 print(agora)
5 print(agora.year)
```

```
2020-02-13 16:57:13.377331
2020
```

Funções com e sem parâmetros

- Exemplos:

Sem parâmetros:

```
def soma():  
    a = 9  
    b = 8  
    print(a+b)
```

Chamada
da função

soma()

17

Com parâmetros:

```
def soma(a, b):  
    print(a+b)
```

soma(3,4)

Chamada
da função

7

Funções - *return*

- Além dos parâmetros, as funções podem ou não ter um **valor de retorno**
- O retorno é definido pela palavra-chave **return**
- Exemplos:

Sem parâmetros:

```
def soma():  
    a = 9  
    b = 8  
    return(a+b)  
  
print(soma())
```

17

Com parâmetros:

```
def soma(a, b):  
    return(a+b)  
  
print(soma(3,4))
```

7

PROGRAMAÇÃO ORIENTADA A OBJETOS COM PYTHON

Exemplo Classes

```
1 class NomeClasse:
2     # atributos
3
4     # métodos
```

```
1 class Aluno:
2     nome = ""
3     ra = 0
4
5     def mostraAluno(self):
6         print("Nome: %s" % self.nome)
7         print("R.A.: %d" % self.ra)
```


Exemplo Classes

```
1 class Aluno:
2     nome = " "
3     ra = 0
4
5     def mostraAluno(self):
6         print("Nome: %s" % self.nome)
7         print("R.A.: %d" % self.ra)
8
9 aluno = Aluno()
10 aluno.nome = "Danilo"
11 aluno.ra = 123456789
12 aluno.mostraAluno()
13
14
```

Nome: Danilo
R.A.: 123456789

Exemplo Classes

```
1 class Aluno:
2     nome = " "
3     ra = 0
4
5     def mostraAluno(self):
6         print("Nome: %s" % self.nome)
7         print("R.A.: %d" % self.ra)
8
9 aluno = Aluno()
10 aluno.nome = "Danilo"
11 aluno.ra = 123456789
12 aluno.mostraAluno()
13
14
```

cria o objeto
da classe
Aluno

Nome: Danilo
R.A.: 123456789

Exemplo Classes

```
1 class Aluno:
2     nome = " "
3     ra = 0
4
5     def mostraAluno(self):
6         print("Nome: %s" % self.nome)
7         print("R.A.: %d" % self.ra)
8
9 aluno = Aluno()
10 aluno.nome = "Danilo"
11 aluno.ra = 123456789
12 aluno.mostraAluno()
13
14
```

executa o
método do
objeto aluno


Nome: Danilo
R.A.: 123456789

Construtor

No Python o método construtor da classe é chamado de `__init__`

```
1 class Aluno:
2     nome = " "
3     ra = 0
4
5     def __init__(self, nome, ra):
6         self.nome = nome
7         self.ra = ra
8
9     def mostraAluno(self):
10        print("Nome: %s" % self.nome)
11        print("R.A.: %d" % self.ra)
12
13 aluno = Aluno("Danilo", 123456789)
14 aluno.mostraAluno()
15
16
```

construtor:
inicializa o valor
dos atributos
quando o objeto é
instanciado



```
Nome: Danilo
R.A.: 123456789
```

EXERCÍCIOS

Exercícios

1. Faça um programa que solicita do usuário uma quantidade de dias, horas, minutos e segundos. Calcule e imprima o total convertido em somente segundos.
2. Faça um Programa que peça dois números e, então, imprima o maior deles.
3. Escreva um programa que pergunte o salário de um funcionário e calcule o valor do aumento. Para salários superiores a R\$ 1250,00, calcule um aumento de 10%. Para inferiores ou iguais, de 15%. Imprima o novo salário.

Exercícios

4. Um posto está vendendo combustíveis com a seguinte tabela de descontos:
- a. Álcool:
 - i. até 20 litros, desconto de 3% por litro
 - ii. acima de 20 litros, desconto de 5% por litro
 - b. Gasolina:
 - i. até 20 litros, desconto de 4% por litro
 - ii. acima de 20 litros, desconto de 6% por litro

Escreva um programa que solicita o número de litros vendidos e o tipo de combustível (álcool ou gasolina). Então, calcule e imprima o valor a ser pago pelo cliente sabendo-se que o preço do litro da gasolina é R\$ 4,59 e, do álcool, R\$ 3,19.

Exercícios

5. Escreva um programa que leia números digitados pelo usuário. O programa deve ler os números até que 0 (zero) seja digitado. Quando 0 for digitado, o programa deve exibir a quantidade de dígitos que foram digitados, a somatória destes dígitos e a média aritmética.
6. Faça um Programa que leia 20 números inteiros e armazene-os em uma lista. Armazene os números pares na lista **par** e os números **ímpares** na lista ímpar. Imprima as três listas no final.

Exercícios

7. Para ganhar o prêmio principal em uma determinada loteria, é preciso combinar todos os 6 números em seu bilhete com os 6 números entre 1 e 49 que são sorteados pelo organizador da loteria. Escreva um programa que gere uma seleção aleatória de 6 números para um bilhete de loteria. Assegure-se de que os 6 números selecionados não contenham duplicatas. Exiba os números gerados.

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

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

Gera números inteiros entre 1 e 100

Exercícios

8. Uma data é considerada mágica quando o dia multiplicado pelo mês é igual ao ano de dois dígitos. Por exemplo, 10 de junho de 1960 é uma data mágica porque junho é o sexto mês e 6 vezes 10 é 60, o que equivale ao ano de dois dígitos. Escreva uma função que determine se uma data é ou não uma data mágica. Use sua função em um programa principal que deve encontrar e exibir todas as datas mágicas do século XX.

Exercício 9 - classe Retângulo

- Atributos:
 - lado1 e lado2
- Métodos:
 - *set* e *get* para os atributos
 - *area()*, que retorna a área do retângulo
 - *perimetro()*, que retorna o perímetro do retângulo
 - construtor que inicializa lado1 e lado2
- *main()*:
 - obtenha pelo teclado o valor de lado1 e lado2
 - crie um objeto (ret1) do tipo Retangulo com lado1 e lado2
 - obtenha novamente lado1 e lado2 e crie outro objeto (ret2)
 - exiba o conteúdo dos atributos de ret1 e ret2 utilizando os *gets*
 - exiba o perímetro e a área dos objetos ret1 e ret2 por meio dos métodos *area()* e *perimetro()*