AC3 - Programação Orientada a Objetos

Total de pontos 7/10

Programação Orientada a Objetos

O e-mail do participante (**ricardo.csantos@aluno.faculdadeimpacta.com.br**) foi registrado durante o envio deste formulário.

✓ Dada a classe Individuo abaixo, assinale a alternativa correta: * 1/1

```
1  class Individuo:

2    def __init__(self, nome, altura, idade):
3      self.__nome = nome
4      self.altura = altura
5      nova_idade = self.idade

6  # Programa principal:
7  indiv = Individuo('Carolina', 1.59, 37)
```

- ● A linha 5 causa um erro de execução, pois o atributo público idade ainda não foi definido ✓
- ○ O construtor da classe deveria ser declarado como: def Individuo(nome, altura, idade)
- ○ A classe possui 2 atributos privados e 2 públicos
- ○ A classe possui 2 atributos privados e 1 público
- ○ O construtor da classe recebe 4 parâmetros, portanto sua chamada na linha 7 está errada

Feedback

Na linha 5, tentamos obter o valor do atributo público self.idade e atribuí-lo à variável nova_idade, mas esse atributo nunca foi definido, portanto essa linha causará um erro de execução.

✓ Sobre a classe Empresa a seguir e seus conhecimentos sobre encapsulamento, assinale a alternativa correta: *1/1

```
1   class Empresa:

2     def __init__(self, nome, cnpj):
3       self.__nome = nome
4       self.__cnpj = None
5       self.set_cnpj(cnpj)

6     def get_nome(self):
7       return self.__nome

8     def set_nome(self, nome):
9       self.__nome = nome

10    def get_cnpj(self):
11      return self.__cnpj

12    def set_cnpj(self, cnpj):
13      if len(str(cnpj)) == 14:
14        __cnpj = cnpj
```

- ● O atributo cnpj sempre terá o valor None, independente do cnpj informado por parâmetro ✓
- ○ Os atributos nome e cnpj são públicos
- ○ Chamar o método set_cnpj(cnpj) no construtor causará um erro de execução
- ○ O método get_cnpj() sempre devolve uma string
- ○ O atributo cnpj terá um valor válido somente quando o cnpj passado por parâmetro tiver tamanho 14

Feedback

O atributo cnpj sempre terá o valor None, independente do cnpj informado por parâmetro, pois a linha 4 será o único momento em que ele terá um valor atribuído (None). O método set_cnpj(cnpj) não usa o atributo privado cnpj. No lugar, ele define uma variável local chamada __cnpj. Note que para usar o atributo privado cnpj, a linha 14 deveria ser: self.__cnpj = cnpj

✓ Qual dos trechos de código abaixo completará as linhas faltantes para que seja exibida a mensagem "*Valor inválido: informe uma idade entre 26 e 75 anos*", caso o usuário informe um valor fora dessa faixa para a idade? *1/1

```
1   try:
2     idade = int(input('Informe a idade: '))
3     ------------------------:
4       --------------------------
5   except ValueError:
6     print('Valor inválido: informe uma idade entre 26 e 75 anos.')
7   except TypeError:
8     print('Erro!')
9   except:
10    print('Erro!')
```

- ○ Linha 3) if idade >= 26 and idade <= 75 ; Linha 4) raise ValueError
- ○ Linha 3) if idade >= 26 and idade <= 75 ; Linha 4) raise Exception
- ● Linha 3) if not (idade >= 26 and idade <= 75) ; Linha 4) raise ValueError ✓
- ○ Linha 3) if not (idade >= 26 and idade <= 75) ; Linha 4) raise Exception
- ○ Linha 3) if idade < 26 or idade > 75 ; Linha 4) raise Error

Feedback

A sentença lógica not (idade >= 26 and idade <= 75) será verdadeira quando a idade estiver fora da faixa entre 26 e 75 anos.

A mensagem será exibida quando o bloco try/except capturar um ValueError, portanto esse deve ser o tipo de exceção lançada.

✓ Suponha os dois arquivos Python a seguir, ambos na mesma pasta. Marque a afirmação verdadeira: *1/1

Arquivo: pessoa.py

```
1   class Pessoa:

2     def __init__(self, cpf, nome):
3       self.__cpf = cpf
4       self.__nome = nome
5
6     def imprimir_dados(self):
7       print(self.__cpf, self.__nome)

8   if __name__ == "__main__":
9     pes1 = Pessoa(1234, "Maria")
10    pes1.imprimir_dados()
```

Arquivo: principal.py

```
1   import pessoa

2   pes1 = pessoa.Pessoa(9876, "Pedro")
3   print(pessoa.__name__)
```

- ○ Ao importar o arquivo pessoa.py, será impresso na tela a string "1234 Maria"
- ○ Rodar o arquivo pessoa.py diretamente causa um erro de execução
- ● Ao rodar o arquivo principal.py diretamente, será impresso na tela a string "pessoa" ✓
- ○ Ao rodar o arquivo pessoa.py diretamente, nada será impresso na tela
- ○ Ao executar o arquivo principal.py diretamente, será impresso na tela a string "1234 Maria"

Feedback

Ao rodar o arquivo pessoa.py diretamente, será impresso na tela a string "1234 Maria", pois a variável __name__ terá o valor "__main__".

Importar o arquivo pessoa.py (como foi feito no arquivo principal.py) não imprime nada na tela, pois a variável __name__ do arquivo pessoa.py terá o valor "pessoa", portanto o if será falso.

Já o arquivo principal.py importa o arquivo pessoa.py (linha 1). Ao instanciar um novo objeto da classe Pessoa, nada será impresso (linha 2). Já ao imprimir o valor de pessoa.__name__, será impresso na tela a string "pessoa" (linha 3).

✗ Assinale a afirmação correta sobre a classe Paciente e o objeto **p** instanciado a partir dela, conforme o código a seguir: *0/1

```
class Paciente:

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

  @property
  def altura(self):
    return self.__altura

  @altura.setter
  def altura(self, valor):
    if valor < 1:
      valor = 1.0
    self.__altura = valor

# Programa principal
p = Paciente("Bia", 0.4)
print(p.altura)
```

- ● O objeto p possui um atributo privado: nome com o valor "Bia" e um atributo público: altura com o valor "1.0" ✗
- ○ Apesar de estar definido na classe, o setter altura não é executado neste exemplo
- ○ O objeto p possui um atributo privado: nome com o valor "Bia" e um atributo público: altura com o valor "0.4"
- ○ O objeto p possui dois atributos privados: nome com o valor "Bia" e altura com o valor "1.0"
- ○ O objeto p possui dois atributos privados: nome com o valor "Bia" e altura com o valor "0.4"

Resposta correta

- ● O objeto p possui dois atributos privados: nome com o valor "Bia" e altura com o valor "1.0"

Feedback

A classe Paciente cria dois atributos privados: nome, diretamente construtor, e altura, definido de maneira indireta através do setter. O setter é chamado quando fazemos a atribuição self.altura = altura. Como existe um setter definido com esse nome (altura), a classe o chama ao invés de criar um atributo público.
O setter também garante que a altura mínima de um paciente não pode ser menor que 1.0.

✓ A respeito da classe **Televisao** e do programa principal escrito a seguir, assinale a afirmação FALSA: *1/1

```
1   class Televisao:

2     def __init__(self, modelo, preco):
3       self.set_modelo(modelo)
4       self.set_preco(preco)

5     def get_modelo(self):
6       return self.__modelo
7
8     def get_preco(self):
9       return self.__preco

10    def set_modelo(self, valor):
11      self.__modelo = valor
12
13    def set_preco(self, valor):
14      if valor < 1000:
15        raise ValueError
16      else:
17        self.__preco = valor

18  # Programa principal
19  try:
20    tv = Televisao("TV123ABC", 900)
21    print(tv.get_modelo())
22    print(tv.get_preco())
23  except ValueError:
24    print("Não é possível instanciar a TV")
25  except:
26    print("Erro genérico")
```

- ○ A classe permite instanciar, sem erros, o objeto: tv2 = Televisao("XYZ", 1234)
- ● Um erro do tipo ValueError é gerado ao mudar o código da linha 20 para: tv = Televisao("TV123ABC", 1000) ✓
- ○ O programa imprime a string "Não é possível instanciar a TV"
- ○ Ao instanciar um objeto da classe Televisao, um ValueError será gerado ao chamar o método set_preco() com valor igual a 100
- ○ As instruções print() nas linhas 21 e 22 não serão executadas

Feedback

Ao instanciar um objeto da classe Televisao, o construtor inicializa os atributos privados modelo e preco a partir dos métodos set_modelo() e set_preco(), respectivamente.
O método set_preco() verifica se o preço é menor que 1000, e se isso for verdadeiro, lança uma exceção do tipo ValueError. Isso significa que se o preço for maior ou igual a 1000, essa verificação será falsa, e nenhuma exceção será gerada, permitindo que o construtor termine a sua execução e o objeto seja instanciado com sucesso.

Em resumo, a classe Televisao permite instanciar qualquer objeto cujo preço seja maior ou igual a 1000.

✓ Assinale a alternativa FALSA sobre as classes **Produto** e **ProdutoFisico** mostradas a seguir: *1/1

```
class Produto:

  def __init__(self, descricao, preco):
    self.__descricao = descricao
    self.__preco = preco

  @property
  def descricao(self):
    return self.__descricao

  @property
  def preco(self):
    return self.__preco

  @descricao.setter
  def descricao(self, valor):
    self.__descricao = valor

  @preco.setter
  def preco(self, valor):
    self.__preco = valor


class ProdutoFisico(Produto):

  def __init__(self, descricao, preco, peso):
    super().__init__(descricao, preco)
    self.__peso = peso

  @property
  def peso(self):
    return self.__peso

  @peso.setter
  def peso(self, valor):
    self.__peso = valor
```

- ○ Um objeto prod instanciado a partir da classe ProdutoFisico possui a propriedade (property) descricao, e pode ter seu valor acessado com o comando: prod.descricao
- ○ A classe ProdutoFisico apresenta um exemplo da aplicação do conceito de herança
- ● Um objeto prod instanciado a partir da classe Produto possui a propriedade (property) peso, e pode ter seu valor acessado com o comando: prod.peso ✓
- ○ Um objeto prod instanciado a partir da classe ProdutoFisico possui a propriedade (property) preco, e pode ter seu valor acessado com o comando: prod.preco
- ○ O construtor da classe ProdutoFisico apresenta um exemplo da aplicação do conceito de polimorfismo

Feedback

A classe ProdutoFisico herda da classe Produto. Em seu construtor, há um exemplo de polimorfismo, uma vez que os construtores possuem o mesmo nome. Através da função super(), reaproveitamos o construtor da classe Produto para inicializar a descricao e preco do produto.

A classe Produto só possui duas properties (propriedades): descricao e preco, e ela não herda de nenhuma outra classe que possui a property peso. Quem possui as três properties é a classe ProdutoFisico: descricao e preco (herdadas da classe Produto) e peso, definida em seu código.

✓ Dada a hierarquia de classes abaixo e os seus conhecimentos sobre herança, assinale a resposta correta: *1/1

```
class Maquina:
  def __init__(self, peso):
    self.peso = peso

class Veiculo(Maquina):
  def __init__(self, peso, placa):
    super().__init__(peso)
    self.placa = placa

  def get_placa(self):
    return self.placa

  def set_placa(self, placa):
    self.placa = placa

class Carro(Veiculo):
  def __init__(self, peso, placa, qtd_portas):
    super().__init__(peso, placa)
    self.qtd_portas = qtd_portas

class Onibus(Veiculo):
  def __init__(self, peso, placa, qtd_assentos):
    super().__init__(peso, placa)
    self.qtd_assentos = qtd_assentos
```

- ○ A herança está feita de maneira errada, pois só uma classe pode herdar da classe Veiculo
- ○ A classe Maquina consegue acessar o atributo placa
- ○ A classe Onibus não possui os métodos get_placa() e set_placa(placa)
- ○ A classe Carro não possui os métodos get_placa() e set_placa(placa)
- ● Objetos instanciados a partir das classes Carro e Onibus possuem os atributos placa e peso ✓

Feedback

Objetos instanciados a partir das classes Carro e Onibus possuem os atributos placa e peso, pois essas classes herdam da classe Veiculo, que possui o atributo público placa. Por sua vez, a classe Veiculo herda da classe Maquina, que possui o atributo público peso.
Desta forma, toda classe que herdar de Veiculo possuirá os atributos públicos placa e peso.

✗ Analise o código classe **Relogio**, em seguida assinale a afirmação FALSA sobre ela: *0/1

```
class Relogio:

  def __init__(self, hora, minuto):
    self.hora = hora
    self.minuto = minuto

  @property
  def hora(self):
    return self.__hora

  @property
  def minuto(self):
    return self.__minuto

  @hora.setter
  def hora(self, nova_hora):
    if nova_hora >= 0 and nova_hora <= 23:
      self.__hora = nova_hora
    else:
      raise ValueError

  @minuto.setter
  def minuto(self, novo_minuto):
    if novo_minuto >= 0 and novo_minuto <= 59:
      self.__minuto = novo_minuto
    else:
      raise ValueError

  def tictac(self):
    self.__minuto += 1
    if self.__minuto > 59:
      self.__minuto = 0
      self.__hora += 1
      if self.__hora > 23:
        self.__hora = 0
```

- ○ Ao chamar o método tictac(), o tempo do relógio avança em exatamente 1 minuto. É garantido que a hora sempre estará entre 0 e 23, e o minuto entre 0 e 59
- ● Ao chamar o construtor (através da instanciação de um novo objeto), serão criados dois atributos privados. O construtor cria esses atributos através dos setters hora e minuto ✗
- ○ Tentar instanciar os objetos r1 = Relogio(24, 0) e r2 = Relogio(23, 60) faz com que a classe lance uma exceção do tipo ValueError
- ○ A classe define dois atributos públicos: hora e minuto, que armazenam essas respectivas informações
- ○ O construtor valida os valores de hora e minuto usando os setters hora e minuto, respectivamente

Resposta correta

- ● A classe define dois atributos públicos: hora e minuto, que armazenam essas respectivas informações

Feedback

Apesar de parecer que a classe Relogio está definindo dois atributos públicos, uma vez que definimos os getters e setters hora e minuto através de properties, são eles que serão chamados pelo construtor.
Esses setters por sua vez fazem uma validação dos dados, e caso eles estejam dentro da faixa correta, criam os atributos privados.
Em resumo, estamos chamando os setters hora e minuto dentro do construtor, e dessa forma fazendo a validação dos dados.

✗ Avalie o código a seguir e marque a alternativa que descreve corretamente o que ocorre ao executarmos o arquivo **programa.py**. Assuma que todos os arquivos estão na mesma pasta. *0/1

Arquivo: programa.py

```
from carro import Carro
from motorista import Motorista

m = Motorista()
c = Carro()
m.dirige(c)
```

Arquivo: motorista.py

```
class Motorista:
  def dirige(self, carro):
    carro.acelerar()
    carro.freiar()
```

Arquivo: carro.py

```
class Carro:
  def acelerar(self):
    print("Acelerando...")

  def freiar(self):
    print("Freiando...")
```

- ○ Um erro de execução, pois não foram passados os argumentos obrigatórios ao instanciar o objeto de Carro
- ○ Um erro de execução, pois não é possível passar um objeto como argumento para um método
- ○ Executa sem nenhum erro e exibe na tela os textos: "Acelerando..." e "Freiando..."
- ○ Executa sem nenhum erro e não exibe nada na tela
- ● Um erro de execução, pois não foram passados os argumentos obrigatórios ao instanciar o objeto de Pessoa ✗

Resposta correta

- ● Executa sem nenhum erro e exibe na tela os textos: "Acelerando..." e "Freiando..."

Feedback

O arquivo principal.py importa corretamente as classes: Motorista, do arquivo motorista.py e Carro, do arquivo carro.py.
As classes não definem nenhum construtor, mas elas são instanciadas sem nenhum problema ao fazer fazer atribuições como: m = Motorista() e c = Carro(). Elas só não terão nenhum atributo.

O método dirige() da classe Motorista recebe um objeto carro por parâmetro, e já espera que esse objeto contenha os métodos acelerar() e freiar().

Portanto, ao chamar o método dirige() de um objeto da classe Motorista, será exibido na tela as mensagens: "Acelerando..." e "Freiando...", respectivamente.