Faculdade IMPACTA

Alex Silva de Sousa <alex.ssousa@aluno.faculdadeimpacta.com.br>

Atividade Contínua 02 - Programação Orientada a Objetos

1 mensagem

Formulários Google <forms-receipts-noreply@google.com> 29 de fevereiro de 2024 às 22:25
Para: alex.ssousa@aluno.faculdadeimpacta.com.br

Agradecemos o preenchimento de Atividade Contínua 02 - Programação Orientada a Objetos

Veja as respostas enviadas.

Atividade Contínua 02 - Programação Orientada a Objetos

As questões contidas nessa atividade estão relacionadas aos conteúdos das partes 04, 05, 06 e 07.

Seu e-mail (**alex.ssousa@aluno.faculdadeimpacta.com.br**) foi registrado quando você enviou este formulário.

Seção 1 - Conceitos de POO

Marque a alternativa que descreve CORRETAMENTE o conceito de encapsulamento. *

- (●) É quando ocultamos do mundo exterior (do restante da aplicação), atributos ou métodos de uma classe, que devem então ser acessados apenas internamente, por outros métodos da própria classe.
- ( ) É quando criamos atributos com valores iniciais padronizados.
- ( ) É a capacidade de objetos de classes diferentes possuírem métodos de mesmo nome e assinatura, mas com comportamentos diferentes.
- ( ) É a capacidade de uma classe reaproveitar a estrutura de outra classe.

- ( ) Não existe esse conceito em POO.

Marque a alternativa que descreve CORRETAMENTE o conceito de herança. *

- (•) É quando uma classe reaproveita a estrutura de outra classe mais geral, e podemos adicionar características e comportamentos extras conforme necessário.
- ( ) É quando duas classes são criadas no mesmo arquivo Python.
- ( ) É quando um classe é importada de outro módulo no Python.
- ( ) É quando instanciamos objetos de uma classe.
- ( ) Não existe esse conceito em POO.

Marque a alternativa que descreve CORRETAMENTE o conceito de polimorfismo. *

- ( ) É o suporte à herança.
- ( ) É a capacidade da linguagem de programação em tornar os atributos ou métodos privados.
- ( ) É quando uma classe herda a estrutura de outra classe.
- (•) É a capacidade de objetos de classes diferentes possuírem os mesmos métodos (ações), mas com comportamentos internos diferentes para cada tipo de objeto.
- ( ) Não existe esse conceito em POO.

Seção 2 - Conceitos de POO em Python

Marque a alternativa que descreve CORRETAMENTE como é implementado o polimorfismo de sobrescrita em Python. *

- (•) Ao criar uma classe que herda de outra, podemos redefinir na classe filha os métodos para os quais desejamos alterar o comportamento interno, mantendo a mesma assinatura do método equivalente da classe mãe.
- ( ) Ao criar uma classe que herda de outra, precisamos alterar os nomes dos métodos que desejamos sobrescrever.
- ( ) Devemos adicionar o decorador @polymorphism ao método que desejamos alterar.

- ( ) Devemos adicionar o decorador @doublemethod ao método que desejamos alterar.
- ( ) Não é possível implementar o polimorfismo em Python.

Ao definirmos um método em uma classe, qual deve ser obrigatoriamente o primeiro parâmetro. *

- (•) O primeiro parâmetro deve ser uma referência ao objeto a partir do qual o método será usado, normalmente usa-se a palavra self.
- ( ) O primeiro parâmetro deve ser o nome do próprio objeto, normalmente usa-se a palavra name.
- ( ) O primeiro parâmetro deve ser uma referência para o módulo no qual o objeto é importado, normalmente usa-se a palavra module_id
- ( ) O primeiro parâmetro deve ser uma referência ao gerenciador de contexto do Python, normalmente usa-se a palavra this.
- ( ) Não há nenhuma regra quando ao primeiro parâmetro de métodos de uma classe em Python.

Seção 4 - Classes e objetos em Python

Veja a imagem a seguir e marque a alternativa que apresenta a sintaxe correta para criação de uma classe com herança em Python *

parênteses	`class ClasseDerivada(ClasseBase):` `  pass`
seta	`class ClasseDerivada <- ClasseBase:` `  pass`
dois pontos	`class ClasseDerivada: ClasseBase` `  pass`
extends	`class ClasseDerivada extends ClasseBase:` `  pass`
from	`class ClasseDerivada from ClasseBase:` `  pass`

- (•) parênteses
- ( ) seta

- ( ) dois pontos
- ( ) extends
- ( ) from

O que irá acontecer ao ser executado o seguinte trecho de código? *

```
class Conta:
    def __init__(self, numero, titular):
        self.numero = numero
        self.titular = titular

conta = Conta()
```

- ( ) Executa sem nenhum erro, criando o objeto conta, mas nada é exibido na tela.
- ( ) Executa sem nenhum erro, exibindo o nome do titular e o número da conta na tela.
- ( ) Apresenta um erro de sintaxe na primeira linha, pois a palavra chave class está escrita errada.
- ( ) Apresenta um erro de execução na segunda linha, pois o método __init__ pode ter apenas 1 único parâmetro, que é o self.
- (•) Apresenta um erro de execução na última linha, pois não foram passados os argumentos obrigatórios (número da conta e nome do titular) para criar uma instância da classe Conta.

O que irá acontecer ao ser executado o seguinte trecho de código? *

```
class Conta:
    def __init__(self, numero, titular):
        self.numero = numero
        self.titular = titular

conta = Conta(3002, 'Megan')
```

- (•) Executa sem nenhum erro, criando o objeto conta, mas nada é exibido na tela.
- ( ) Executa sem nenhum erro, exibindo o nome do titular e o número da conta na tela.

- Apresenta um erro de sintaxe na primeira linha, pois a palavra chave class está escrita errada.
- Apresenta um erro de execução na segunda linha, pois o método __init__ pode ter apenas 1 único parâmetro, que é o self.
- Apresenta um erro de execução na última linha, pois não foram passados os argumentos obrigatórios (número da conta e nome do titular) para criar uma instância da classe Conta.

Avalie o código a seguir e marque a alternativa que descreve corretamente o que ocorre ao executarmos o arquivo main.py. *

O código representa 3 arquivos distintos, cujo nome é dado nos comentários, e assuma que todos os arquivos estão na mesma pasta.

```
##################################
####### arquivo main.py #######
##################################
from pessoa import Pessoa
from carro import Carro

p = Pessoa()
c = Carro()

p.dirige(c)

##################################
###### arquivo pessoa.py ######
##################################
class Pessoa:
    def dirige(self, carro):
        carro.acelera()
        carro.freia()

##################################
###### arquivo carro.py #######
##################################
class Carro:
    def acelera(self):
        print('acelerando...')

    def freia(self):
        print('freiando...')
```

- ( ) Um erro de execução pois não foram passados os argumentos obrigatórios ao instância o objeto de Pessoa.
- ( ) Um erro de execução pois não foram passados os argumentos obrigatórios ao instância 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 não exibe nada na tela.
- (•) Executa sem nenhum erro e exibe na tela os textos "acelerando..." e "freiando...".

Seção 3 - Módulos

Sobre modularização, é CORRETO afirmar que: *

- ( ) A divisão de um projeto em módulos é uma má prática.
- ( ) Módulos devem ser utilizados apenas em último caso, pois pioram o desempenho do código. O ideal é sempre escrever todo o código do projeto em um único arquivo.
- ( ) Para criar um módulo em Python é preciso instalar uma biblioteca externa chamada create_modules.
- ( ) Não é possível dividir um projeto Python em módulos.
- (•) Todo arquivo Python já é automaticamente um módulo, e a divisão de um projeto em módulos ajuda na organização do código e separação de responsabilidades.