

Repositório para salvar códigos de PLP



REOs

- [REO 2 - Paradigma Imperativo](#)
- [REO 3 - Paradigma Orientado a Objetos](#)



Reo 2 - Paradigma Imperativo

- [Configurações](#)
 - [Ambiente virtual](#)
 - [Extensões para VsCode](#)
 - [settings.json](#)
- [Aulas](#)
 - [Videoaula de introdução ao Python: GCC198](#)
 - [Paradigma Imperativo : Variáveis e Tipos de Dados](#)
 - [Paradigma Imperativo : Avaliação de Expressões e Controle de Fluxo](#)
 - [Paradigma Imperativo : Subprogramas](#)
- [Atividade Avaliativa](#)

Configurações

- [Ambiente Virtual](#)
 - [Windows](#)

```
py -3 -m venv venv
//NO VENV
pip install wheel
```

- [Linux e Mac](#)

```
python3 -m venv venv
//NO VENV
venv/bin/activate
pip install wheel
deactivate
```

- [Extensões para VSCode:](#)
 - [Code Runner](#)
 - [Python \(Microsoft \)](#)
 - [Python Test Explorer for Visual Studio Code](#)

- Python Preview
- Python Docstring Generator
- settings.json

Na pasta do projeto, crie uma pasta chamada .vscode e dentro dela um arquivo chamado settings.json

- Windows

```
{
  "python.pythonPath": "venv\\Scripts\\python.exe",
  "code-runner.executorMap": {
    "python": "venv\\Scripts\\python.exe",
  },
  "code-runner.ignoreSelection": true,
  "code-runner.runInTerminal": true,
  "python.linting.mypyEnabled": true,
  "python.linting.flake8Enabled": true,
  "python.testing.unittestEnabled": true,
  "[python]": {
    "editor.formatOnSave": true
  }
}
```

- Linux

```
{
  "python.pythonPath": "venv/bin/python",
  "code-runner.executorMap": {
    "python": "venv/bin/python",
  },
  "code-runner.ignoreSelection": true,
  "code-runner.runInTerminal": true,
  "python.linting.mypyEnabled": true,
  "python.linting.flake8Enabled": true,
  "python.testing.unittestEnabled": true,
  "[python]": {
    "editor.formatOnSave": true
  }
}
```

Aulas

- **Videoaula de introdução ao Python: GCC198**
 - [Conditionals](#)
 - [Iterations](#)
 - [Exceptions](#)
 - [Files](#)
 - [CommandArguments](#)

- Extras

- **Conditionals**

- [Vídeo-aula de introdução ao Python : GCC198](#)

```
valor = input("Digite um valor")
valor = int(valor)
if ((valor % 2) == 0):
    print('Número par')
else:
    print('Número ímpar')
```

Digite um número 2

Número par

```
valor = int(input("Digite um valor"))

if valor == 0:
    print('Zero!')
elif valor % 2 == 0:
    print('Número par')
else:
    print('Número ímpar')
```

Digite um número 0

Zero!

```
valor = int(input("Digite um valor"))

msg = 'par' if valor % 2 == 0 else 'ímpar'
print(msg)
```

Digite um número 2

par

- **Iterations**

```
for i in range(10):
    print(i)
```

0
1
2
3
4
5
6
7
8
9

```
for i in range(5,10):  
    print(i)
```

5
6
7
8
9

```
blacklist = ["palavrão", "palavrona", "palavreadão"]  
for palavra in blacklist:  
    print("Palavra proibida: {}".format(palavra))
```

Palavra proibida: palavrão
Palavra proibida: palavrona
Palavra proibida: pavreadão

```
blacklist = ["palavrão", "palavrona", "palavreadão"]  
texto = input("Digite uma frase: ")  
palavras = texto.split()  
for palavra in palavras:  
    if palavra.lower() in blacklist:  
        print("A palavra {} é proibida!".format(palavra))
```

Digite uma frase: Oi, palavrão tal

A palavra palavrão é proibida!

◦ Exceptions

```
try:
    valor = int(input("Digite um número"))
    if valor == 0:
        print('Zero!')
    elif valor % 2 == 0:
        print('Número par')
    else:
        print('Número ímpar')
except:
    print('Valor digitado não é um número!')
```

Digite um número a

Valor digitado não é um número!

◦ Files

dataset.csv :

7,8,9

3,4,5

2,4,1

90,89,20

8,4,12

```
xs = []
ys = []
zs = []
with open('dataset.csv','r') as file:
    lines = file.readlines()
    for line in lines:
        x, y, z = line.split(',')
        xs.append(x)
        ys.append(y)
        zs.append(z)

print(xs)
print(ys)
print(zs)
```

```
['7', '3', '2', '90', '8']  
['8', '4', '4', '89', '4']  
['9\n', '5\n', '1\n', '20\n', '12']
```

```
xs = []  
ys = []  
zs = []  
with open('dataset.csv', 'r') as file:  
    lines = file.readlines()  
    for line in lines:  
        x, y, z = line.split(',')  
        xs.append(x)  
        ys.append(y)  
        zs.append(z.strip())  
  
print(xs)  
print(ys)  
print(zs)
```

```
['7', '3', '2', '90', '8']  
['8', '4', '4', '89', '4']  
['9', '5', '1', '20', '12']
```

◦ Command Arguments

- python_example1.py

```
import sys  
  
caminho_do_arquivo = sys.argv[1] #o sys.argv[0] é o nome do  
próprio arquivo python  
  
xs = []  
ys = []  
zs = []  
  
with open(caminho_do_arquivo, 'r') as file:  
    lines = file.readlines()  
    for line in lines:  
        x, y, z = line.split(',')  
        xs.append(x)  
        ys.append(y)  
        zs.append(z.strip())  
  
print(xs)  
print(ys)  
print(zs)
```

- A chamada é feita por terminal de comando

```
python3 python_example1.py dataset.csv
```

```
['7', '3', '2', '90', '8']  
['8', '4', '4', '89', '4']  
['9', '5', '1', '20', '12']
```

- **Extras**

```
numeros = [1, 10, 100, 1000, 2, 20, 200]  
print(sum(numeros))  
print(max(numeros))  
print(min(numeros))
```

```
1333
```

```
1000
```

```
1
```

```
numeros = [1, 10, 100, 1000, 2, 20, 200]  
print("Média da lista", sum(numeros)/len(numeros))
```

```
190.42857142857142
```

- **Paradigma Imperativo : Variáveis e Tipos de Dados**

- [Vídeo-aula Paradigma Imperativo : Variáveis e Tipos de Dados](#)
- [Slide Variáveis e tipos de dados](#)

```
my_string = 'Hello, World!'  
myflt = 45.06  
mybool = 5 > 9  
my_list = ['item1', 'item2']  
my_tuple = ('item1', 'item2')  
my_dict = {'letter': 'g', 'number': 7}
```

- **Paradigma Imperativo : Avaliação de Expressões e Controle de Fluxo**

- [Paradigma Imperativo : Avaliação de Expressões e Controle de Fluxo](#)
- [Avaliações de expressões e controle de fluxo](#)

- **Paradigma Imperativo : Subprogramas**

- Paradigma Imperativo : Subprogramas
- Subprogramas

```
def subprograma(a,b,c):  
    print(a + b + c)
```

- Recursao

```
def fatorial(n):  
    if (n <= 1):  
        return 1  
    else:  
        return (n * fatorial(n-1))
```

Atividade Avaliativa

- Sobre
 - Programa pra rodar o coeficiente da correlação de Pearson
- Rodar

```
py -3 atividade_avaliativa_Rafael_Porto_reo2 teste.txt
```

Reo 3 - Paradigma Orientado a Objetos

- Aulas
 - **Paradigma Orientado a Objetos : Conceitos iniciais**
 - **Paradigma Orientado a Objetos : Encapsulamento**
 - **Paradigma Orientado a Objetos : Herança e Composição**
 - **Paradigma Orientado a Objetos : Polimorfismo**
- Python
- Atividade Avaliativa

Paradigma Orientado a Objetos : Conceitos iniciais

[Video Aula](#)

Conjunto de princípios

- Orientam a criação de sistemas computacionais, objetos que interagem entre si.

Em termos de LPs, conceitos formais surgem com Simula 67, sendo consolidados com Smalltalk (primeira linguagem orientada a objetos).

Popularizado com a difusão de interfaces gráficas de usuários (GUIs)

- Surgimento de ferramentas com suporte para desenvolvimento de aplicações gráficas (C++, FoxPro, Delphi).

Suportado por várias linguagens (ex: Python, Ruby, C#)

- Atualmente sua maior expressão comercial é dada pelo Java

Pilares da OO

Conceitos fundamentais (pilares) que norteiam o desenvolvimento OO:

- Abstração;
- Encapsulamento;
- Herança;
- Polimorfismo.

Abstração

Representação de uma entidade do mundo real, com seu comportamento e características.

"Modelos Mentais"

- Classes;
- Objetos;
- Métodos;
- Atributos;

Classes: Uma classe pode ser entendida como um módulo ou uma estrutura de dados abstrata.

Uma visão mais ampla pode levar à seguinte definição:

- Uma classe é um tipo abstrato de dados, que reúne objetos com características similares.
- O comportamento destes objetos é descrito pelo conjunto de métodos disponíveis.
- O conjunto de atributos da classe descrevem as características de um objeto.

Classes

Considere a definição de uma classe Pessoa. Existem diversas pessoas, e cada um deles se diferencia pelo nome e cpf.

```
class Pessoa:
    def __init__(self, cpf, nome):
        self.cpf = cpf
        self.nome = nome
```

Nome da classe

Método construtor

Atributos

Objetos:

Um objeto pode ser entendido como um ser, lugar, evento, coisa ou conceito do mundo real que possa ser aplicável a um sistema.

É comum que haja objetos diferentes com características semelhantes. Esses objetos são agrupados em classes.

Classes são um agrupamento de objetos com características similares!

Objetos são entidades (instâncias) únicas de uma classe!

Objetos

Considere a classe Pessoa podemos ter um objeto p1, com cpf 123.456.789-10 e nome João da Silva.

variável que armazena
objeto criado

Instanciação (criação) de um objeto da
classe Pessoa

```
p1 = Pessoa('123.456.789-10', 'João da  
Silva')
```

Atributos:

Um atributo é uma característica de um grupo de entidades do mundo real, agrupados em uma classe.

Um atributo pode ser um valor simples (um inteiro, por exemplo) ou estruturas complexas (um outro objeto, por exemplo).

Atributos

Considere a classe Pessoa. Seus atributos são o cpf e o nome.

```
class Pessoa:
    def __init__(self, cpf, nome):
        self.cpf = cpf
        self.nome = nome
```

Atributos

Atributos de classe

- Em geral, os atributos pertencem a cada objeto instanciado, ou seja, a cada nova instanciãção de uma mesma classe, cada instância pode ter valores distintos para cada atributo.
- Atributos de classe são definidos para terem o mesmo valor para todas as instâncias de uma classe.

Exemplo: Atributos de classe

```
class Pessoa:
    __total_pessoas = 0

    def __init__(self, cpf, nome):
        self.cpf = cpf
        self.nome = nome
        Pessoa.__total_pessoas += 1

    def get_total_pessoas(self):
        return Pessoa.__total_pessoas
```

Atributos de instância. Esses atributos receberão valores distintos a cada instanciãção

Em Python, atributos de classe devem ser definidos com dois 'underlines' como prefixo do nome da variável

Forma de acessar atributo de classe

```
p1 = Pessoa('123.456.789-10', 'Fulano Ciclano', '01/02/1995')
print(p1.get_total_pessoas()) # erro
print(Pessoa.get_total_pessoas(p1)) # OK
```

Métodos

Semelhante a uma função, é a implementação de uma ação da entidade representada pela classe; Conjunto de métodos define o comportamento dos objetos de uma classe.

Métodos

```
from datetime import datetime
```

```
class Pessoa:
```

```
    def __init__(self, cpf, nome, data_nascimento):  
        d, m, a = data_nascimento.split("/")  
        self.cpf = cpf  
        self.nome = nome  
        self.data_nascimento = datetime(a, m, d)
```

Método que retorna a data de nascimento de uma pessoa. O *self* indica que o atributo pertence ao objeto. Semelhante ao *this* do Java

```
    def get_data_nascimento(self):  
        return self.data_nascimento.strftime("%x")
```

Construtores

É um método especial para a criação e inicialização de uma nova instância de uma classe.

Um construtor inicializa um objeto e suas variáveis, cria quaisquer outros objetos de que ele precise, garantindo que ele seja configurado corretamente quando criado.

Na maioria das LPs, o construtor é um método que tem o mesmo nome da classe, que geralmente é chamado quando um objeto da classe é declarado ou instanciado.

Exemplo: Construtores

```
from datetime import datetime
```

```
class Pessoa:
```

```
    def __init__(self, cpf, nome, data_nascimento):  
        d, m, a = map(int, data_nascimento.split("/"))  
        self.cpf = cpf  
        self.nome = nome  
        self.data_nascimento = datetime(a, m, d)
```

Esse construtor recebe 3 parâmetros reais, o primeiro (não contado aqui), indicado como *self*, serve como referência para o próprio objeto.

Exemplo: Construtores

Imagine o caso de um prontuário médico, poderíamos ter, adicionalmente, as seguintes classes:

```
class Medicamento:
```

```
    def __init__(self, nome):  
        self.nome = nome
```

```
class Prontuario:
```

```
    def __init__(obj, paciente):  
        obj.paciente = paciente  
        obj.medicamentos = []
```

Observe que, ao invés de usar *self*, foi utilizado *obj* como referência ao objeto. Não se trata de uma palavra reservada, mas do primeiro argumento de um método do objeto. Costuma-se utilizar *self*.

Destrutores

De forma similar aos construtores, os destrutores são métodos fundamentais das classes, sendo geralmente chamados quando termina o tempo de vida do objeto.

Em algumas linguagens como C++, ocupam um papel tão importante quanto os construtores, por conta da necessidade de desalocação de memória.

Em outras linguagens como Java, o Garbage Collector (Coletor Automático de Lixo) faz esse papel, desalocando aquilo que não é mais utilizado. Há o método `finalize()`, mas raramente é utilizado (há dúvidas se sempre funciona, inclusive).

Tanto os construtores, quanto os destrutores são métodos que não precisam ser definidos em Orientação a Objetos em Python, caso o comportamento esperado seja o padrão.

Geralmente, define-se o construtor para a passagem de argumentos na criação do objeto. Já o destrutor não se costuma definir.

Caso seja necessário realizar algum procedimento na destruição do objeto, define-se o método destrutor, como será exemplificado.

Destrutores

```
p1 = Pessoa('123.456.789-10', 'Fulano Ciclano', '01/02/1995')
p2 = Pessoa('123.456.789-11', 'Ciclano Fulano', '31/12/1996')

del p1 # nesse caso, o objeto instanciado é desalocado da memória (destruído)
del p2 # nesse caso, o objeto instanciado é desalocado da memória (destruído)
```

Destrutores

```
class A:
    def __init__(self):
        print ("A has been created")

    def __del__(self):
        print ("A has been destroyed")

a = A()
del a
```

Método destrutor dos objetos da classe A.

Quando o objeto a for destruído, a mensagem definida no destrutor será exibida

Garbage Collection em Java

- Em C++ a memória é alocada e desalocada explicitamente
- Java possui gerenciamento automático de memória, realizado pela JVM
 - Evita vazamento de memória e bugs de ponteiros
 - Consome recursos computacionais quanto à decisão de desalocação
 - É um processo "não determinístico"

Paradigma Orientado a Objetos : Encapsulamento

[Video Aula](#)

Paradigma Orientado a Objetos : Herança e Composição

Paradigma Orientado a Objetos : Polimorfismo

Paradigma Orientado a Objetos : Python

Conceitos Iniciais:

Definição de classe:

```
class Pessoa:
    def __init__(self, cpf, nome):
        self.cpf = cpf
        self.nome = nome
```

Objetos:

```
p1 = Pessoa('123.456.789-10', 'João da Silva')
```

Atributos são o `self.cpf` e `self.nome`

Atributos de classe nesse exemplo seria o `__total_pessoas`

```
class Pessoa:
    __total_pessoas = 0
    def __init__(self, cpf, nome):
        self.cpf = cpf
        self.nome = nome
        Pessoa.__total_pessoas += 1
    def get_total_pessoas(self):
        return Pessoa.__total_pessoas

p1 = Pessoa('123.456.789-10', 'Bissexto')
print(p1.get_total_pessoas()) #erro
print(Pessoa.get_total_pessoas(p1)) #OK
```

Métodos

```
from datetime import datetime
class Pessoa:
    def __init__(self, cpf, nome, data_nascimento):
        d, m, a = data_nascimento.split("/")
        self.cpf = cpf
        self.nome = nome
```

```
self.data_nascimento = datetime(a,m,d)

def get_data_nascimento(self):
    return self.data_nascimento.strftime("%x")
```

Construtores

```
class Medicamento:
    def __init__(self, nome):
        self.nome = nome
```

Destrutores

```
class A:
    def __del__(self):
        print("A has been destroyed")
```