PARADIGMA DA ORIENTAÇÃO A OBJETOS:

Até o momento estudamos os paradigmas imperativo, procedural e funcional. Esses paradigmas de programação estão alinhados com a construção de códigos sequenciais onde as decisões são tomadas muitas vezes com a utilização de processos iterativos. Segundo Menezes (2019), a **P**rogramação **O**rientada a **O**jbetos é um paradigma de programação que organiza nossos programas em Classes e Objetos em vez de apenas funções.

A **P**rogramação **O**rientada a **O**bjetos é utilizada para mapear objetos do mundo real para modelos computacionais. Suas principais características são: **abstração**, **encapsulamento**, **herança** e **polimorfismo.**

ABSTRAÇÃO: é uma maneira de representar um objeto dentro de um programa. Essa representação consiste em modelar o que esse objeto irá realizar dentro de nosso programa. Nessa abstração devemos considerar três pontos principais que são: identidade, propriedade e método.

Identidade: deve ser única dentro do programa.

Propriedades referem-se às características que o objeto possui. No mundo real qualquer objeto possui elementos que o definem. Dentro dos nossos modelos com **P**rogramação **O**rientada a **O**bjetos, essas características são nomeadas propriedades. Por exemplo as propriedades de um objeto "Elefante" poderiam ser "Tamanho", "Peso" e "Idade".

Métodos: são as ações do objeto. Esses métodos podem ser os mais diversos possíveis, como, por exemplo, para o elefante temos comer(), andar() e dormir(), (Menezes, 2019 e F. V. C. 2020).

ENCAPSULAMENTO: "...está relacionado à segurança de funcionamento da representação de objetos pelo fato de esconder as propriedades, criando uma espécie de caixa preta. Dessa maneira, detalhes internos de uma classe podem permanecer ocultos e/ou protegidos para um objeto. As linguagens orientadas a objetos implementam o encapsulamento baseado em propriedades privadas. Essa atitude evita o acesso direto à propriedade do objeto, adicionando uma camada a mais de segurança à aplicação". (F. V. C., 2020, p. 201)

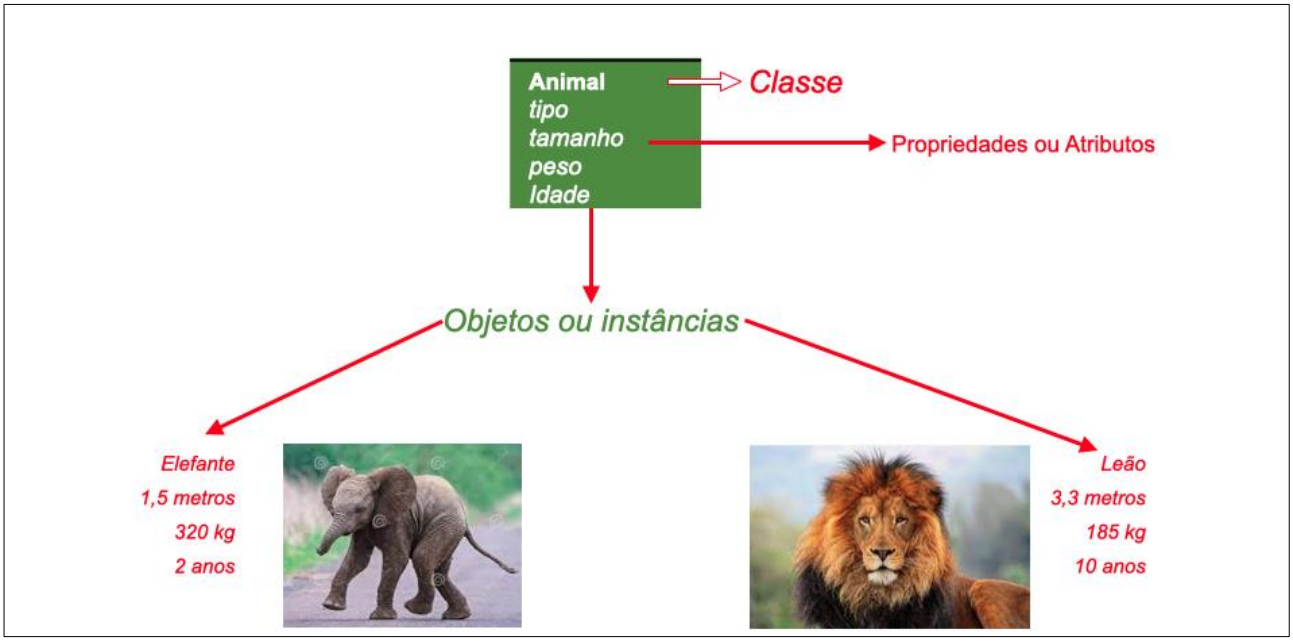
POLIMORFISMO: permite que os objetos que herdam características possam alterar seu funcionamento interno a partir de métodos herdados de um objeto pai. C., 2020, p. 201)

Partindo desse raciocínio, podemos concluir que um objeto pode ser entendido como uma variável cujo tipo é uma classe, ou seja, um **objeto** é uma **instância** de uma **Classe** Menezes (2019, pg. 233) e **Classes** são como os **moldes** formadores dos **objetos**. O modo de abstração de um fenômeno a partir de um objeto é eficaz para modelar seu funcionamento.

Na Programação Orientada a Objetos, todo objeto apresenta pelo menos duas características principais: os **atributos** e **métodos**. Os atributos estão relacionados às *propriedades estáticas* de um objeto, já os métodos tentam representar as *ações* que podem ser admitidas para o objeto, F. V. C. (2020) e Menezes (2019),

PRATICANDO:

Desenvolva um módulo que tenha uma classe chamada Animal.
- Por convenção, o nome de uma classe deve começar com letra maiúscula;
- A palavra reservada class em Python é utilizada para criar classes;
- Ao declarar uma classe, criamos um molde para a partir dele criar novos objetos;
- Dentro das classes especificaremos os métodos e atributos dos objetos da classe;



- Orientação a Objetos

1 - Criando uma classe

```
class Animal:
    def __init__(self, tipo, tamanho = None, peso = None, idade = None):
        self.tipo = tipo
        self.tamanho = tamanho
        self.peso = peso
        self.idade = idade
```

1 – Define a classe Animal:

A palavra reservada class é utilizada para criar classes. Quando estamos declarando uma classe seu nome sempre deve começar com letra maiúscula, estamos criando um molde que possibilitará a criação de novos objetos. Dentro das classes devemos especificar os métodos e atributos que serão concedidos aos objetos da classe. F. V. C. (2020)

2 – define o método construtor da classe animal:

A diretiva **__init__** define um método que será chamado sempre que precisarmos instanciar (criar) um novo objeto a partir de uma Classe. Esse método é chamado de **método construtor**, que inicia nosso objeto com alguns valores padrão. F. V. C. (2020). Ainda nessa linha o parâmetro **self representa** o objeto animal.

3 – o parâmetro self.tipo é um atributo do objeto que está recebendo o valor da variável tipo.

4 – o parâmetro self.tamanho é um atributo do objeto que está recebendo o valor da variável tamanho.

5 – o parâmetro self.peso é um atributo do objeto que está recebendo o valor da variável peso.

6 – o parâmetro self.idade é um atributo do objeto que está recebendo o valor da variável idade.

 # 1.1 - Definindo o método comer() da classe Aninal

```
def comer(self):
    return print('O ', self.tipo, ' está comendo!')
```

É dentro das classes que especificamos os métodos e atributos que serão concedidos aos objetos da classe. O self é uma referência a cada atributo de um objeto criado a partir da classe. Observe que na classe Animal criamos o método o comer(). Uma vez instanciado um objeto nós teremos acesso aos atributos tipo, tamanho, peso, idade bem como do método 'comer'. F. V. C. (2020).

 # 2 - Instanciando (criando) o objeto animal chamado elefante passando seus argumentos.

```
elefante = Animal('Elefante', 3, 2700, 30)
```

 # 3 - Apresentando na tela os atributos do objeto elefante recebidos da classe Animal

```
print('\nTipo do animal: ', elefante.tipo)
print('Tamanho: ',elefante.tamanho, ' metros de altura')
print('Peso: ',elefante.peso, '  quilos')
print('Idade: ',elefante.idade, ' anos ')
print(f'Ele é da classe {type(elefante)}')
```

```
Tipo do animal:  Elefante
Tamanho:  3  metros de altura
Peso:  2700   quilos
Idade:  30  anos
Ele é da classe <class '__main__.Animal'>
```

Observação: Quando estamos planejamento um software e definimos quais classes que o sistema terá, chamamos **cada classe criada de objeto** e estes serão chamados de **entidade**.

PRINCIPAIS ELEMENTOS DA ORIENTAÇÃO A OBJETOS

- **Classe**: = modelo do objeto do mundo real – representado computacionalmente
- **Atributo:** = características do objeto;
- **Construtor**: método especial utilizado para definir (criar) os objetos;
- **Método:** = comportamento do Objeto;
- **Objeto**: = "...pode ser entendido como uma variável cujo tipo é uma classe, ou seja, um objeto é uma instância de uma classe". Menezes (2019, página 222)

PROGRAMAÇÃO ORIENTADA A OBJETOS: "...é uma técnica de programação que organiza nossos programas em classes e objetos em vez de apenas funções. (MENEZES, 2019, página (222)".

CLASSES: são modelos dos objetos do mundo real representados computacionalmente, e devem conter: Atributo; Construtor e Métodos. Segundo Menezes (2019 – página 222), "Classes são a definição de um novo tipo de dados que associa dados e operações em uma só estrutura".

 # 4 - **Exercício:** desenvolver em um script uma classe chamada Lampada. O objetivo será automatizar o controle das lâmpadas da uma casa, seguindo as seguintes premissas e critérios:

- Partindo da premissa que os atributos representam as características do objeto. Então no caso das lâmpadas precisamos saber se a lâmpada é 110v ou 220 volts ou bivolt;

se ela é branca, amarela ou vermelha, se a tecnologia é filamento, led, florescente, além da luminosidade e seu status: se está ligada ou desligada.

- Se os métodos (com exceção do método construtor) representam os comportamentos do objeto, ou seja, as ações que este objeto pode realizar no seu sistema: No caso da lâmpada qual o comportamento que ela terá?

- Por convenção o nome das classes inicia em Maiúsculo;

- Se o nome for composto utiliza-se as iniciais de ambas as palavras em Maiúsculos todas juntas **CamelCase**.

Instancie um objeto chamado lampada_sl e apresente o valor seus atributos;

Instancie um objeto chamado lampada_qt e apresente o valor de seus atributos;

Instancie um objeto chamado lampada_cz e apresente o valor de seus atributos;

ATRIBUTOS: são as características estáticas de um objeto. Através deles representamos computacionalmente os estados de um objeto, como no exemplo da lâmpada que tinha os atributos: voltagem, cor, tipo, lumens...

Em Python dividimos os atributos em **três grupos**:

- Atributos de instância;
- Atributos de classe;
- Atributos dinâmicos.

Para compreendermos **atributos de instância**, precisamos entender a base conceitual de método. Em Python, métodos tentam representar a dinâmica das ações que o objeto pode ter (F. V. C. 2020).

Quanto estamos definindo (criando) uma classe que será o molde para a construção de objetos temos que iniciá-la com o **método construtor**. Em Python o **Método construtor** é um método especial utilizado para a construção do molde (classe) do objeto.

Atributo de Instância: são atributos declarados dentro do método construtor da classe, usando a diretiva **__init__** que irá determinar que este é um método especial. O método **__init__** será chamado sempre que estamos instanciando (criando) um determinado objeto a partir de uma classe.

 # 05 - Atributos de instâncias - privados – classe Lampada

```
class Lampada:

    def __init__(self, voltagem, cor, status):
        self.__voltagem = voltagem
        self.__cor = cor
        self.__status = 'Ligada'
```

O duplo *underline* depois da palavra reservada ***self.***__ indica que o atributo é privado e só pode ser acessado dentro da classe.

Caso utilize o comando ***self.***atributo sem o duplo *underline*, estará sendo definido um **atributo público**. Em Python por **convenção** foi estabelecido que todo atributo de instância de uma classe é público, desde que não possua o duplo *underline* em sua definição.

 # 06 - **Atributos de instâncias** - públicos – classe Lampada

```
class Lampada:

    def __init__(self, voltagem, cor, status):
        self.voltagem = voltagem
        self.cor = cor
        self.status = 'Ligada'
```

Todo objeto que tenha um atributo público pode ser acessado por qualquer módulo do projeto. Também, por convenção, o primeiro parâmetro de um método é a palavra reservada ***self.***, que é considerado o próprio objeto.

A palavra reservada ***self*** usada como parâmetro do método construtor ou utilizada como ligação do atributo, exemplo (***self*.voltagem**), é uma referência a cada atributo de um objeto criado a partir da classe, ou como costumamos falar em Python, é o objeto que está executando o método.

Quando fazemos o acesso dentro do método usando por exemplo: **self.cor** = cor e partindo da premissa que em Python ***self*** é o próprio objeto, estamos dizendo ao Python que o objeto Lampada no atributo cor irá receber o valor da variável cor.

Quando instanciamos um objeto, estamos executando o método construtor. Ex: lâmpada = **Lampada()**, neste caso é o método construtor **__init__** da classe **Lampada()** quem está sendo executado.

Seguindo o exemplo acima e partindo da premissa que "*Será na fase de planejamento do software que definiremos quais classes teremos*" defina (crie) as seguintes classes abaixo e de acordo com o nome sugerido para cada classe defina o método construtor e os atributos (públicos) de instância das classes:

 # 07 - ContaCorrente

 # 08 - Produto

 # 09 - Usuario

VISIBILIDADE DOS ATRIBUTOS DE INSTÂNCIA

Atributos de instância público: por *default*, quando declaramos um atributo de instância dentro de uma classe ele terá visibilidade pública e poderá ser acessado por qualquer módulo do projeto. As classes que definimos nos exercícios tem o atributo de instância **PÚBLICO**.

Atributos de instância privado: para definir um atributo de instância como privado digitamos o **dunder** (*double underline* __) no início de seu nome. O **dunder** determina que o atributo só deve ser utilizado dentro da própria classe. Essa ação é conhecida como ***name Mangling***.

 # 10 - Definindo atributos de instância privados de uma classe

```
class LoginIntranet:

    def __init__(self, email, senha):
        self.__email = email
        self.__senha = senha
```

Por convenção o Python **não irá impedir** que façamos acesso aos atributos definidos como privados fora da classe, exemplo:

 # 11 – tentando acessar um atributo de instância privado de fora da classe

```
usuario = LoginIntranet('teste@gmail.com', '123456') # instanciando o objeto

print(usuario.__email)
print(usuario.__senha)
```

```
Traceback (most recent call last):
  File "/Users/Claudinei/Documents/01 - IFRO 2022/06 - Disciplin
2022 - POO/30 - modulos_customizados_externos_dunder.py",
    print(usuario.__email)
AttributeError: 'LoginIntranet' object has no attribute '__email'
```

Atenção: Esse é um nível de segurança do Python para informar o desenvolvedor que ele, não deve fazer acesso dessa maneira a um atributo de instância privado.

 # 12 - verificando as classes que estão relacionadas ao objeto **usuario** instanciado

```
print(dir(usuario))
```

```
['_LoginIntranet__email', '_LoginIntranet__senha', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__']
```

O Python apresentou **_LoginIntranet__email** e **_LoginIntranet__email** entre outros relacionamentos. *Underline Login_Intranet é a nossa classe e duplo underline email ou duplo underline senha* é o atributo de instância privado da classe.

 # 13 – Apresentando o valor de um atributo de instancia privado - ***Name Mangling***

```
print(f'\n O e-mail do usuário é: {usuario._LoginIntranet__email}')
print(f'\n A senha do usuário é: {usuario._LoginIntranet__senha}')
```

```
O e-mail do usuário é: teste@gmail.com

A senha do usuário é: 123456
```

Esse é um tipo de acesso que deve ser evitado.

Name Mangling: é a modificação do nome do atributo que o Python faz.

Atenção: O acesso ao atributo privado deveria acontecer apenas dentro da classe.

 # 14 – Definindo uma classe e métodos para acessar seus atributos de instância público ou privado de fora da classe

```
class LoginIntranetDois:

    def __init__(self, email, senha):
        self.email = email
        self.__senha = senha

    def mostra_email(self):
        print(self.email)

    def mostra_senha(self):
        print(self.__senha)
```

 # 15 – Acessando atributos de instância público e privado de fora da classe

```
usuario_dois = LoginIntranetDois('Gnomica@gmail.com', '654321')

usuario_dois.mostra_email()
usuario_dois.mostra_senha()
```

```
Gnomica@gmail.com
O e-mail do objeto usuário_dois é: Gnomica@gmail.com

654321
A senha do objeto usuário_dois é: 654321
```

Nos atributos de instância cada instância (objeto) terá seu valor. É isso que permite que o Python crie várias instancias (objetos) a partir de uma classe (molde)

 # 16 – Definindo objetos a partir de uma classe (molde)

```
usuario_tres = LoginIntranetDois('Gnomica@gmail.com', '654321')

usuario_quatro = LoginIntranetDois('Gertrudez@gmail.com', '01010101')

usuario_cinco = LoginIntranetDois('Genoveva@gmail.com', '4503215')

usuario_sei = LoginIntranetDois('Gerimunda@gmail.com', '987654')
```

 # 17 – Acessando atributos de instância público e privado de fora da classe dos objetos criados a partir da classe modelo

```
usuario_tres.mostra_email()

usuario_quatro.mostra_email()

usuario_cinco.mostra_email()

usuario_seis.mostra_email()
```

```
Gnomica@gmail.com
O e-mail do objeto usuário_dois é: Gnomica@gmail.com

Gertrudez@gmail.com
O e-mail do objeto usuário_dois é: Gertrudez@gmail.com

Genoveva@gmail.com
O e-mail do objeto usuário_dois é: Genoveva@gmail.com

Gerimunda@gmail.com
O e-mail do objeto usuário_dois é: Gerimunda@gmail.com
```

VISIBILIDADE DOS ATRIBUTOS DE CLASSE

Atributos de classe: são valores dos atributos declarados diretamente na classe **FORA** do **método construtor**. Estes valores serão compartilhados dentro de todas as instâncias da classe.

A lógica aqui está: ao invés de cada instância da classe ter seus próprios valores, todas as instancias terão **o mesmo Valor** para o atributo.

Aplicabilidade: Num cenário onde todos os produtos de uma determinada loja de computadores deve ser recolhidos para o Governo Federal 0.08% de valor de venda de cada equipamento. Esta é uma das situações em que devemos utilizar **atributo de classe.**

 # 18 – Acessando um atributo de classe

```
print(Produto.imposto)
```

```
1.08
```

 # 19 - Atributo de classe - loja de venda de computadores

```
class Produto:

    imposto = 1.08  #  imposto sobre o valor de venda

    def __init__(self, descricao,cor,marca,tela,valor):
        self.descricao = descricao
        self.cor = cor
        self.marca = marca
        self.tela = tela
        self.valor = (valor * Produto.imposto)

    def mostra_na_tela(self):
        print(self.descricao)
        print(self.cor)
        print(self.marca)
        print(self.tela)
        print(self.valor)
```

Atenção: Não há necessidade de criar uma instância de uma classe para fazer o acesso a um atributo de classe.

 # 20 – Acessando atributos dos objetos criados a partir da classe modelo com o valor acrescido de 8%

```
produto_1  = Produto('Notebook Gamer', 'Preto','Dell', 'Monitor 15', 13542.25)

produto_2 = Produto('Magic Mouse 2 A1657', 'Branco','Apple', '2,16 X 5,71 cm', 675.00)

produto_1.mostra_na_tela()

print()

produto_2.mostra_na_tela()
```

```
Notebook Gamer
Preto
Dell
Monitor 15
14625.630000000001

Magic Mouse 2 A1657
Branco
Apple
2,16 X 5,71 cm
729.0
```

 # 21 - Atributo de classe - loja de venda de computadores – outro exemplo

```
class Produto:

    imposto = 1.12  #  imposto - atributo de classe
    contator = 0  # atributo de classe

    def __init__(self, descricao,cor,marca,tela,valor):
        self.id = Produto.contator + 1
        self.descricao = descricao
        self.cor = cor
        self.marca = marca
        self.tela = tela
        self.valor = (valor * Produto.imposto)
        Produto.contator = self.id # atributo de classe

    def mostra_na_tela(self):
        print(self.id)
        print(self.descricao)
        print(self.cor)
        print(self.marca)
        print(self.tela)
        print(self.valor)
```

 # 22 - Acessando atributos dos objetos criados a partir da classe modelo com o valor acrescido de 8% - com contador de produtos

```
produto_1  = Produto('Notebook Gamer', 'Preto','Dell', 'Monitor 15', 13542.25)

produto_2 = Produto('Magic Mouse 2 A1657', 'Branco','Apple', '2,16 X 5,71 cm', 675.00)

produto_1.mostra_na_tela()
print()
produto_2.mostra_na_tela()
```

```
1
Notebook Gamer
Preto
Dell
Monitor 15
15167.320000000002
```

```
2
Magic Mouse 2 A1657
Branco
Apple
2,16 X 5,71 cm
756.0000000000001
```

VISIBILIDADE DOS ATRIBUTOS DE INSTÂNCIA – ATRIBUTOS DINÂMICOS

Atributos Dinâmicos: são atributos criados em tempo de execução. **Atenção:** o atributo de dinâmico será **exclusivo** da instância que o criou. Exemplo: A partir da classe produto criada acima vamos definir um atributo dinâmico para nossa instância.

 # 23 - Criando um atributo dinâmico - **em tempo de execução – Não usual**

```
produto_1.peso = '5kg' # não existe o atributo peso na classe

produto_2.peso = '950gr'

produto_1.mostra_na_tela()
print(f'Peso: {produto_1.peso }')

print()
produto_2.mostra_na_tela()
print(f'Peso: {produto_2.peso }')
```

```
1
Notebook Gamer
Preto
Dell
Monitor 15
15167.320000000002
Peso: 5kg

2
Magic Mouse 2 A1657
Branco
Apple
2,16 X 5,71 cm
756.0000000000001
Peso: 950gr
```

24 - verificando as estruturas dos objetos produto_1 e produto_2

```
print('\n',dir(produto_1))

print('\n',dir(produto_2))
```

```
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'contator', 'cor', 'descricao', 'id', 'imposto', 'marca', 'mostra_na_tela', 'peso', 'tela', 'valor']

['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'contator', 'cor', 'descricao', 'id', 'imposto', 'marca', 'mostra_na_tela', 'peso', 'tela', 'valor']
```

25 - deletando atributos dinâmicos ou de classe

```
del produto_2.peso
del produto_2.tela

print('\n',dir(produto_2))
```

```
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'contator', 'cor', 'descricao', 'id', 'imposto', 'marca', 'mostra_na_tela', 'valor']
```

AMBIENTE VIRTUAL EM PYTHON SISTEMA POSIX LINUX E IOS
python -m venv env
source env/bin/activate
/Users/projts/env
pytest test_integration.py
pip install pytest-mock
pip install readchar
Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned
pip install windows-curses
https://www.geradorcpf.com – gerador de cpf para testes de programação
pip install pytest pytest-benchmark
pip install Flask
pip install locust
pylint --generate-rcfile > .pylintrc