CLASSES E OBJETOS

Herança, Heranças Multiplas e Polimorfismo.

Trabalhando com Herança:

Vamos criar uma classe principal também conhecida como SuperClasse. Depois vamos conhecer o método super() para trazer os atributos da classe principal.

```
class Revendedora():
    def __init__(self):
        print("Veiculo Cadastrado")

    def tipoveiculo(self,tipoveiculo):
        self.tipoveiculo = tipoveiculo
        print("Tipo Adicionado")
```

CLASSES E OBJETOS

```
def quantidade(self,quantidade):
    self.quantidade = quantidade
    print("Quantidade Adicionada")

  def valor(self,valor):
    self.valor = valor
    print("Valor Adicionado")
```

Vamos criar uma classe filha chamada RevendedoraCarros onde irá receber os atributos e métodos da classe Revendedora.

Vamos declarar a classe RevendedoraCarros

```
from classerevendedora import Revendedora
```

CLASSES E OBJETOS

```
class RevendedoraCarros(Revendedora):
  def __init__(self):
      super().__init__()
      print("Carro Cadastrado")

def marca(self,marca):
      self.marca = marca
      print("Marca Adicionada")

  def modelo(self,modelo):
      self.modelo = modelo
      print("Modelo Adicionado")
```

Criando o arquivo main para importar as classes

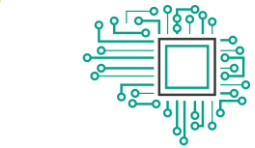
CLASSES E OBJETOS

```
from classerevendoracarros import RevendedoraCarros

carros1 = RevendedoraCarros()

carros1.modelo("Gol")
carros1.quantidade(3)
carros1.valor(34.342)
carros1.marca("Volks")
carros1.tipoveiculo("Carro")
```

Efetuamos os testes com os métodos da classe RevendaCarros e também da classe Revenda.
Vamos criar uma segunda classe herdando atributos e métodos da superclasse.
Vamos criar a classe RevendaMotos



CLASSES E OBJETOS

```
from classerevendedora import Revendedora

class RevendedoraMotos(Revendedora):
    def __init__(self):
        super().__init__()
        print("Moto Cadastrada")

    def marca(self,marca):
        self.marca = marca
        print("Marca Adicionada")

    def modelo(self,modelo):
        self.modelo = modelo
        print("Modelo Adicionado")

    def potencia(self,potencia):
        self.potencia = str(potencia)
        print("Potencia Adicionada")
```

CLASSES E OBJETOS

Importando a classe no main e instanciando os objetos.

```
from revendedoramotos import RevendedoraMotos

moto1 = RevendedoraMotos()

moto1.tipoveiculo("Motocicleta")
moto1.valor(12.422)
moto1.marca("Honda")
moto1.modelo("CB100")
moto1.potencia("1000CC")
```

CLASSES E OBJETOS

Método super() o método super é utilizado para vincular atributos ou métodos da super classe.

Vamos utilizar a classe RevendedoraMotos e trazer um método da super Classe revendedora.

```
class RevendedoraMotos(Revendedora):
    def __init__(self):
        super().__init__()
        super().valor
        print("Moto Cadastrada")
```

CLASSES E OBJETOS

Sobreposição de atributos ou métodos. (Polimorfismo)

Nas classes filhas que herdam atributos ou objetos das classes pai ou superclasses podem haver métodos ou atributos conflitantes. Os conflitantes irão respeitar o atributo da classe e não da herança. Criando o conceito de polimorfismo. (Ou sobrescrita de métodos). Tomando uma ação diferente do método herdado.

```
def tipoveiculo (self,tipoveiculo):
        self.tipoveiculo = tipoveiculo
        print("Tipo de Método Acionado pela Classe Motos")
```

JCAVI
TREINAMENTOS EM TI

EXERCÍCIO DE FIXAÇÃO

Crie uma classe e defina ela como uma super classe.

Crie uma classe filha e configure a herança de classe com a superclasse.

Instancie atributos da classe filha e atributos da super classe.

Crie uma superposição de atributos ou métodos de uma classe filha.

JCAVI
TREINAMENTOS EM TI

CLASSES E OBJETOS

HERANÇAS MÚLTIPLAS.

Herança Múltiplas é onde uma classe pode herdar atributos métodos de várias outras classes atribuídas na herança.

Vamos criar a classe de Animal para tartaruga a mesma pode ser marinha ou terrestre.

```
class Animal():
    def __init__(self,especie):
        self.especie = especie

    def tipo(self):
        return(f" È da especie {self.especie}" )


    def som (self):
        print(f" A {self.especie} não faz som ")
```

CLASSES E OBJETOS

Vamos criar as classes Marinhas e Terrestres que herdam da Classe Animal.

```
class Aquatico(Animal):
    def __init__(self,especie):
        super().__init__(especie)

    def tipo(self):
            print(f" È da especie {self.especie} que nada no mar" )


class Terrestre(Animal):
    def __init__(self,especie):
        super().__init__(especie)

    def tipo(self):
            print(f" È da especie {self.especie} que Anda na Terra" )
```

CLASSES E OBJETOS

Por fim a classe Tartaruga que vai herdar das classes Terrestre e Marinha.

```
class Tartaruga(Aquatico,Terrestre):
    def __init__(self,especie):
        super().__init__(especie)
```

Vamos instanciar os objetos e testar os atributos e métodos herdados.

```
t1 = Tartaruga("Marinha")

print(t1.especie)

t1.tipo()

t1.som()
```

CLASSES E OBJETOS

```
t2 = Tartaruga("Jabuti")
print(t2.especie)
t2.tipo()
```

Vemos que a saída foi: È da especie Jabuti que nada no mar. Porém o Jabuti é terrestre não do Mar.
Isso ocorre porque o Python obedece o MRO (Method Resolution Order). Ele está respeitando a ordem de herança. Devemos inverter o modo da herança para ele respeitar os métodos

JCAVI
TREINAMENTOS EM TI

EXERCICIO DE FIXAÇÃO

Crie 4 Classes e faça uma herança simples nas primeiras e uma múltipla na ultima.

Crie métodos nas instancias e tente utilizar métodos e atributos herdados de outras classes.

CLASSES E OBJETOS

Associação, Agregação, Composição.

Trabalhando com Associação: Associação descreve um vínculo que ocorre entre classes.

Vamos utilizar a classe Livro para exemplo de associação.

```
class Livro():
    def __init__(self,nome):
        self.nome = nome
        self.__editora = None
        self.__autor = None

  @property
    def editora(self):
        return self.__editora
```

CLASSES E OBJETOS

```
@editora.setter
    def editora (self,editora):
        self.__editora = editora

@property
    def autor(self):
        return self.__autor

@autor.setter
  def autor (self,autor):
      self.__autor = autor
```

Criando a Classe Editora.

```
class Editora():
    def insereeditora(self,insereeditora):
        print(f"Essa é a Editora: {insereeditora}")
```

CLASSES E OBJETOS

Criando a Classe Autor

```
class Autor():
      def insereautor(self,insereautor):
        print(f"Esse é o Autor: {insereautor}")
```

Instanciando os objetos.

```
l1 = Livro("Os Calculos")
aut1  = Autor()
ed1= Editora()
print(l1.nome)

l1.editora = ed1
l1.autor = aut1

l1.editora.insereeditora("Editora das Contas")
l1.autor.insereautor("O Calculador")
```

CLASSES E OBJETOS

Trabalhando com Agregação: É um tipo especial de associação onde tenta-se demonstrar que as informações de um objeto (chamado objeto-todo) precisam ser complementados pelas informações contidas em um ou mais objetos de outra classe (chamados objetos-parte);

Vamos utilizar a classe CompraFruta e Frutas para praticar a agregação de classes. Utilizando uma lista para popular os objetos

Criando a classe CompraFruta.

```
class CompraFrutas():
    def __init__(self):
        self.frutas = []

    def inserir_frutas(self, frutas):
        self.frutas.append(frutas)
```

CLASSES E OBJETOS

```
def lista_frutas(self):
    for frutas in self.frutas:
        print(frutas.nome)


def soma_frutas(self):
    total_valor = 0
    for frutas in self.frutas:
        total_valor += frutas.valor
    return total_valor
```

Criando a Classe Frutas

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

CLASSES E OBJETOS

Instanciando os objetos

```
cestafrutas = CompraFrutas()

fruta1 = Frutas('Maça', 10)
fruta2 = Frutas('Morango', 15)

cestafrutas.inserir_frutas(fruta1)
cestafrutas.inserir_frutas(fruta2)

cestafrutas.lista_frutas()

print(cestafrutas.soma_frutas())
```

JCAVI
TREINAMENTOS EM TI

CLASSES E OBJETOS

Trabalhando com Composição: Pode-se dizer que composição é uma variação da agregação. Uma composição tenta representar também uma relação todo - parte. No entanto, na composição o objeto-pai (todo) é responsável por criar e destruir suas partes.

Vamos trabalhar com uma classe que registra o funcionário em um setor. A mesma vai armazenar a informação dos funcionários em uma lista.

Criando a classe RegistroSetor.

```
class RegistroSetor():
    def __init__(self, setor, funcao):
        self.setor = setor
        self.funcao = funcao
        self.funcionarios = []
```

CLASSES E OBJETOS

```
def insere_funcionarios(self, nome, sobrenome):
        self.funcionarios.append(Funcionarios(nome, sobrenome))

    def lista_funcionario(self):
        for funcionarios in self.funcionarios:
            print(funcionarios.nome, funcionarios.sobrenome,self.funcao)
```

Criando a Classe Funcionário onde receberá o cadastro.

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

CLASSES E OBJETOS

Instanciando os Objetos.

```
func1 = RegistroSetor('Faturamento', "Analista")
func1.insere_funcionarios('João' ,"Cavichiolli")
print(func1.setor)
func1.lista_funcionario()


func2 = RegistroSetor('Financeiro', "Analista")
func2.insere_funcionarios('Juca' ,"Bala")
print(func2.setor)
func2.lista_funcionario()
```

Para testarmos o funcionamento da exclusão de um registro de um objeto instanciado do pai o python trata de apagar os registros da classe filha também.

CLASSES E OBJETOS

Para testar esse funcionamento vamos declarar o método del (__del__) nas classes instanciadas.

```
def __del__(self):
        print("Apagou da Classe Setor")


  def __del__(self):
          print("Apagou da Classe Funcionarios")
```

JCAVI
TREINAMENTOS EM TI

EXERCÍCIO DE FIXAÇÃO

Criar Classes utilizando o método de Associação e instanciar objetos

Criar Classes utilizando o método de Agregação e instanciar objetos

Criar classes utilizando métodos de composição e instanciar objetos.