

Importando uma classe como um módulo em um programa principal. Vamos criar o arquivo main.py

Vamos criar a classe para trabalhar os atributos, métodos e instancias.

```
class Carros():
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo
```

```
def imprime(self,marca,modelo):
print("Esse carro é %s e o Modelo %s" %(self.marca,self.modelo))
```





from classescarros import Carros

**Instanciar o objeto** 

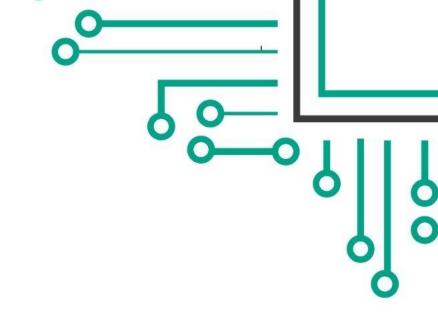
carro1 = Carros("Chevete","Chevrolet")

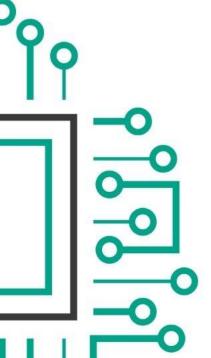
carro2 = Carros("Ranger","Ford")

Mostrar os valores dos objetos

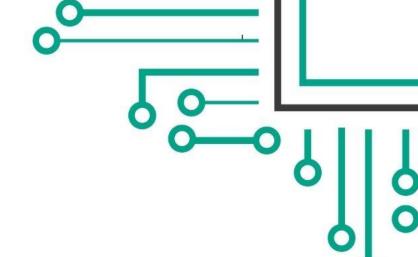
print(carro1.modelo)
print(carro1.marca)
print(carro2.modelo)
print(carro2.marca)

carro2.imprime("Ranger","Ford")









Atributos de Instancias X Atributos de Classes.

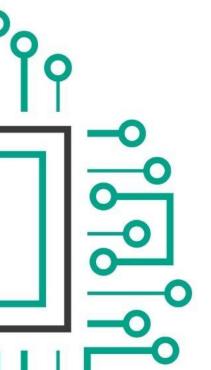
Até o momento trabalhamos com valores de atributos de instancia. Porém podemos definir valores de atributos de classe onde irá aplicar o mesmo valor para todas as instancias.

Atributos de Instancia:

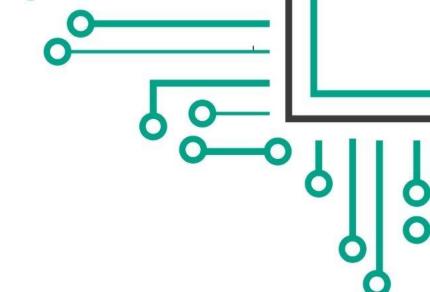
Carro3= Carros("KA", "Ford")

Para trabalharmos com atributo de classe é necessário atribuir um valor de atributo na classe fora do método construtor o \_\_init\_\_.

Vamos então refatorar(reescrever) a classe Carros.







```
class Carros():
    cor = "branco"
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo
```

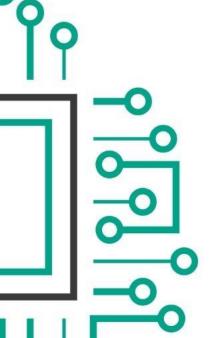
def imprime(self,marca,modelo): print("Esse carro é %s e o Modelo %s" %(self.marca,self.modelo))

Vamos validar os atributos de classe.

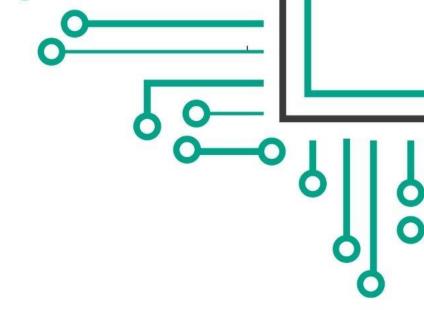
carro4 = Carros("Siena","Fiat")

print(carro4.cor) # Método incorreto de Exibir um atributo de classe

print(Carros.cor) # Método correto de exibir o atributo de classe







Exibir as instancias dos objetos com o método \_\_dict\_\_

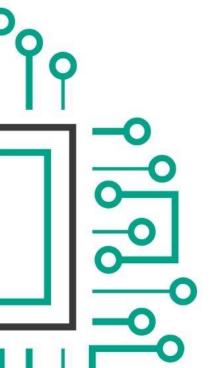
print(carro4.\_\_dict\_\_)

O mesmo não mostra os atributos de classe. Mas se utilizarmos o método na classe a mesma exibe os atributos de classe.

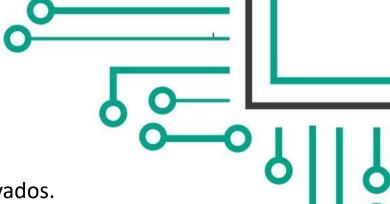
print(Carros.\_\_dict\_\_)

Atributos Públicos X Atributos Privados.

Todos os atributos declarados numa classe eles estão por padrão públicos. Mas podemos criar atributos privados não acessíveis pelas instancias. Declarando o atributo com duplo \_\_\_ underline.







Vamos criar uma nova classe com utilização de atributos públicos e privados.

```
class Cadastrouser():

def __init__(self,usuario,senha):

self.usuario = usuario

self.__senha = senha
```

Instanciando a classe

user1 = Cadastrouser('joao',123456)

Acessando os os valores da Instancia

print(user1.usuario)
print(user1.senha)





Para podermos exibir uma propriedade privada de uma classe vamos utilizar a função dir. Vamos entender a função dir().

O dir() retorna todas as propriedades e métodos do objeto especificado.

Vamos analisar a instancia com o a função dir.

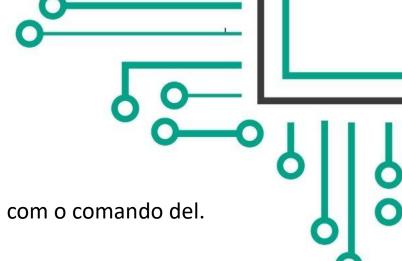
print(dir(user1))

Vamos pegar o atributo de senha \_Cadastrouser\_\_senha e vamos exibir na saída.

print(user1.\_Cadastrouser\_\_senha)

A senha do mesmo é exibida mesmo sendo privada.

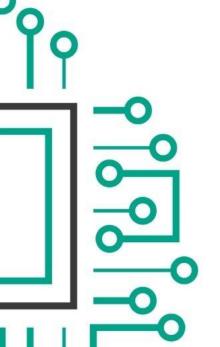




Podemos deletar um atributo de uma instancia em tempo de execução com o comando del.

del user1.usuario

print(user1.\_\_dict\_\_)



# JCAVI TREINAMENTOS EM TI



# EXERCÍCIO DE FIXAÇÃO



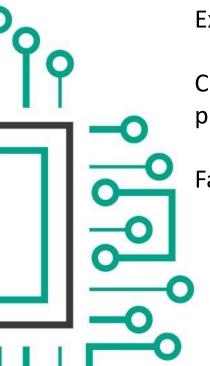
Crie uma classe e importe o mesma através de módulos no programa main.

Crie uma classe utilizando atributos de classe e atributos de instancia e teste as mesmas nos valores das instancias.

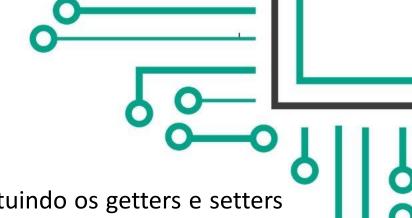
Exiba os valores das instancias com o método \_\_dict\_\_.

Crie uma classe com atributos públicos e privados. Instancie objetos com os atributos públicos e privados e tente exibi-los na saída.

Faça o Python exibir o atributo privado na saída na classe criada anteriormente.







Utilizando o método decorador @property e @.setter substituindo os getters e setters nas classes pythonicas.

Podemos deixar um atributo privado como publico com o @property usando a função de getters.

Para isso vamos criar uma classe.

```
class Vendaprodutos():
    def __init__(self,produto,quantidade,valor):
        self.__produto = produto
        self.__quantidade = quantidade
        self.__valor = valor
```

Vamos instanciar uma valor

produto1 = Vendaprodutos("Arroz",34,12.45)





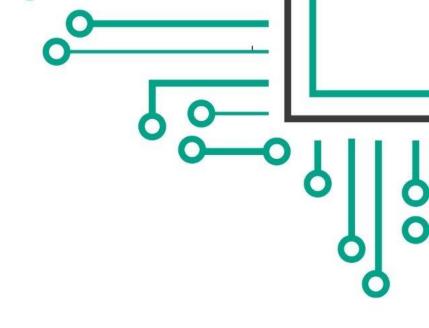
print(produto1.\_\_produto)

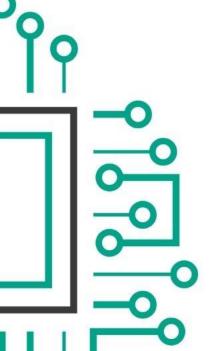
Vamos criar o método @property no atributo produto

@property
 def produto(self):
 return self.\_\_produto

Exibindo o Valor do Atributo

print(produto1.produto)









Agora vamos trabalhar com o atributo setter.

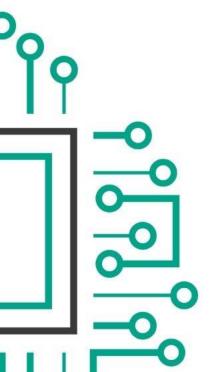
Para trabalhar com o @.setter e necessário trabalharmos com um atributo decorado. O setter modifica o valor padrão da instancia setando um novo valor.

Criando o @property do atributo.

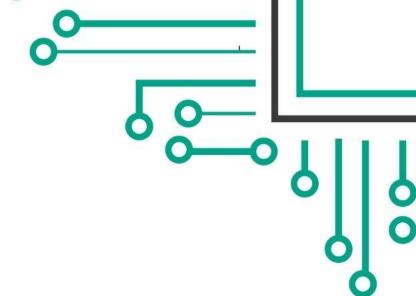
@property
 def quantidade(self):
 return self.\_\_quantidade

Agora vamos atribuir o setter para o atributo decorado.

@quantidade.setter
 def quantidade(self,nova\_quantidade):
 self.\_\_quantidade = nova\_quantidade







Alterar o valor de atributo da instancia.

produto1.quantidade = 45

Mostrando o valor alterado da instancia.

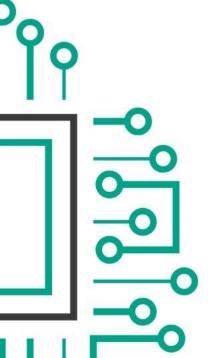
print(produto1.quantidade)

Podemos criar um método através da função decorada @property

@property
 def valor\_total\_compra(self):
 return self.\_\_quantidade \* self.\_\_valor

Retornar o valor total da Instancia.

print(produto1.valor\_total\_compra)





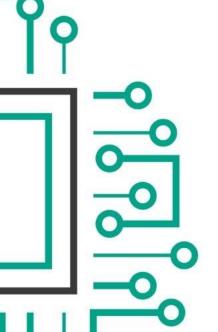
# EXERCÍCIO DE FIXAÇÃO

odo @properties

Crie uma classe com os atributos privados e exiba os valores de atributos com o método @properties na instancia do objeto.

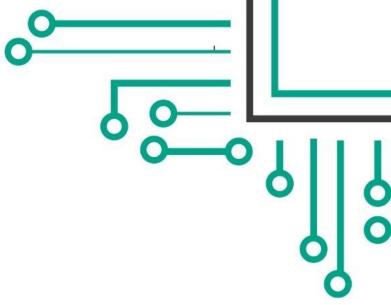
Na mesma classe aplique a função @property e setter em um dos atributos reescrevendo o valor do atributo na instancia.

Crie um método através de uma decoradora @property e execute a mesma na instancia do objeto.



# TREINAMENTOS EM TI





Trabalhando com Tipos de Métodos em Classes no Python.

- @classmethod
- @staticmethod

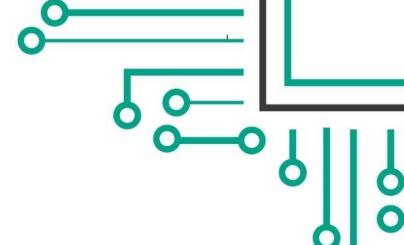
Private Method

Vamos Ver todos os tipos de métodos aplicando em uma classe.

Vamos Criar a Classe Livros.

```
class Livro():
    ano_atual = 2020
    def __init__(self,titulo,autor,ano):
        self.titulo = titulo
        self.autor = autor
        self.ano =ano
```





Metodo de Instancia Imprime

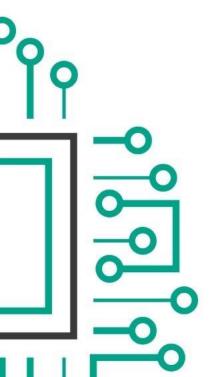
def imprime(self):
 print("Esse livro é %s e o Autor %s" %(self.titulo,self.autor))

Testando as variaveis em método de Instancia

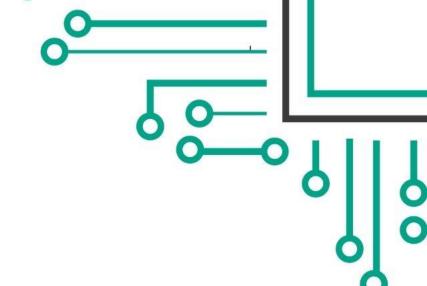
livro1 = Livro("O Pescador", "José Cunha", 1998) livro1.imprime()

Vamos criar um segundo atributo de instancia.

def anopublicacao(self):
 print(" Tempo de Publicação", self.ano\_atual - self.ano, "anos")







Chamando o atributo na instancia existente

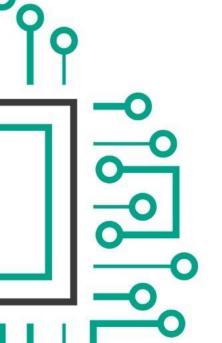
livro1.anopublicacao()

livro2 = Livro("Perdidos na Russia", "Antonio Vodka", 2015)

livro2.anopublicacao()

Vamos criar um método a nível classe onde os atributos e métodos são definidos na classe. Para criar os métodos de classe utilizamos o método decorado @classmethod.

```
@classmethod
  def calculoanopublicacao(vclasse,nome,ano):
     calculo = vclasse.ano_atual - ano
     return(nome,"Tempo Publicacao", str(calculo), "anos")
```







Testando por método de Classe. Valores específicos do método passados no método da classe

livro3 = Livro.calculoanopublicacao("Criando Ondas",1984)
print(livro3)

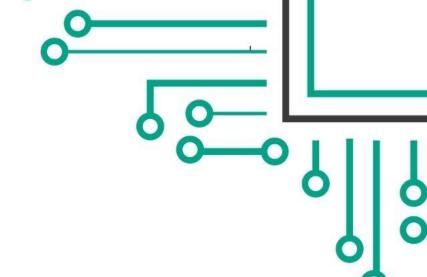
Criando métodos ocultos basta utilizar o \_\_ 2 underline para criar o método oculto. Vamos testar com o atributo autor e criar um método oculto para o mesmo.

self.\_\_autor = autor

Criar o método oculto

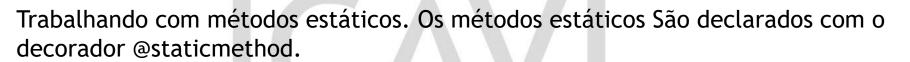
def \_\_autor(self):
 return(self.autor)





Criando a instancia e chamando o método oculto.

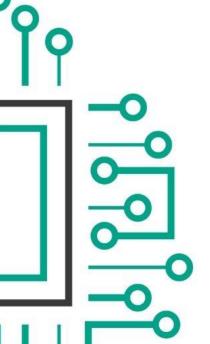
livro4 = Livro("João e Pé de Feijão","O Gigante",1830 )
print(livro4.autor)
print(livro4.\_Livro\_\_autor())



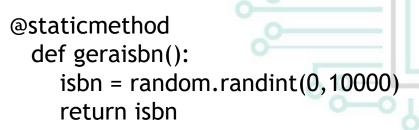
Vamos gerar o isbn dos nossos livros automaticamento utilizando um método estático onde a instancia não passa o valor para o mesmo.

Para esse exemplo vamos usar a biblioteca random.

Import random



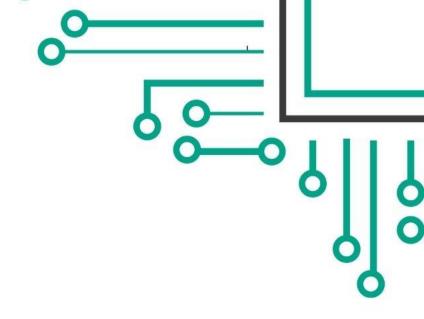




Testar método estático de uma instancia existente.

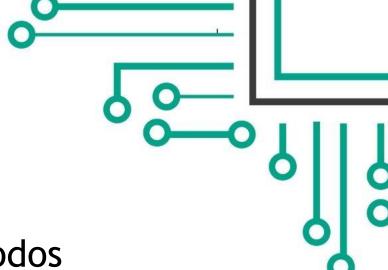
livro4.geraisbn()
print(livro4.geraisbn())
print(livro4.\_\_dict\_\_)



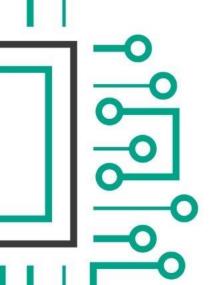




# EXERCÍCIO DE FIXAÇÃO



Crie uma classe aplicando os tipos de métodos mostrados até o momento na aula. Instancie os objetos e valides os valores e atributos definidos nos métodos.



TREINAMENTOS EM TI





Abstração e Encapsulamento:

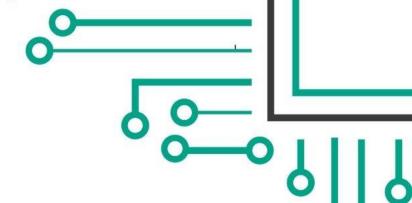
Com a abstração você expõe apenas os elementos públicos da classe(relevantes para a instancia). Escondendo os atributos e métodos privados.

Com encapsulamento você define atributos ou métodos acessíveis apenas dentro de uma classe apesar do Python deixar que o objeto consiga acessar esses atributos ou método. Isso em Python se chama de Naming Mangling.

Por convenção para tornar o atributo ou método privado em python podemos utilizar o 1 \_ underline ou 2 \_\_ underline.

- 1 \_ protect (ainda acessível)
- 2 \_\_\_ private (não acessível. Acessível apenas por naming mangling.)



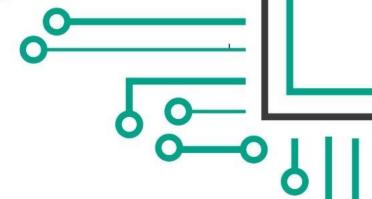


Vamos criar uma classe com atributos de abstração e encapsulamento.

import datetime

```
class Pessoa():
    anoatual = datetime.date.today().year
    def __init__(self,nome,sobrenome,cpf,anonascimento):
        self.nome = nome
        self.sobrenome = sobrenome
        self.__cpf = cpf
        self._anonascimento = anonascimento
        @classmethod
    def idade(vanoatual,_anonascimento):
        calculaidade = vanoatual.anoatual - _anonascimento
        print ("Idade é :",calculaidade)
```





Criamos um atributo privado CPF e um atributo protected a data de nascimento. Vamos criar uma classe agora para testarmos os encapsulamentos.

import datetime

```
class Pessoa():
    anoatual = datetime.date.today().year
    def __init__(self,nome,sobrenome,cpf,anonascimento):
        self.nome = nome
        self.sobrenome = sobrenome
        self.__cpf = cpf
        self._anonascimento = anonascimento
    @classmethod
    def idade(vanoatual,_anonascimento):
        calculaidade = vanoatual.anoatual - _anonascimento
        print ("Idade é :",calculaidade)
```





Vamos agora instanciar os objetos e testar os atributos de encapsulamento e de abstração.

p1 = Pessoa("joao", 'cavichiolli', 4454444, 1985)

Exibindo atributos protected

print(p1.\_anonascimento)

Exibindo Método de Classe

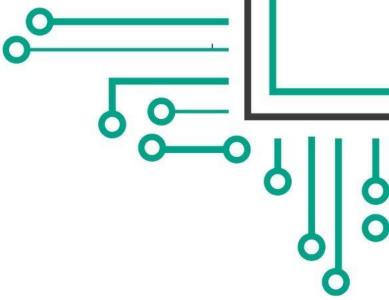
p1.idade(1985)

**Exibindo atributos Privados** 

p1.cpf







Exibindo atributo privado por naming mangling.

print(p1.\_Pessoa\_\_cpf)

Vamos deixar o método idade privado e vamos rodar o programa com o objeto criado e vamos validar o que vai ocorrer.

p1.idade(1985)

Trazendo o atributo privado

(p1.\_Pessoa\_\_idade(1985))

Lembrando que não é correto trazer um atributo ou método de classe em uma instancia de objeto.



# EXERCICIO DE FIXAÇÃO



Crie uma classe com atributos ou métodos públicos, privados ou protegidos.

Testes os valores tentando acessar os mesmos através de uma instancia.

Com esses conceitos iremos entender o que é abstração e encapsulamento de objetos na programação orientada a objetos.

Utilizando Naming Mangling exiba os valores das instancia com atributos ou métodos privados.

