

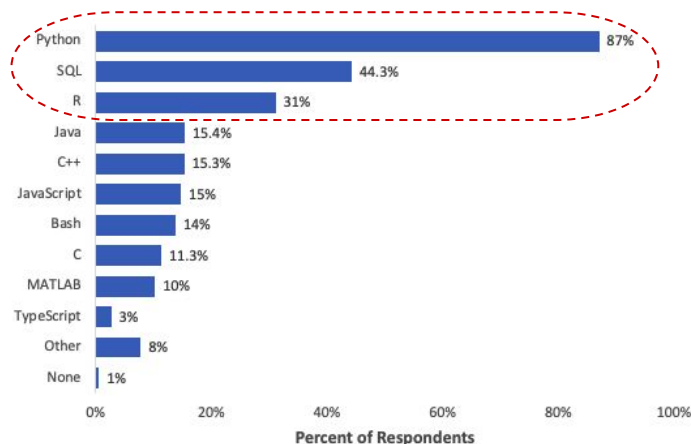
# Introdução ao Python

mentorama.

mentorama.

# Linguagens

What programming languages do you use on a regular basis?



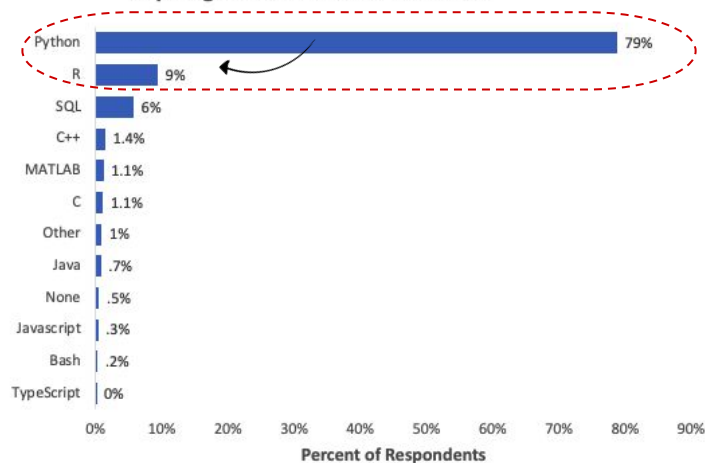
Note: Data are from the 2019 Kaggle ML and Data Science Survey. You can learn more about the study here: <https://www.kaggle.com/c/kaggle-survey-2019>.

A total of 19717 respondents completed the survey; the percentages in the graph are based on a total of 14762 respondents who provided an answer to this question.



Copyright 2020 Business Over Broadway

What programming language would you recommend an aspiring data scientist to learn first?



Note: Data are from the 2018 Kaggle ML and Data Science Survey. You can learn more about the study here: <https://www.kaggle.com/c/kaggle-survey-2019>.

A total of 19717 respondents completed the survey; the percentages in the graph are based on a total of 14377 respondents who provided an answer to this question.



Copyright 2020 Business Over Broadway

# Python

- Python foi criada por Guido van Rossum em 1991, no Centro de Matemática e Computação, em Amsterdã;
- Possui uma sintaxe concisa e clara, cujo objetivo é ser simples e de fácil aprendizado;
- Suporta a maioria das técnicas da programação orientada a objeto.
- Case sensitive: sensível a letras maiúsculas e minúsculas.
- Interpretada
- Linguagem de alto nível;
- Tipagem dinâmica e forte;
- Grande comunidade ativa;
- Pacotes exclusivos para Ciência de Dados (Numpy/Pandas/ scikit-learn)



# O ambiente de desenvolvimento

mentorama.

mentorama.

# Anaconda

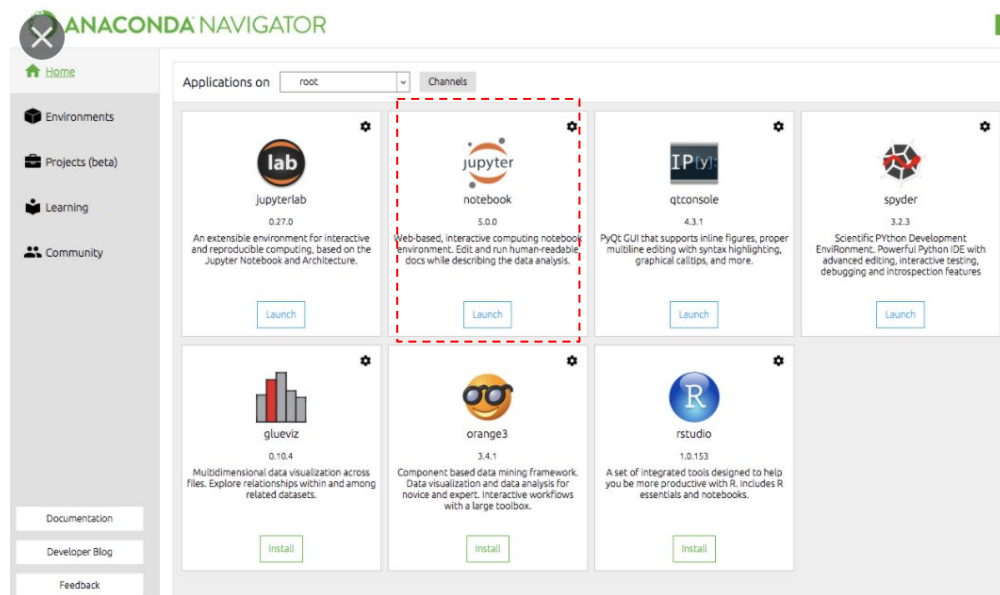


Ferramentas para análise  
de dados em um único  
arquivo

Praticidade

Windows, Linux, e macOS

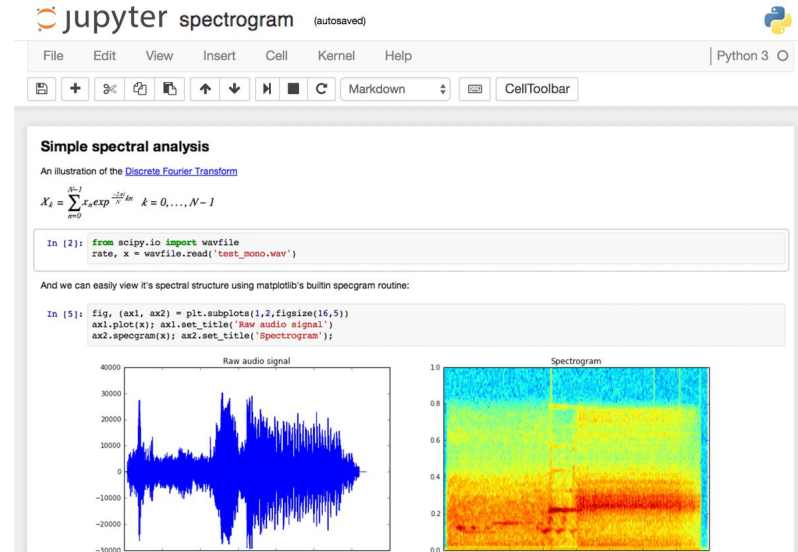
Free e  
Open-Source



mentorama.

# Jupyter notebook

Ambiente computacional web;  
Códigos, gráficos, textos;  
Sequências de células de códigos;  
Organização;  
Didático;



Curiosidades...

 jupyter = Julia + Python e R



Galileu que registram a descoberta das luas de Júpiter.

# mentorama.

# Instalando o ambiente

- Acessar o site: <https://www.anaconda.com/products/individual>;
- No fim da página (Anaconda Installers), escolher qual instalador de anaconda de acordo com seu sistema operacional;
- NNF (Instalador next-next-finish)
- Abrimos o anaconda navigator;
- Launch jupyter;

# Instalando o anaconda (Windows/Linux)

Acessar o site <https://docs.anaconda.com/anaconda/install/> e seguir o passo a passo.



# Fundamentos de Python 1

Jéssika Ribeiro

mentorama.

mentorama.

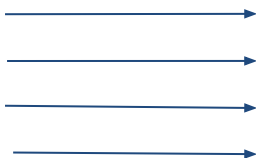
# Variáveis

# Variáveis

Uma variável é um objeto, um espaço em memória que utilizamos para representar um valor, que será usado no código.

## Variáveis

```
Nome = "Jessika"  
Idade = 28  
Altura = 1.68  
e_matematica = True
```



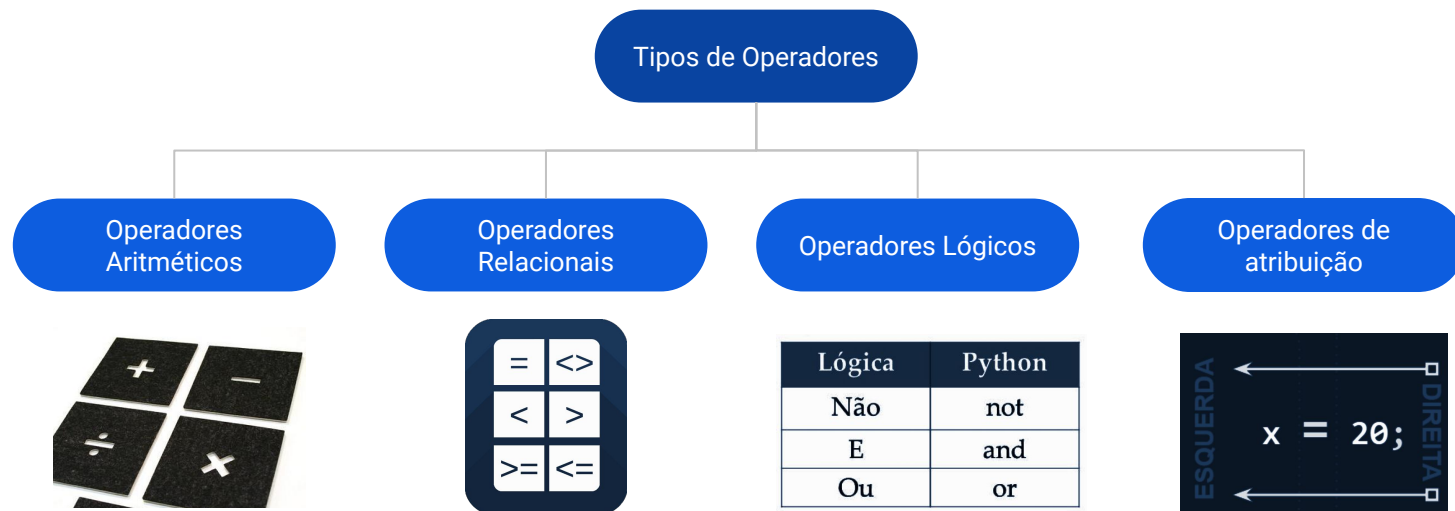
## Tipos

```
String  
Int  
Float  
Boolean
```

# Operadores

# Operadores

Formas de se combinar elementos básicos da linguagem para formar expressões mais complexas.



# Estruturas de dados

# Estruturas de dados

Coleções de dados com características específicas

**Listas:** `[1,2.5, "dados",[4,8]]` → Mutáveis, dados de vários tipos;

**Tuplas:** `(1,2,3,2)` → Imutável (alterar a tupla toda apenas) , dados de vários tipos;

**Dicionários:** `{"AI":1, "ML":2, "DL":[3,4]}` → Mutáveis, conceito chave-valor;

**Conjuntos:** `{1,2,3}` → Mutáveis, não ordenado, elementos únicos;

# Fundamentos de Python 2

Jéssika Ribeiro

mentorama.

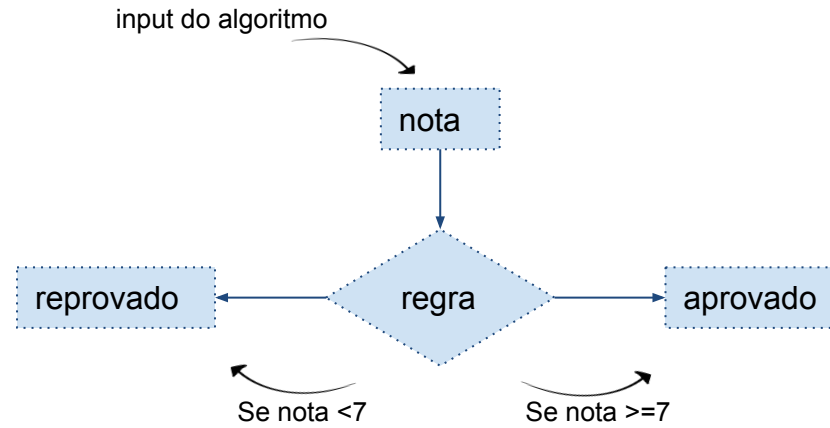
mentorama.



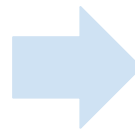
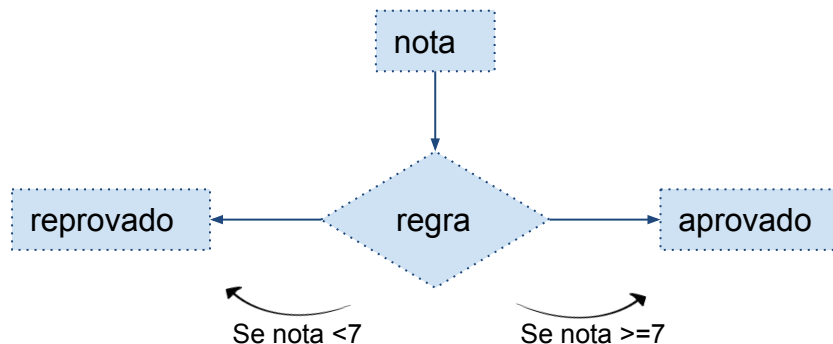
# Estruturas de decisão

# Estruturas de decisão

Utilizadas quando queremos que o algoritmo tome caminhos diferentes de acordo com o valor de alguma variável de interesse



# If - Else



```
nota = 5

if nota >= 7:
    print("Parabéns! Você está aprovado!")
else:
    print("Você foi reprovado! :/" )
```

```
if (condição logica):
    executa se for verdadeiro
else:
    executa se for falso
```

Se a condição for verdadeira, executa o bloco indentado abaixo do if

Senão executa o bloco indentado abaixo do else

# Estruturas de repetição

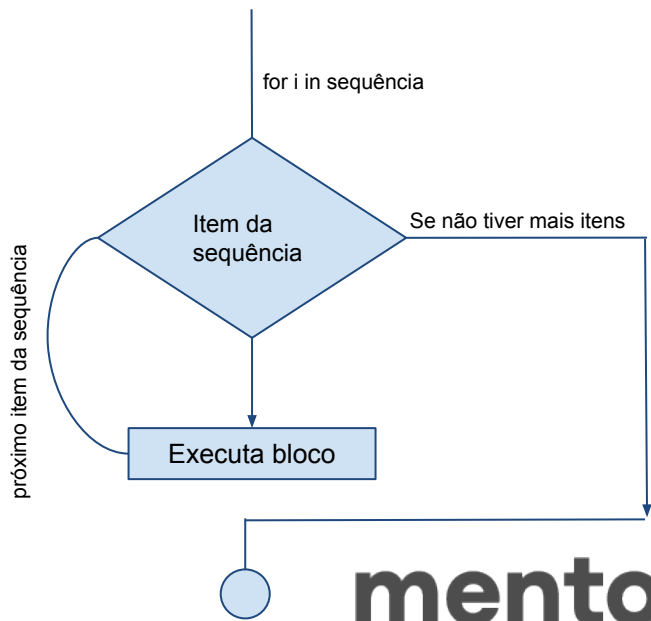
# Estruturas de repetição

As estruturas de repetição são utilizadas quando queremos que um bloco de código seja executado mais de uma vez.



# Loop For

Usado quando queremos executar um bloco de código um número fixo de vezes.



sequência

1º loop

2º loop

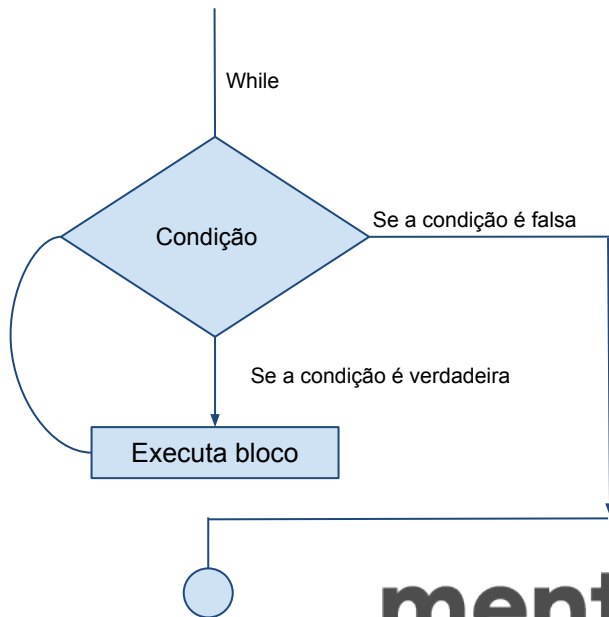
```
notas = [9,9.5,8,8.5,7,6.5]
```

var iteradora

```
for nota in notas:  
    print(nota)
```

# While

É usado quando queremos repetir um bloco de código enquanto uma condição é verdadeira



Condição do while

```
i = 1  
while i < 6:  
    print(i)  
    i += 1
```

i = 1  
i = 2  
i = 3  
i = 4  
i = 5  
i = 6

# Métodos e funções



# Métodos e funções

Métodos e funções são nossos velhos conhecidos...

```
linguagens.append(["Java", "C++"])  
linguagens
```

```
['Python', 'SQL', 'R', 'Java', 'C++']
```

```
: primeiro_dict.update({"Joaquim": 7.7})
```

```
: primeiro_dict
```

```
: {'Joao': 9, 'Maria': 7.5, 'Pedro': 6, 'Mar
```

Blocos de código reutilizáveis, ou seja, que podem ser chamados em qualquer parte do código

**Métodos:** Um bloco de instrução, com nome único e que nunca retorna valores.

```
a = print("Ola Mundo")
```

```
Ola Mundo
```

```
a
```

**VS**

**Funções :** Um bloco de instrução, com nome único e que sempre retorna valores.

```
a = len("Ola Mundo")
```

```
a
```

```
9
```

# Métodos e funções

## Built-ins

- `abs(object)`
- `bool(object)`
- `dict(k1=v1)`
- `dir(object)`
- `divmod(x, y)`
- `enumerate(iter)`
- `file(na, mod, buff)`
- `float(object)`
- `help(object)`
- `int(object)`
- `isinstance(obj, cls)`
- `len(iter)`
- `list(iter)`
- `long(object)`
- `max(iter)`
- `min(ter)`
- `open(na, mod, buff)`
- `range(start, stop, step)`
- `reversed(iter)`
- `set(iter)`
- `sorted(iter)`
- `str(object)`
- `sum(iter)`
- `tuple(iter)`
- `type(object)`
- `unicode(object)`

## Customizável

Definição da função inicia com "def"

Nome da função

Argumentos

Identação

```
def funcao_que_faz_algo(arg1, arg2):  
    """Texto de documentação"""  
    linha1  
    linha2  
    return alguma_coisa
```

"return" indica o retorno da função

# Módulos e Pacotes

# Módulos

Módulos são arquivos de código python que agrupam instruções e definições sobre um determinado assunto

Interface conhecida

# Módulos

```
def cadastra_cliente(nome,email,cpf):  
    cli = {}  
    cli['nome'] = nome  
    cli['email'] = email  
    cli['cpf'] = cpf  
  
    return cli  
  
nome = str(input("Digite seu nome"))  
email = str(input("Digite seu email"))  
cpf = int(input("Qual o número do seu cpf?"))  
  
cli1 = cadastra_cliente(nome, email,cpf)
```



```
def cadastra_cliente(nome,email,cpf):  
    cli = {}  
    cli['nome'] = nome  
    cli['email'] = email  
    cli['cpf'] = cpf  
  
    return cli  
  
def verifica_cadastro_ja_existente(cpf):  
    cpf_clientes_cadastrados = [123,456,678,990,110]  
    if cpf in cpf_clientes_cadastrados:  
        print("Ops! Você já possui cadastro!")  
  
def verifica_email_valido(email):  
    if "@" not in email:  
        print("Esse e-mail não é valido")  
  
nome = str(input("Digite seu nome"))  
email = str(input("Digite seu email"))  
cpf = int(input("Qual o número do seu cpf?"))  
  
cli1 = cadastra_cliente(nome, email,cpf)  
verifica_cadastro_ja_existente(cpf)  
verifica_email_valido(email)
```

# Módulos

```
def cadastra_cliente(nome, email, cpf):  
    cli = {}  
    cli['nome'] = nome  
    cli['email'] = email  
    cli['cpf'] = cpf  
  
    return cli  
  
def verifica_cadastro_ja_existente(cpf):  
    cpf_clientes_cadastrados = [123,456,678,990,110]  
    if cpf in cpf_clientes_cadastrados:  
        print("Ops! Você já possui cadastro!")  
  
def verifica_email_valido(email):  
    if "@" not in email:  
        print("Esse e-mail não é valido")  
  
nome = str(input("Digite seu nome"))  
email = str(input("Digite seu email"))  
cpf = int(input("Qual o número do seu cpf?"))  
  
cli1 = cadastra_cliente(nome, email, cpf)  
verifica_cadastro_ja_existente(cpf)  
verifica_email_valido(email)
```

Jupyter funcoes\_clientes.py ✓ 12 minutos atrás

```
File Edit View Language  
  
1 def cadastra_cliente(nome, email, cpf):  
2     cli = {}  
3     cli['nome'] = nome  
4     cli['email'] = email  
5     cli['cpf'] = cpf  
6  
7     return cli  
8  
9 def verifica_cadastro_ja_existente(cpf):  
10    cpf_clientes_cadastrados = [123,456,678,990,110]  
11    if cpf in cpf_clientes_cadastrados:  
12        print("Ops! Você já possui cadastro!")  
13  
14 def verifica_email_valido(email):  
15     if "@" not in email:  
16         print("Esse e-mail não é valido")
```

```
import funcoes_clientes as fc
```

```
nome = str(input("Digite seu nome"))  
email = str(input("Digite seu email"))  
cpf = int(input("Qual seu cpf?"))
```

```
cli1 = fc.cadastra_cliente(nome, email, cpf)  
fc.verifica_cadastro_ja_existente(cpf)  
fc.verifica_email_valido(email)
```

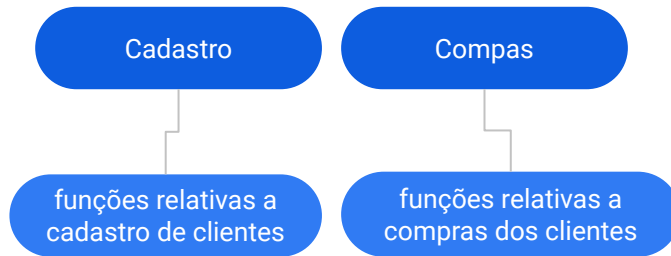
como conectar?

# Pacotes

E se nosso módulo ficasse tão grande que também ficasse difícil mantê-lo?

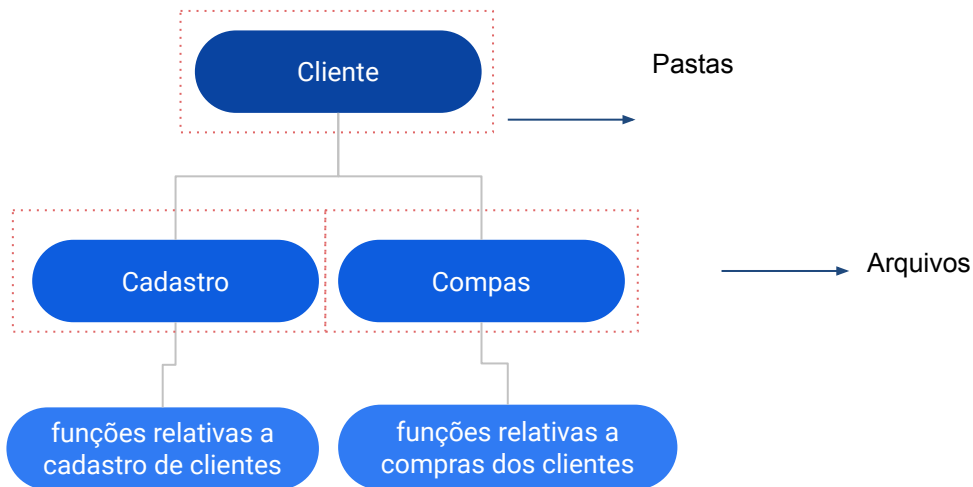
```
def func_1():  
    return ""  
  
def func_2():  
    return ""  
  
def func_3():  
    return ""  
  
def func_4():  
    return ""  
  
def func_5():  
    return ""  
  
def func_6():  
    return ""  
  
def func_7():  
    return ""  
  
def func_8():  
    return ""  
  
def func_9():  
    return ""  
  
def func_10():  
    return ""  
  
def func_11():  
    return ""
```

```
def func_12():  
    return ""  
  
def func_13():  
    return ""  
  
def func_14():  
    return ""  
  
.  
.  
.
```



# Pacotes (Bibliotecas)

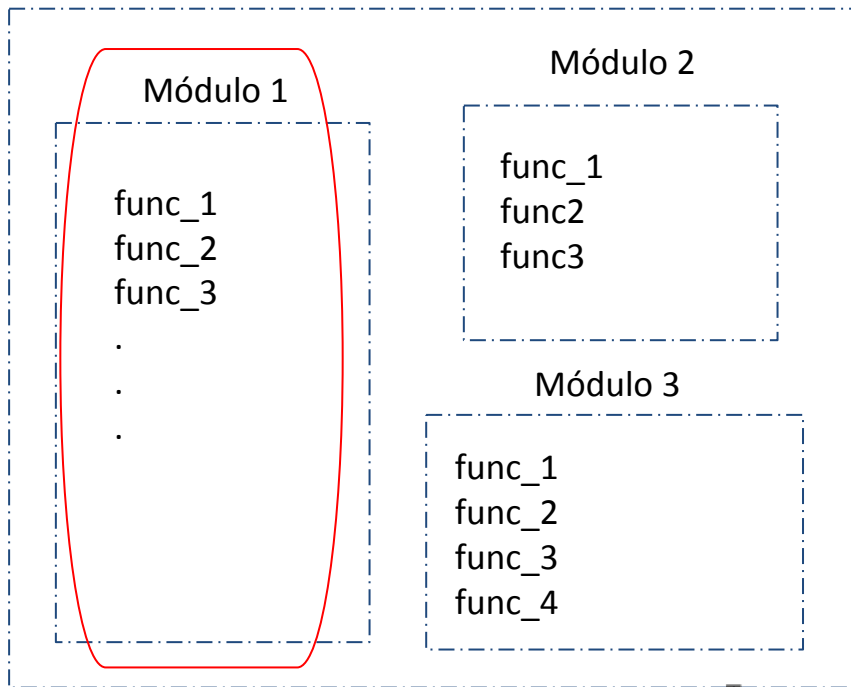
Pacotes são conjuntos de módulos.





# Pacotes (Bibliotecas)

Pacote

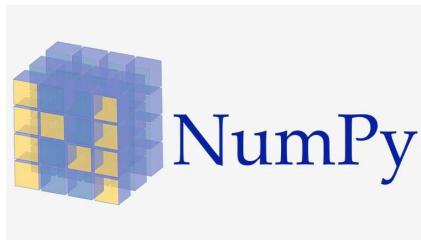


Em python:

```
import pacote
```

```
from pacote import modulol1
```

# Pacotes famosos para Data Science



Oferece uma gama de funções que nos permitem executar facilmente cálculos numéricos



Oferece uma gama de funções para manipulação e análise de dados



Oferece uma gama de funções para criação de gráficos e visualização de dados



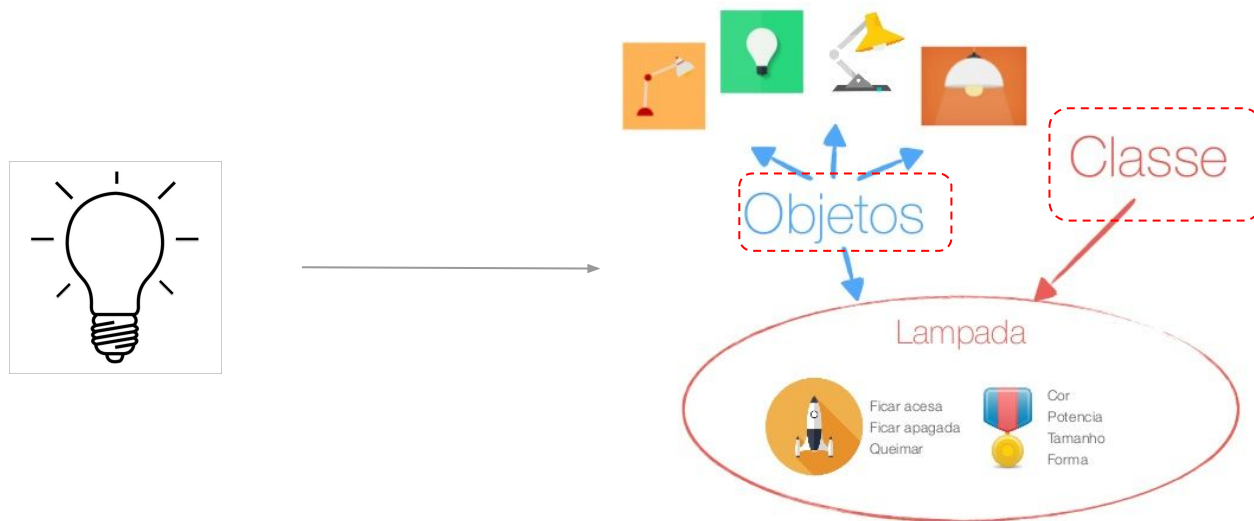
Pacote para aprendizado de máquina

# mentorama.

# Orientação a objetos - POO

# Orientação a objetos - POO

Paradigma de programação no qual o python e outras inúmeras linguagens são baseadas



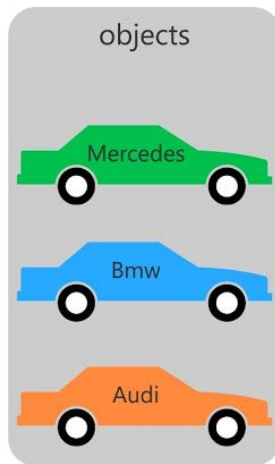
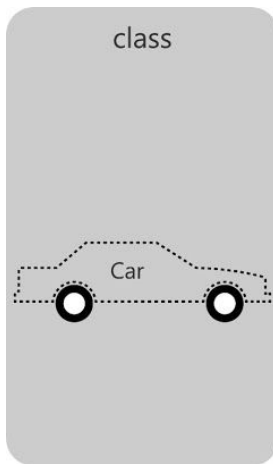
# Classes e Objetos - POO

Objetos: Algo **material ou abstrato**, que pode ser descrito por suas **características e comportamentos**



# Classes e Objetos - POO

Classe: **Molde, receita** que descreve os atributos e métodos de um determinado tipo de objeto (ideia abstrata de um tipo de objeto)

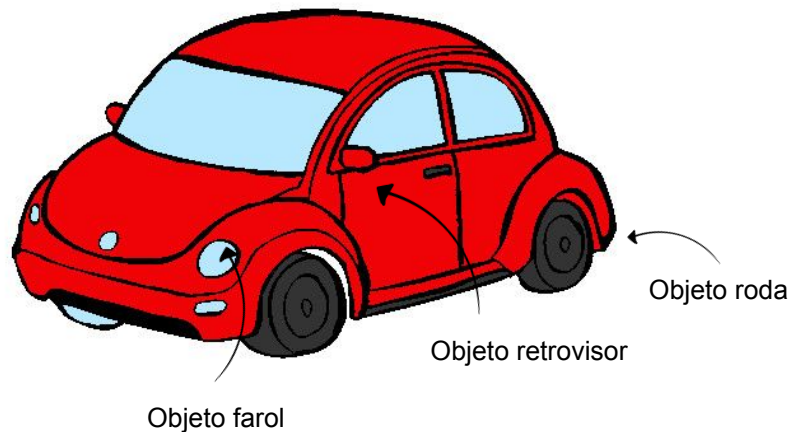


Classe Carro



## Vantagens de - POO

- Conceitos naturais - intuitivo
- Confiável
- Facilita manutenção
- Extensível



# Tratamento de erros e exceções



# Erros e exceções

## Programa

```
valor = 1000

parcelas = int(input("Número de parcelas: "))

valor_da_parcela = valor/parcelas

print("Ok! Nesse caso o valor de cada parcela é:", round(valor_da_parcela,2))
```

## Cenário ótimo

```
valor = 1000

parcelas = int(input("Número de parcelas: "))

valor_da_parcela = valor/parcelas

print("Ok! Nesse caso o valor de cada parcela é:", round(valor_da_parcela,2))
```

Número de parcelas: 3

Ok! Nesse caso o valor de cada parcela é: 333.33

## Cenário erro 1:

```
valor = 1000

parcelas = int(input("Número de parcelas: "))

valor_da_parcela = valor/parcelas

print("Ok! Nesse caso o valor de cada parcela é:", round(valor_da_parcela,2))

Número de parcelas: 0

-----
ZeroDivisionError                                Traceback (most recent call last)
<ipython-input-36-806b25e1033b> in <module>
      4 parcelas = int(input("Número de parcelas: "))
      5
----> 6 valor_da_parcela = valor/parcelas
      7
      8 print("Ok! Nesse caso o valor de cada parcela é:", round(valor_da_parcela,2))

ZeroDivisionError: division by zero
```

## Cenário erro 2:

```
valor = 1000

parcelas = int(input("Numero de parcelas:"))

valor_da_parcela = valor/parcelas

print("Ok! Nesse caso o valor de cada parcela é:", round(valor_da_parcela,2))


Numero de parcelas:nove

-----
ValueError                                Traceback (most recent call last)
<ipython-input-7-40fe677377ef> in <module>
      1 valor = 1000
      2
----> 3 parcelas = int(input("Numero de parcelas:"))
      4
      5 valor_da_parcela = valor/parcelas

ValueError: invalid literal for int() with base 10: 'nove'
```

# Erros e exceções

## Erro sintático



```
print("Ola")
```

-----

**NameError**

<ipython-input-1-b75582997d1f> in <module>  
----> 1 print("Ola")

**NameError:** name 'print' is not defined

## Exceção

```
valor = 1000
```

```
parcelas = int(input("Número de parcelas: "))
```

```
valor_da_parcela = valor/parcelas
```

```
print("Ok! Nesse caso o valor de cada parcela é:", round(valor_da_parcela,2))
```

Número de parcelas: 0

-----

**ZeroDivisionError**

Traceback (most recent call last)

<ipython-input-36-806b25e1033b> in <module>

4 parcelas = int(input("Número de parcelas: "))

5

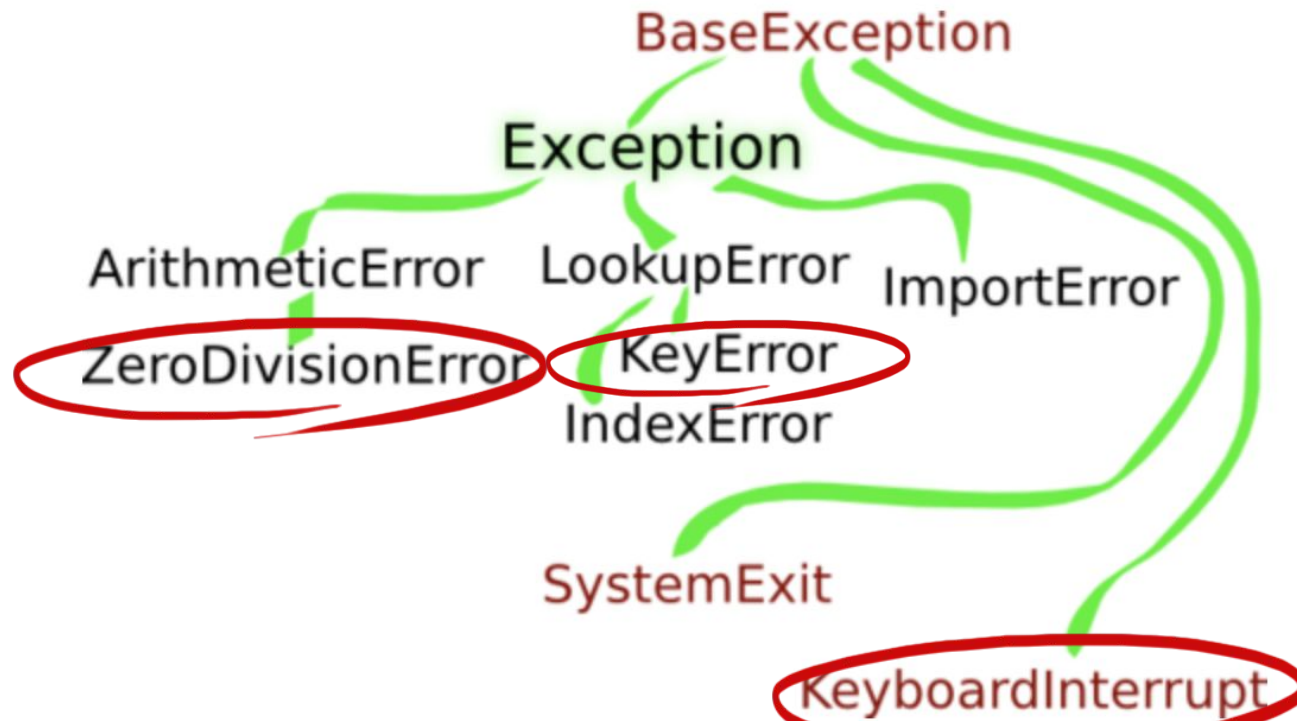
----> 6 valor\_da\_parcela = valor/parcelas

7

8 print("Ok! Nesse caso o valor de cada parcela é:", round(valor\_da\_parcela,2))

**ZeroDivisionError:** division by zero

## Tipos de exceções



# Como tratar exceções?

Try

Operações que prevemos  
que por algum motivo  
podem dar errado

Except

Código a ser executado  
caso a tentativa acima  
falhe

Antes

```
valor = 1000

parcelas = int(input("Número de parcelas: "))

valor_da_parcela = valor/parcelas

print("Ok! Nesse caso o valor de cada parcela é:", round(valor_da_parcela,2))
```

Depois

```
valor = 1000

parcelas = int(input("Numero de parcelas:"))

try:
    valor_da_parcela = valor/parcelas
    print("Ok! Nesse caso o valor de cada parcela é:", round(valor_da_parcela,2))
except:
    print("Algo está errado!")
```

```
Numero de parcelas:10
Ok! Nesse caso o valor de cada parcela é: 100.0
```