João Victor da Silva Ferreira | Informática – P7

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.

b) Define a representação de um objeto.

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

d) Possui tamanho fixo.

e) Instância de uma classe.

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

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

h) Forma de chamar um comportamento de um objeto

i) Reuso de código na formação de hierarquias de classes

j) Permite inserções e remoções

Alternativas:

(c) Encapsulamento

(h) Mensagem

(i) Herança

(a) Polimorfismo

(f) Dependência

(j) Lista

(b) Classe

(e) Objeto

(g) Composição

(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;

```
class Veiculo:
 def __init__(self, tipo, marca, km):
  self.tipo = tipo
  self.marca = marca
  self.km = km

class Carro(Veiculo):
 def __init__(self, tipo, marca, modelo, cor, ano, km, portas):
  Veiculo.__init__(self, tipo, marca, km) #Herdando da Classe Veiculo
os seus atributos

  self.portas = portas
  self.modelo = modelo
  self.cor = cor
  self.ano = ano

 def printCarro(self):
  print("Tipo: " + str(self.tipo), "\nMarca: " + str(self.marca),
"\nModelo: " + str(self.modelo),"\nCor: " + str(self.cor), "\nAno: " +
str(self.ano), "\nCom", self.km, "km rodados e", self.portas,
"portas")

carro = Carro("Carro", "Audi", "Audi Skysphere","Cinza", 2021,
"10.000", 2)
carro.printCarro()
```

b) Encapsulamento;

```
class Funcionario:
    def __init__(self, nome, cargo, valor_hora_trabalhada):
        self.nome = nome
        self.cargo = cargo
        self.valor_hora_trabalhada = valor_hora_trabalhada
        self.__salario = 0
        self.__horas_trabalhadas = 0

    @property
    def salario(self): #Propriedade salario criada
        return self.__salario

    @salario.setter
    def salario(self, novo_salario):
    #Uma vez restringido o acesso à propriedade salário, é instruído
```

```
que altere o valor da variável salario usando a função
calcula_salario()
        raise ValueError("Impossivel alterar o salário diretamente.
Use a função calcula_salario()")

    def registra_hora_trabalhada(self):
        self.__horas_trabalhadas += 1

    def calcula_salario(self):
        self.__salario = self.__horas_trabalhadas *
self.valor_hora_trabalhada

func = Funcionario('João', 'Programador', 50)
func.salario = 100000
```

c) Polimorfismo;

```
class Super:

 def Classe(self):
  print("Classe A")

class Sub (Super):

 def Classe(self):
  print("Classe B")

class Subsub (Sub):

 def Classe(self):
  print("Classe C")

chamada = Subsub()
chamada.Classe()
#Embora esteja sendo chamando sempre o mesmo método (Classe), que está
presente em todas as classes e nas herdadas, o método se comporta de
diferente
```

d) Variáveis de Instância;

```
class Veiculo:
    tipo = "Carro" # Variável de classe compartilhada por todas as
instâncias

    def __init__(self, marca):
        self.marca = marca    # Variável de instância única para cada
instância

c1 = Veiculo("Audi") # Único para c1
c2 = Veiculo("BMW")  # Único para c2
c3 = Veiculo("Ford") # Único para c2

print("Tipo do Veículo: " + c1.tipo + " | Marca do Veículo: " +
c1.marca)
print("Tipo do Veículo: " + c2.tipo + " | Marca do Veículo: " +
c2.marca)
print("Tipo do Veículo: " + c3.tipo + " | Marca do Veículo: " +
c3.marca)
```

e) Métodos construtores

```
class Pessoa:

    def __init__(self, sexo, nome, idade, altura, peso): # Método
construtor
        self.sexo = sexo
        self.nome = nome
        self.idade = idade
        self.altura = altura
        self.peso = peso

    def printInfo(self):
        print("Sexo: " + str(self.sexo), "\nNome: " + str(self.nome),
"\nIdade: " + str(self.idade),"\nAltura: " + str(self.altura),
"\nPeso: " + str(self.peso))

ps  = Pessoa("Masculino", "João Victor", "17 anos", "1,69m", "70kg")
ps.printInfo()
```

f) Dependência

```
# Há uma relação de dependência com as classes abaixo
class Carro:

    def __init__(self,marca):
        self.marca = marca

    def ligandoCarro(self):
        print("O carro da marca " + str(self.marca) + " foi ligado")

    def desligandoCarro(self):
        print("O carro da marca " + str(self.marca) + " foi
desligado")

class Pessoa:

    def ligarCarro(self, acao):
        acao.ligandoCarro()

    def desligarCarro(self, acao):
        acao.desligandoCarro()

veiculo = Carro("Audi")
pessoa = Pessoa()

pessoa.ligarCarro(veiculo)
#O método ligarCarro da classe Pessoa depende do método ligandoCarro
da classe Carro para funcionar

pessoa.desligarCarro(veiculo)
#Assim como no exemplo anterior, o método desligarCarro da classe
Pessoa depende do método desligandoCarro da classe Carro para
funcionar
```

g) Associação

```
# As duas classes possuem uma relação de associação

class Contato:
    def __init__(self, nome):
        self.nome = nome

    def ligar(self):
        print("Ligando para " + str(self.nome))

    def mensagem(self):
        print("Enviando mensagem para " + str(self.nome))

class Pessoa:

    def ligar_contato(self, contato):
        contato.ligar()

    def mensagem_contato(self, contato):
        contato.mensagem()

pessoa = Pessoa()
contato_raquel = Contato("Raquel")
mensagem_carlos = Contato("Carlos")

pessoa.ligar_contato(contato_raquel)
pessoa.mensagem_contato(mensagem_carlos)

# O objeto pessoa da classe Pessoa está se associando com a classe
Contato
```

h) Relacionamento TODO-PARTE

```
# Uma Composição é uma relação entre dois objetos para criar um só,
representando um relacionamento do tipo "todo/parte"
class Contato:

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

 def retornaNome(self):
  return self.__nome

class Agenda:

 cnt = []

 def __init__(self):

  while True:
   print("\nDigite 1 para adicionar um contato")
   print("Digite 2 para exibir a lista de contatos")
   op = int(input("\nEscolha uma opção: "))

   if op == 1:
    self.adicionar()

   elif op == 2:
    self.exibir()
```

```
   else:
    print("Opção invalida")

 def adicionar(self):
  nome = input("Escreva o nome do contato: ")
  self.cnt.append(Contato(nome))
  # Á medida que for contratado um funcionário, ele será adicionado na
lista de funcionários, a func. Caracterizando desse modo, uma
composição

 def exibir(self):
     print("\nLista de contatos")

     for contato in self.cnt:
         print(contato.retornaNome())

Agenda()
```

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():

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def getX(self):
        return self.x

    def setX(self, valor):
        self.x = valor

    def getY(self):
        return self.y

    def setY(self, valor):
        self.y = valor

class Reta():

    def __init__(self, a, b):
        self.a = a
        self.b = b

    def getPontoA(self):
        return self.a

    def setPontoA(self, valor):
        self.a = valor

    def getPontoB(self):
```

```
        return self.b

    def setPontoB(self, valor):
        self.b = valor

    def calcularDistancia(self):

        dist = math.sqrt(math.pow((self.getPontoB().getX() -
self.getPontoA().getX()), 2) + math.pow((self.getPontoB().getY() -
self.getPontoA().getY()), 2))

        print("\nPonto A: (" + str(self.getPontoA().getX()) + "," +
str(self.getPontoA().getY()) +")")
        print("Ponto B: (" + str(self.getPontoB().getX()) + "," +
str(self.getPontoB().getY()) +")")

        print("\nValor da distância: " + str(dist))

PA = Ponto(int(input("Insira a coordenada X do ponto A: ")),
int(input("Insira a coordenada Y do ponto A: ")))
PB = Ponto(int(input("\nInsira a coordenada X do ponto B: ")),
int(input("Insira a coordenada Y do ponto B: ")))

reta = Reta(PA, PB)
reta.calcularDistancia()
```