Matéria: Tecnologia da Informação

Assunto: Programação e Linguagens de Programação

Resumo Teórico do Assunto

Este resumo aborda conceitos fundamentais de programação e linguagens, focando em aspectos de Machine Learning com Python, manipulação de strings em Swift e princípios de Orientação a Objetos em Java.

---

1. Python e Machine Learning (Regressão Linear)

A questão 36 explora o uso de Python para **Machine Learning (ML)**, especificamente a **Regressão Linear**, utilizando as bibliotecas **NumPy** e **Scikit-learn**.

• **Machine Learning (ML)**: É um campo da Inteligência Artificial que permite que sistemas aprendam a partir de dados, identificando padrões e fazendo previsões ou decisões com mínima intervenção humana.
• **Regressão Linear**: É uma técnica de ML supervisionado utilizada para modelar a relação entre uma variável dependente (geralmente denotada por `y`) e uma ou mais variáveis independentes (denotadas por `x`). O objetivo é encontrar a linha reta que melhor se ajusta aos dados, expressa pela equação `y = ax + b`, onde:
* `a` é o **coeficiente angular** (inclinação da linha), também conhecido como `coef_` no Scikit-learn.
* `b` é o **coeficiente linear** (interseção com o eixo y), também conhecido como `intercept_` no Scikit-learn.
• **NumPy**: É uma biblioteca fundamental em Python para computação numérica, especialmente para trabalhar com arrays multidimensionais (vetores e matrizes). A função `reshape((-1, 1))` é comumente usada para transformar um array unidimensional em um array bidimensional de uma coluna, formato exigido por muitas funções do Scikit-learn para variáveis independentes. O `-1` indica que o NumPy deve inferir o número de linhas.
• **Scikit-learn (skl)**: É uma das bibliotecas mais populares em Python para Machine Learning. Ela oferece uma vasta gama de algoritmos de classificação, regressão, agrupamento e pré-processamento de dados.
* **`sklearn.linear_model.LinearRegression`**: É a classe dentro do Scikit-learn que implementa o algoritmo de Regressão Linear.
* **Instanciação do Modelo**: Para usar um modelo do Scikit-learn, primeiro você precisa criar uma instância da classe do modelo. Por exemplo: `model = skl.LinearRegression()`. Note que `LinearRegression()` é uma chamada de construtor, por isso requer parênteses.
* **Treinamento do Modelo (`fit()` método)**: Após instanciar o modelo, ele precisa ser "treinado" com os dados. O método `fit(X, y)` é usado para este propósito, onde `X` são as

variáveis independentes (features) e `y` é a variável dependente (target). Este método ajusta os parâmetros internos do modelo (como `a` e `b` na regressão linear) para que ele aprenda a relação entre `X` e `y`.
* **Acesso aos Parâmetros Treinados**: Após o treinamento, os parâmetros aprendidos podem ser acessados como atributos do objeto `model`:
* `model.coef_`: Retorna o coeficiente angular (`a`).
* `model.intercept_`: Retorna o coeficiente linear (`b`).

---

2. Swift e Interpolação de Strings

A questão 37 aborda a manipulação de **Strings** e o conceito de **Interpolação de Strings** na linguagem Swift.

• **Strings**: São sequências de caracteres, como letras, números e símbolos, usadas para representar texto.
• **Interpolação de Strings**: É um recurso que permite incorporar valores de variáveis, constantes ou resultados de expressões diretamente dentro de uma string literal. Isso torna a construção de strings dinâmicas muito mais legível e concisa do que a concatenação tradicional.
• **Sintaxe em Swift**: Em Swift, a interpolação de strings é realizada utilizando a sintaxe `\(expressão)`. Qualquer expressão válida pode ser colocada dentro dos parênteses, e seu valor será convertido para uma string e inserido no local correspondente.
• **`let`**: Em Swift, a palavra-chave `let` é usada para declarar uma **constante**, ou seja, um valor que não pode ser alterado após sua inicialização.

---

3. Java e Programação Orientada a Objetos (Interfaces e Herança)

A questão 38 explora conceitos fundamentais da **Programação Orientada a Objetos (POO)** em Java, especificamente a diferença entre **Interfaces** e **Herança**.

• **Programação Orientada a Objetos (POO)**: É um paradigma de programação que organiza o código em "objetos", que são instâncias de "classes". Os objetos encapsulam dados (atributos) e comportamentos (métodos).
• **Classe**: É um molde ou um projeto para criar objetos. Ela define a estrutura (atributos) e o comportamento (métodos) que os objetos criados a partir dela terão.
• **Interface**: Em Java, uma **interface** é um "contrato" que define um conjunto de métodos que uma classe *deve* implementar. Uma interface não contém a implementação dos métodos (apenas suas assinaturas), nem pode ter atributos de instância (apenas constantes estáticas e finais). O propósito principal das interfaces é definir um comportamento comum que várias classes podem aderir, promovendo o polimorfismo.
* **`implements`**: A palavra-chave `implements` é usada por uma classe para indicar que ela

está aderindo ao contrato de uma ou mais interfaces, ou seja, que ela fornecerá a implementação para todos os métodos abstratos definidos nessas interfaces. Uma classe pode implementar múltiplas interfaces.
* Exemplo: `class MinhaClasse implements MinhaInterface`
• **Herança**: É um mecanismo da POO que permite que uma classe (chamada **subclasse** ou classe filha) adquira as características (atributos e métodos) de outra classe (chamada **superclasse** ou classe pai). Isso promove a reutilização de código e estabelece uma relação "é um tipo de" (e.g., um "Carro é um tipo de Veículo").
* **`extends`**: A palavra-chave `extends` é usada por uma classe para indicar que ela está herdando de outra classe. Em Java, uma classe só pode estender uma única superclasse (herança simples).
* Exemplo: `class Carro extends Veiculo`

Diferença Chave:
• Use `implements` quando uma classe precisa **cumprir um contrato** ou **fornecer um comportamento específico** definido por uma interface.
• Use `extends` quando uma classe precisa **herdar características e comportamentos** de outra classe, estabelecendo uma relação de especialização.

Questões de Provas Anteriores

Fonte: escriturario_agente_de_tecnologia.pdf, Página: 12

pcimarkpci MjgwNDowMTRkOjE0YTU6OTI1ODozOGQ2OjNhMGM6NTM0MzplZmI1:U3VuLCAyNyBKdWwgMjAyNSAyMzo0NzozMSAtMDMwMA==
www.pciconcursos.com.br
12
BANCO DO BRASIL
AGENTE DE TECNOLOGIA - Microrregião 158 -TI GABARITO 1
TECNOLOGIA DA INFORMAÇÃO
36
Analise o código a seguir, feito em Python com o Scikit-learn.

```
import numpy as np
import sklearn.linear_model as skl
base = np.array([1, 2, 3, 4, 5, 6])
x = base.reshape((-1, 1))
y = base*2+3
# a fazer
print('a', model.coef_[0])
print('b', model.intercept_)
```

A partir desse código, um programador quer obter os parâmetros a e b da equação y = ax + b, por meio de uma regressão
linear, usando, para isso, os dados nos vetores x e y definidos no programa.

Qual linha de código deve substituir o comentário # a fazer de modo a realizar essa regressão linear?

(A) model = skl.lr(x, y)
(B) model = skl.lr().fit(x, y)
(C) model = skl.LinearRegression(x, y)
(D) model = skl.LinearRegression([x, y])
(E) model = skl.LinearRegression().fit(x, y)

37

Em um programa em Swift, o programador deseja incluir o resultado de uma operação dentro de uma string.
Nesse contexto, considere o seguinte código:
let quantidade = 4
let valor = 10
Dado o código acima, o programador deseja uma string saida cujo valor seja
"valor total = 40"
Para isso, o programador deve utilizar o seguinte fragmento de código Swift:
(A) let saida := "valor total = \(quantidade*valor)"
(B) let saida := "valor total = \{quantidade*valor}"
(C) let saida = "valor total = %[quantidade*valor]"
(D) let saida = "valor total = \(quantidade*valor)"
(E) let saida = "valor total = \[quantidade*valor]"

38

Um programador foi instruído pelo seu gerente a implementar, em Java, uma classe MemoriaCalculoVenda que imple-
mentasse a interface MemoriaCalculo, já criada pela organização e que representa as exigências da organização para
classes que implementam memórias de cálculo.
Nesse cenário, com que fragmento de código o programador deve começar, de forma correta, a implementação da classe?
(A) class MemoriaCalculoVenda extends MemoriaCalculo
(B) class MemoriaCalculoVenda implements MemoriaCalculo
(C) class MemoriaCalculoVenda imports MemoriaCalculo
(D) class MemoriaCalculoVenda inherits MemoriaCalculo
(E) class MemoriaCalculoVenda uses MemoriaCalculo