Fundamentos de

Python 2

Jéssika Ribeiro

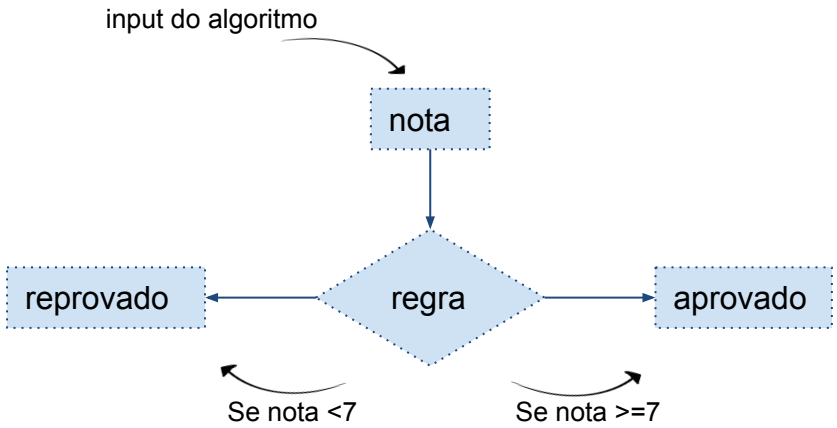
mentorama.

Estruturas de decisão

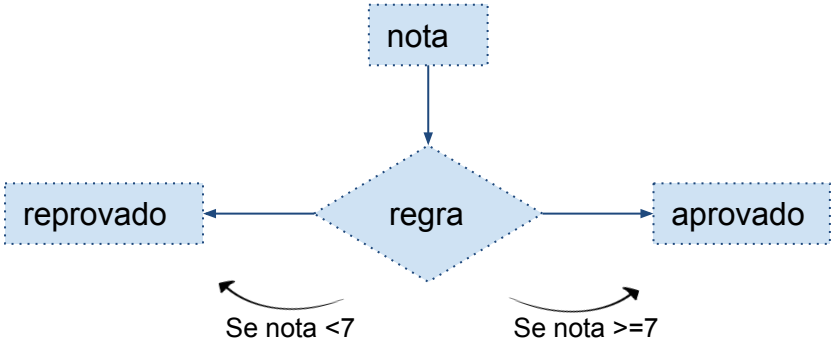
mentorama.

Estruturas de decisão

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



If - Else




```python
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 identado abaixo do if

Senão executa o bloco identado abaixo do else

Estruturas de repetição

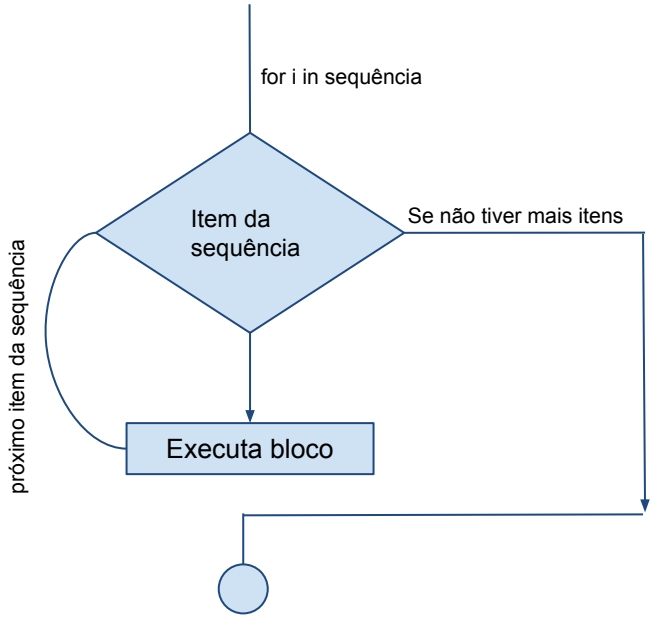
mentorama.

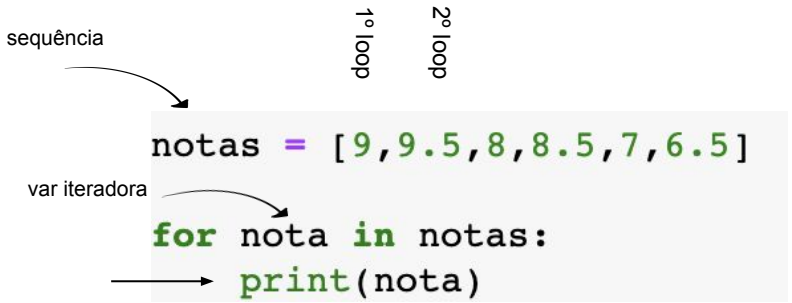
Estruturas de repetição

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

Loop For

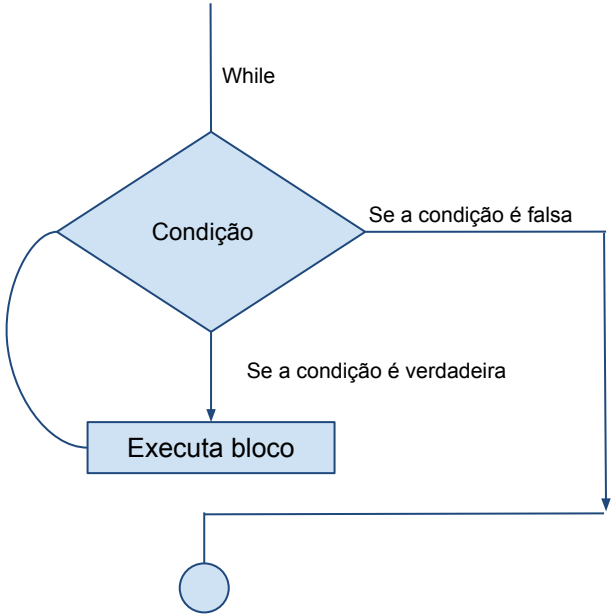
Usado quando queremos executar um bloco de código um número fixo de vezes.





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

mentorama.

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.

VS

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

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

```
Ola Mundo
```

```
a
```

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

```
a
```

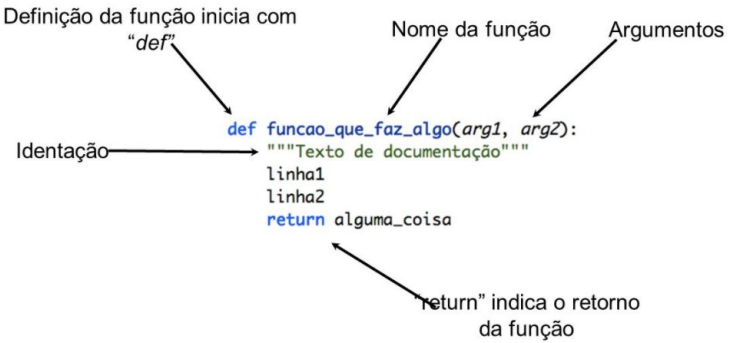
```
9
```

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

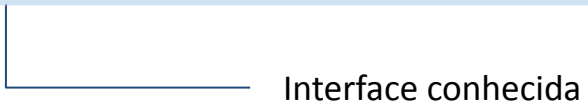


Módulos e Pacotes

mentorama.

Módulos

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



Módulos

```python
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)
```



```python
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

```python
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

```python
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")
```



```python
import funcoes_clientes as fc
```

```python
nome = str(input("Digite seu nome"))
email = str(input("Digite seu nome"))
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?

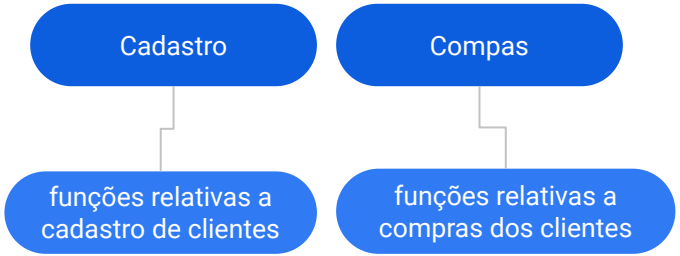
```
def func_1():
    return ""

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 ""
```

.

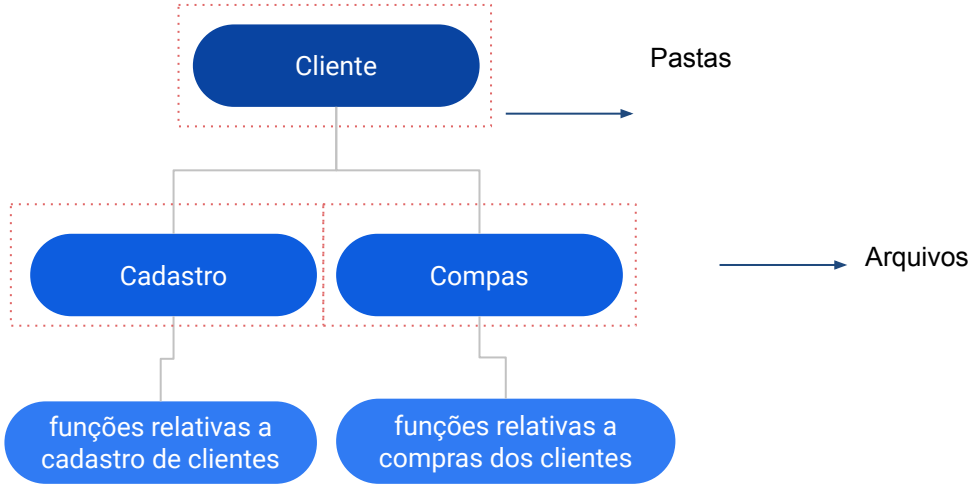
.

.

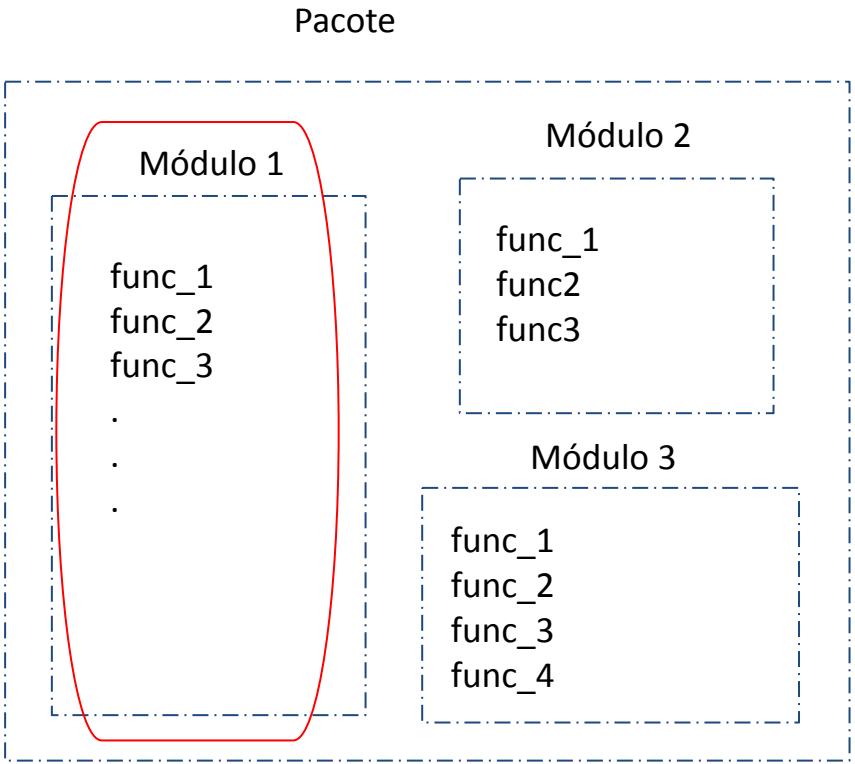


Pacotes (Bibliotecas)

Pacotes são conjuntos de módulos.



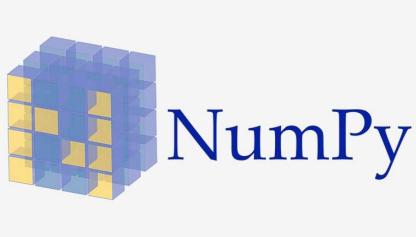
Pacotes (Bibliotecas)



Em python:

```
import pacote


from pacote import modulo1
```

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

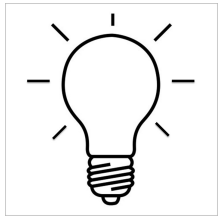
Pacote para aprendizado de máquina

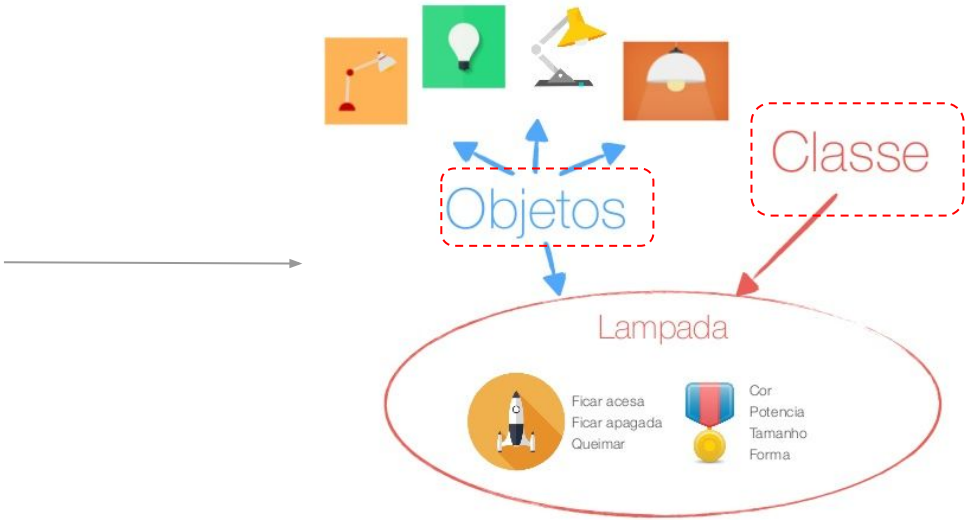
Orientação a objetos - POO

mentorama.

Orientação a objetos - POO

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



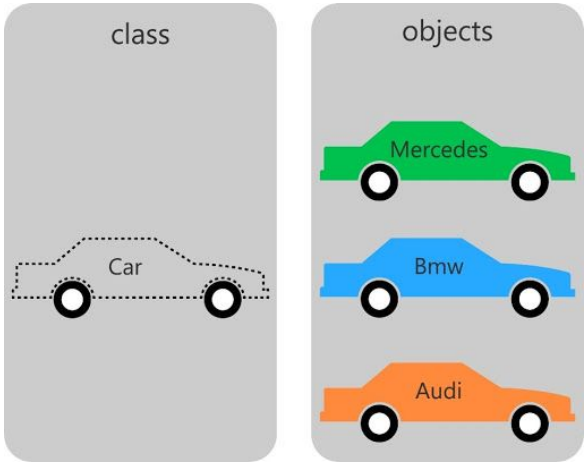


Classes e Objetos - POO

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)



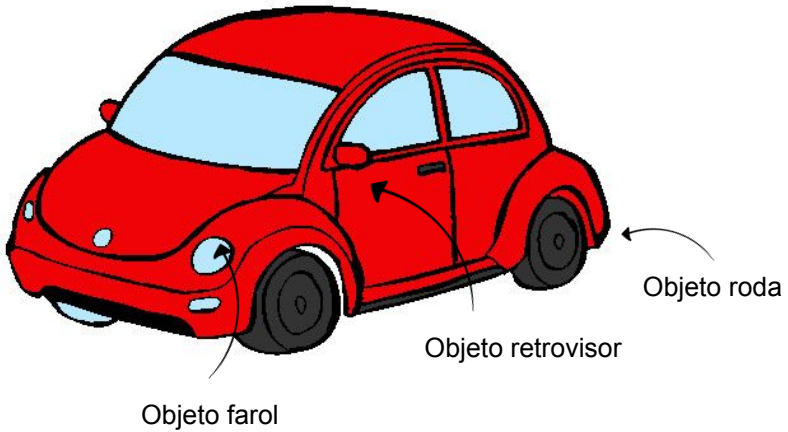
Classe Carro

Modelo
Marca
Ano

Ligar()
Acelerar()
Desligar()

Vantagens de - POO

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



Tratamento de erros e exceções

mentorama.

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

```
primt("Ola")
```

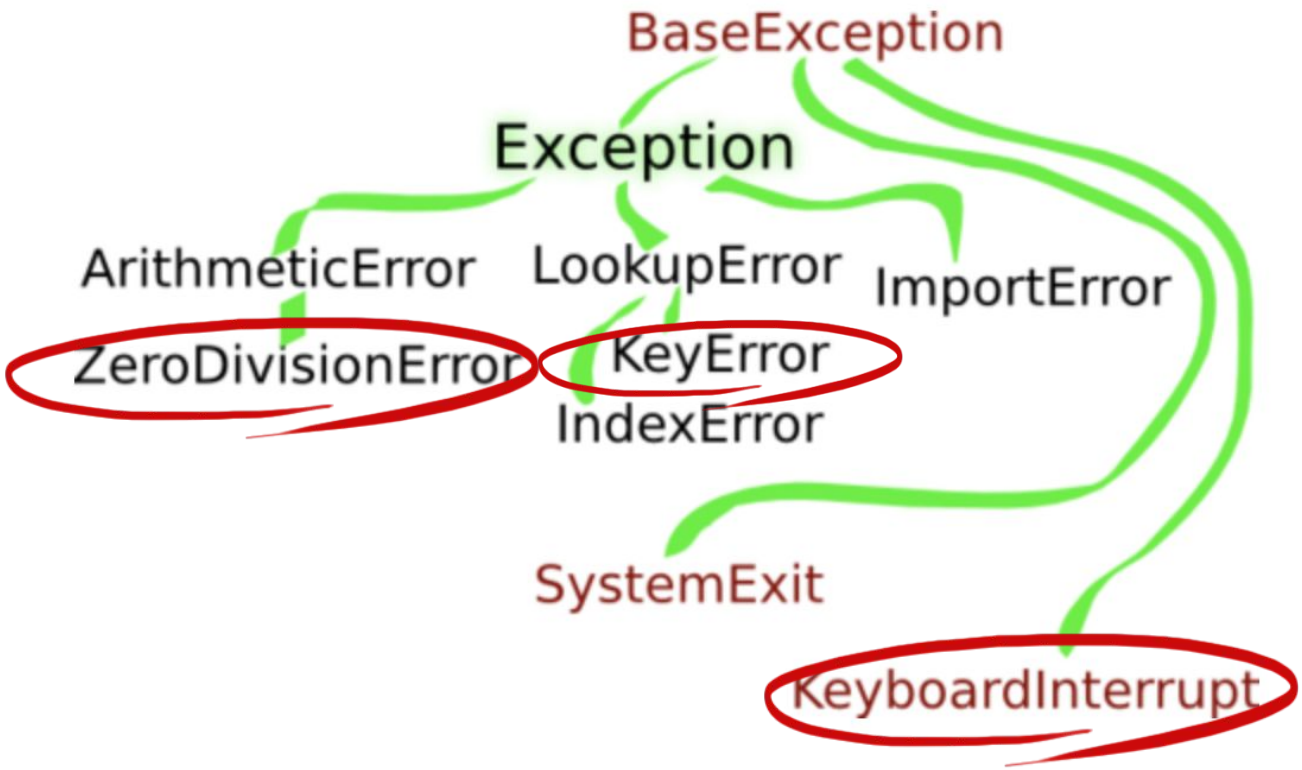
```
---------------------------------------
NameError
<ipython-input-1-b75582997d1f> in <module
----> 1 primt("Ola")

NameError: name 'primt' is not defined
```

Exceção

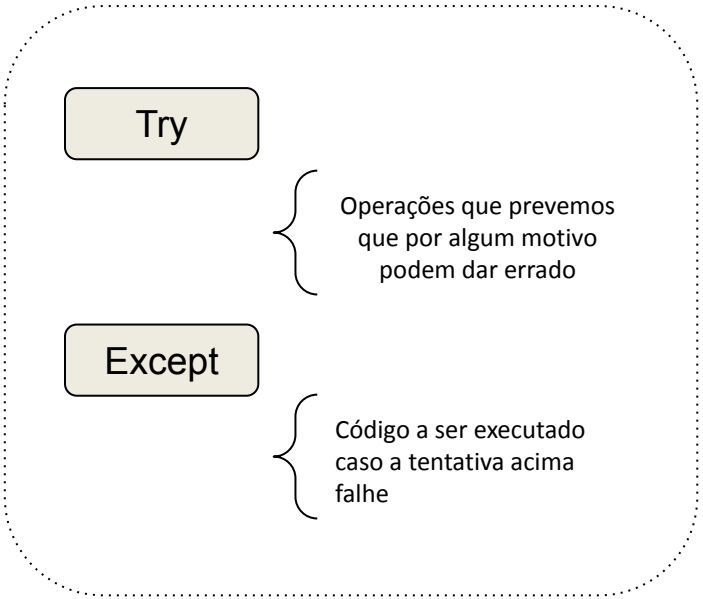
```
valor = 1000

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?



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
```

Tarefa Módulo 3

mentorama.

Tarefa módulo 3

1 - Construa uma função que recebe uma lista de tamanho 5 como parâmetro. Essa função deve percorrer cada elemento dessa lista e verificar se o elemento é par ou não. Se ele for par, salvar ele em uma nova lista que deve ser retornada com todos os valores pares encontrados.

2 - Escreva um código que chama o módulo random. Dentro deste módulo utilize a função que gera números aleatórios inteiros para criar uma função customizada que pede para o usuário digitar quantidade de sorteios que ele quer e roda sorteando essa quantidade de números. Alguns usuários podem não entender que o input é um número e digitar por exemplo "três", o que geraria um erro no código. Então escreva essa função de maneira a tratar esse erro, enviando uma mensagem que ajude o usuário a resolver o problema.

3 - Crie uma classe cliente. Esse cliente deve possuir 2 atributos: nome e saldo e dois métodos: depositar e sacar. Usando a classe, crie um cliente, utilize os métodos criados para depositar 100 reais na conta do cliente, que deve começar com saldo 0 e depois do depósito apresentar saldo 100. Na sequência, faça um saque de 20 reais. Fique atento: o seu código deve impedir que o usuário saque mais do que ele tem em saldo!

Obs: Todos os exercícios devem ser resolvidos no jupyter e vocês devem enviar um arquivo .ipynb como resolução da tarefa.