Avaliação 04 - POO
Aluno: Juliana de Freitas Pereira

1ª Questão (10 Escores). Associe a cada item da 2ª coluna um valor que corresponde a um item da 1ª coluna.

a) Permite que um objeto seja usado no lugar de outro.
(c) Encapsulamento

b) Define a representação de um objeto.
(h) Mensagem

c) Separação de interface e implementação que permite que usuários de objetos possam utilizá-los sem conhecer detalhes de seu código.
(i) Herança

d) Possui tamanho fixo.
(a) Polimorfismo

e) Instância de uma classe.
(f) Dependência

f) Forma de relacionamento entre classes onde objetos são instanciados no código.
(j) Lista

g) Forma de relacionamento entre classes implementado por meio de coleções.

(b) Classe

h) Forma de chamar um comportamento de um objeto.

(e) Objeto

i) Reuso de código na formação de hierarquias de classes.

(g) Composição

j) Permite inserções e remoções.

(d) Array

2ª Questão (10 Escores). Aplique V para as afirmações verdadeiras e F para as afirmações falsas.

a) Métodos construtores devem sempre ser explícitos. (F)

b) A classe Professor tem um relacionamento de agregação com a classe Disciplina. (V)

c) Quando uma classe possui como atributo uma referência para um objeto temos uma dependência. (V)

d) Membros de classes static existem mesmo quando nenhum objeto dessa classe exista. (V)

e) Um relacionamento ´tem um´ é implementado via herança. (F)

f) Uma classe Funcionário tem um relacionamento ´é um´ com a classe Dependente. (F)

g) Uma classe abstract pode ser instanciada. (F)

h) Relacionamentos TODO-PARTE são tipos de associações. (V)
i) Você implementa uma interface ao subscrever apropriada e concretamente todos os métodos definidos
pela interface. (V)
j) Um método static não é capaz de acessar uma variável de instância. (F)

3ª Questão (40 Escores). Escreva exemplos de código Python onde seja possível identificar os seguintes conceitos de POO.
a) Herança;

```
#HERANÇA

class Endereco():
    def __init__(self, rua, numero, bairro):
        self.rua = rua
        self.numero = numero
        self.bairro = bairro
    def mostrar(self):
        print("Rua:", self.rua)
        print("Número da casa:", self.numero)
        print("Bairro:", self.bairro)

class Pessoa(Endereco):
    def __init__(self, rua, numero, bairro, nome):
        super(Pessoa, self).__init__(rua, numero, bairro)
        self.nome = nome

    def mostrar(self):
        super(Pessoa, self).mostrar()
        print("Nome:", self.nome)
```

```
pessoa2 = Pessoa("Rua Planalto Pici", 100, "Planalto do Pici",
"Jules")
pessoa2.mostrar()
```

b) Encapsulamento;

```
 #ENCAPSULAMENTO

class ContaBancaria():
    def __init__(self, saldo):
        self.__saldo = saldo  #Privado

    def sacar(self, valor):
        self.__saldo -= valor

    def depositar(self, valor):
        self.__saldo += valor

    def getSaldo(self):
        return self.__saldo

    def setSaldo(self, novo):
        self.__saldo = novo

conta1 = ContaBancaria(1200)

conta1.depositar(400)
print("Valor após o depósito é:",conta1.getSaldo())

conta1.sacar(300)
print("Valor após o saque é:",conta1.getSaldo())
```

#Isso serve para os casos em que o depósito é feito primeiro e o saque em seguida.

c) Polimorfismo;

```
 #POLIFORMISMO

class Super:
 def hello(self):
  print("Bom dia!")

class Sub (Super):
 def hello(self):
  print("Boa tarde!")

teste = Sub()
teste.hello()
```

d) Variáveis de Instância;

```
 #VARIÁVEIS DE INSTÂNCIA

class ContaBancaria():
    saldoBanco = 50000
    def __init__(self, saldo):
        self.__saldo = saldo
  ContaBancaria.saldoBanco += saldo

    def sacar(self, valor):
        self.__saldo -= valor
        ContaBancaria.saldoBanco -= valor
```

```
    def depositar(self, valor):
        self.__saldo += valor
        ContaBancaria.saldoBanco -= valor

    def getSaldo(self):
        return self.__saldo

    def setSaldo(self, novo):
        self.__saldo = novo



conta1 = ContaBancaria(5000)

conta1.depositar(300)print(conta1.getSaldo())

conta1.sacar(630)
print(conta1.getSaldo())

print(ContaBancaria.saldoBanco)
```

e) Métodos construtores;

```
 #MÉTODOS CONSTRUTORES

class Veiculo():

    def __init__(self, cor, marca, portas):
        self.cor = cor
        self.marca = marca
        self.portas = portas
```

```
celta = ("Branco", "Chevrolet", 4)
```

f) Dependência;

g) Associação;

```
  #ASSOCIAÇÃO

class Musica():
    def __init__(self, nome, artista):
        self.nome = nome
        self.artista = artista

class Artista():
    def __init__(self, id, nome):
        self.id = id
        self.nome = nome
        self.cancoes = list()

    def adicionarMusica(self, musica):
        self.cancoes.append(musica)

    def mostrarMusicas(self):
        for a in self.cancoes:
            print("Nome:", a[1])
            print()

artistaAna = Artista(2, "Ana Carolina")

musicaA = [1, "Quem de nós dois"]
musicaB = [2, "Confesso"]
```

```
artistaAna.adicionarMusica(musicaA)
artistaAna.adicionarMusica(musicaB)

artistaAna.mostrarMusicas()
```

h) Relacionamento TODO-PARTE;

```
  #RELACIONAMENTO TODO-PARTE

class Pedido():
    def __init__(self, id, *itens):
        self.id = id
        self.itens = list(map(lambda x : x.__dict__, itens))

    def pago(self):
        print("Os produtos são:")
        for a in self.itens:
            print(a["nome"])
        print("Total:", sum(list(map(lambda x : x["preco"],
self.itens))))


class itemPedido():
    def __init__(self, nome, preco):
        self.nome = nome
        self.preco = preco

pedido1 = Pedido(1, itemPedido("Maçã", 4),
itemPedido("Frango",25), itemPedido("Macarrão", 5))
pedido1.pagamento()
```

4ª Questão (20 Escores)
Escreva em Python uma classe Ponto que possui os atributos inteiros x e y. Escreva uma classe Reta que possui dois pontos a
e b. Escreva os métodos construtores para a classe Ponto e para a Classe Reta. Escreva os métodos get e set para acessar e
alterar os atributos da classe Ponto e da classe Reta. Escreva um método distancia que retorna um valor real da distância entre os dois pontos da reta.

```
import math

class Ponto(object):
    def init(self,x,y):
        self.x=x
        self.y=y

    def getponto(self):
        print(f'\nx: {self.x}\t y: {self.y}')

    def set_x(self,x):
        self.x=x

    def set_y(self,y):
      self.y=y

class Reta(object):
    def init(self,ax,ay,bx,by):
        self.ax = ax
        self.ay = ay
        self.bx = bx
        self.by = by
```

```
    def get_distancia(self):
        d=math.sqrt((self.bx - self.ax) * (self.bx - self.ax)+(self.by -
self.ay) * (self.by - self.ay))
        print("A distância é igual a {}".format(d))

    def set_a(self,a):
        self.a=a

    def set_b(self,b):
        self.b=b

if name == 'main':
    x = int(input('digite o valor de X: '))
    y = int(input('digite o valor de Y: '))
    print('Forme o ponto a com ax e ay')
    ax = int(input('digite o valor de ax: '))
    ay = int(input('digite o valor de ay: '))
    print('Forme o ponto b com bx e by')
    bx = int(input('digite o valor de bx: '))
    by = int(input('digite o valor de by: '))
    oz = Ponto(x, y)
    oz.getponto()
    iz = Reta(ax,ay,bx,by)
    iz.get_distancia()
```