INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
CEARA

DEPARTAMENTO DE TELEMÁTICA
DISCIPLINA: PROGRAMAÇÃO ORIENTADA A OBJETO
LISTA EXERCICIO

ALUNO: LÍVIA EMILLY SILVA VASCONCELOS

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;

```
class Veiculo:
    def __init__(self, tipo, chassi, marca, modelo, ano):
        self.tipo = tipo
        self.chassi = chassi
        self.marca = marca
        self.modelo = modelo
        self.ano = ano

class Motocicleta(Veiculo):
    def __init__(self, tipo, chassi, marca, modelo, ano, cilindrada):
        super().__init__(tipo, chassi, marca, modelo, ano)
        self.cilindrada = cilindrada
```

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.__horas_trabalhadas = 0
        self.__salario = 0

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

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

c) Polimorfismo;

```
class Super:
        def hello(self):
                print("Olá, sou a superclasse!")
class Super (super):
        def hello(self):
                print("Olá, sou a subclasse!")
class Subsub (Sub):
        def hello(self):
                print("Olá, sou a subsubclasse!")
teste = Subsub()
teste.hello()
```

d) Variáveis de Instância;

```
def scope_test():
    def do_local():
        spam = "local spam"

    def do_nonlocal():
        nonlocal spam
        spam = "nonlocal spam"

    def do_global():
        global spam
        spam = "global spam"

    spam = "test spam"
    do_local()
    print("After local assignment:", spam)
```

```
    do_nonlocal()
    print("After nonlocal assignment:", spam)
    do_global()
    print("After global assignment:", spam)

scope_test()
print("In global scope:", spam)
```

e) Métodos construtores

```
class Carro(object):
    def __init__(self, modelo, ano): #Construtor da classe.
        self.modelo = modelo
        self.ano = ano

meuCarro = Carro("Jeep 4x4", 2015)

print("Modelo: %s" % meuCarro.modelo)
print("Ano: %d" % meuCarro.ano)
```

f) Dependência

```
def MetaIoC(name, bases, namespace):
    cls = type("IoC{}".format(name), Tuple(), namespace)
    return type(name, bases + (cls,), {})

class Entity:
    def _lower_level_meth(self):
        raise NotImplementedError

    @property
    def entity_prop(self):
        return super(Entity, self)._lower_level_meth()


class ImplementedEntity(Entity, metaclass=MetaIoC):
    __private = 'private attribute value'

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

    def _lower_level_meth(self):
        print('{}\n{}'.format(self.pub_attr, self.__private))


if __== '__main__':
    ENTITY = ImplementedEntity('public attribute value')
    ENTITY.entity_prop
```

g) Associação

```
class A(object):
    def __init__(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c

    def addNums():
        self.b + self.c

class B(object):
    def __init__(self, d, e):
        self.d = d
        self.e = e

    def addAllNums(self, Ab, Ac):
```

```
        x = self.d + self.e + Ab + Ac
        return x

ting = A("yo", 2, 6)
ling = B(5, 9)

print ling.addAllNums(ting.b, ting.c)
```

h) Relacionamento TODO-PARTE

```
class A(object):
   def __init__(self, a, b, c):
      self.a = a
      self.b = b
      self.c = c

   def addNums():
      self.b + self.c

class B(object):
   def __init__(self, d, e):
      self.d = d
      self.e = e
      self.A = A("yo", 2, 6)

   def addAllNums(self):
      x = self.d + self.e + self.A.b + self.A.c
      return x

ling = B(5, 9)

print ling.addAllNums()
```

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 distância que retorna um valor real da distância entre os dois pontos da reta.

```
class Ponto():
        def __init__(self, x, y):
                self.x = x
                self.y = y
        def setX(self,new):
                self.x = new
        def setY(self,new):
                self.y = new
        def getX(self):
                return self.x
        def getY(self):
                return self.y
class Reta():
        def __init__(self, a, b):
                self.a = a
                self.b = b
        def setA(self,new):
                self.a = new
        def setB(self,new):
                self.b = new
        def getA(self):
                return self.a
        def getB(self):
                return self.b
        def distancia(self):
                dist = (((self.getb().getx() - self.getA().getx())**2)+((self.getB().getY() - self.getA().getY())**2))**0.5

A = Ponto(2, 5)
```

```
B = Ponto(6, 8)

straightA= Reta(A, B)

print(“\nA distância entre os pontos A e B é:”straightA.distancia())
```