

**DEPARTAMENTO DE TELEMÁTICA**

**DISCIPLINA: PROGRAMAÇÃO ORIENTADA A OBJETO**

**LISTA EXERCICIO**

**ALUNO: \_Samuel Bruno Silva Secunde\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Data: \_30\_/ \_09\_/\_2021\_**

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. | ( h ) | 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. | ( F ) |
| 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.

1. Herança;

class Pessoa:

def \_\_init\_\_ (self, nome, idade, sexo):

self.nome = nome

self.idade = idade

self.sexo = sexo

class Crianca(Pessoa):

pass

c1 = Crianca("Pedro", 7, 'M')

print(c1.nome)

print(c1.idade)

print(c1.sexo)

1. Encapsulamento;

class Salario():

def \_\_init\_\_(self, horas, valor\_hora, saldo\_inicial=0):

self.horas = horas

self.valor\_hora = valor\_hora

self.\_\_saldo\_inicial = saldo\_inicial

def imprimir(self):

valor\_ganho = self.valor\_hora \* self.horas

print("Valor Ganho: " + str(valor\_ganho))

print("Saldo: " + str(self.\_\_saldo\_inicial + valor\_ganho))

funcionario = Salario(10, 100, 0)

funcionario.saldo\_inicial = 999999999999999

funcionario.imprimir()

1. Polimorfismo;

class Pessoa:

def \_\_init\_\_(self,Nome):

self.Nome = Nome

def falar(self):

print(f'{self.Nome} Estah Falando!')

class Aluno(Pessoa):

def falar():

print("Mehtodo Falar Foi Alterado!")

pessoa1 = Pessoa("Jose")

pessoa1.falar()

pessoa2 = Aluno

pessoa2.falar()

1. Variáveis de Instância;

class cadastro\_viagem():

def \_\_init\_\_(self, nome, idade, cidade\_onde\_mora, cidade\_viagem):

self.nome = nome

self.idade = idade

self.cidade\_onde\_mora = cidade\_onde\_mora

self.cidade\_viagem = cidade\_viagem

def imprimir\_cadastro(self):

print(self.nome + "\n" + str(self.idade) + "\n" + self.cidade\_onde\_mora + "\n" + self.cidade\_viagem)

primeiro = cadastro\_viagem("Joao", 27, "Fortaleza", "Paris")

primeiro.imprimir\_cadastro()

1. Métodos construtores

class Dados():

def \_\_init\_\_(self, cnpj, frase):

self.cnpj = cnpj

self.frase = frase

def imprimir(self):

print("CNPJ: " + str(self.cnpj)+ "\n" + "Frase: " + self.frase)

pessoa1 = Dados(12345678910, "O sol eh amarelo!")

pessoa1.imprimir()

1. Dependência

class Pessoa:

def \_\_init\_\_(self,nome):

self.nome = nome

def Parar(self):

print("PARADO!")

class Carro:

def \_\_init\_\_(self,marca):

self.marca = marca

def PararCarro(self,veiculo):

veiculo.Parar()

pessoa = Pessoa("Fulano")

carro = Carro("BMW")

carro.PararCarro(pessoa)

1. Associação

class Cliente:

def \_\_init\_\_(self, nome, idade, cnpj):

self.\_\_nome = nome

self.\_\_idade = idade

self.\_\_cnpj = cnpj

@property

def get\_nome(self):

return self.\_\_nome

@property

def get\_idade(self):

return self.\_\_idade

@property

def get\_cnpj(self):

return self.\_\_cnpj

cliente1 = Cliente("Samuel", 29, 12345678910)

print(cliente1.get\_nome)

print(cliente1.get\_idade)

print(cliente1.get\_cnpj)

1. Relacionamento TODO-PARTE

class Livro:

def \_\_init\_\_(self,nome):

self.nome = nome

class Biblioteca:

def \_\_init\_\_(self):

self.livros = []

def Inserir\_Livro(self,nome\_livro):

self.livros.append(nome\_livro)

def Mostrar\_Catalogo(self):

for nome\_livro in self.livros:

print(f"Livro: {nome\_livro}")

Livro1 = Livro("Harry Potter")

Livro2 = Livro("Como Treinar o Seu Dragao")

DaEsquina = Biblioteca()

DaEsquina.Inserir\_Livro(Livro1)

DaEsquina.Inserir\_Livro(Livro2)

DaEsquina.Mostrar\_Catalogo()

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 distancia entre os dois pontos da reta.

import math

class Ponto:

def \_\_init\_\_(self, x, y):

self.\_\_x = int(x)

self.\_\_y = int(y)

@property

def x(self):

return self.\_\_x

@x.setter

def x(self, valor):

self.\_\_x = int(valor)

@property

def y(self):

return self.\_\_y

@y.setter

def y(self, valor):

self.\_\_y = int(valor)

def Imprimir\_Ponto(self):

return f"{self.\_\_x},{self.\_\_y}"

class Reta:

def \_\_init\_\_(self, a, b):

self.\_\_a = a

self.\_\_b = b

@property

def a(self):

return self.\_\_a

@a.setter

def a(self, valor):

self.\_\_a = valor

@property

def get\_b(self):

return self.\_\_b

@get\_b.setter

def b(self, valor):

self.\_\_b = valor

def distancia(self):

eixoX = math.pow((self.\_\_b.x - self.\_\_a.x), 2)

eixoY = math.pow((self.\_\_b.y - self.\_\_a.y), 2)

return math.sqrt(eixoX + eixoY)

ponto1 = Ponto(1, 1)

ponto2 = Ponto(2, 2)

reta = Reta(ponto1, ponto2)

print(f"A distancia entre o ponto ({ponto1.Imprimir\_Ponto()}) e o ponto ({ponto2.Imprimir\_Ponto()}) eh...\n{reta.distancia()}")