



ChatGPT 4o

Share MIS

Explore GPTs

Projects

- SRES
- Nahiyah Sir
- Opi Sir
- RL_Offloading(Dont edi...)

Today

Slide Content Summary

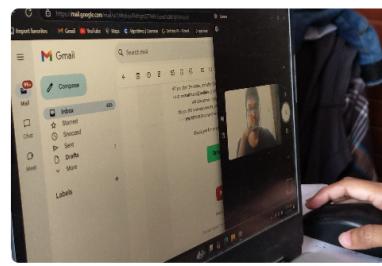
Online Exam Proctoring ...

MCQ Test Assistance

Array Syntax Issue

Yesterday

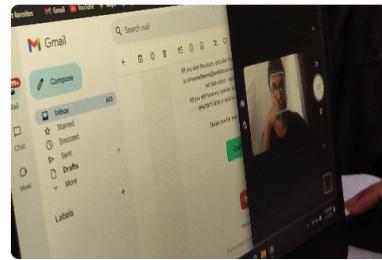
Unknown error troubleshootin



Explain this

The image shows a person using a laptop, with their Gmail inbox open on the screen. The email appears to contain instructions related to an exam, possibly an MCQ-based test for a trainee position, as seen in the browser tab title. Additionally, the laptop screen has a small window open displaying a camera feed, which seems to be using face detection technology, indicating a proctoring or monitoring system for the exam.

This suggests that the person is either preparing for or currently taking an online proctored test, where their identity and behavior are being monitored to prevent cheating. The presence of a verification frame around their face implies that facial recognition or tracking is in use.



What is written in gmail mail?

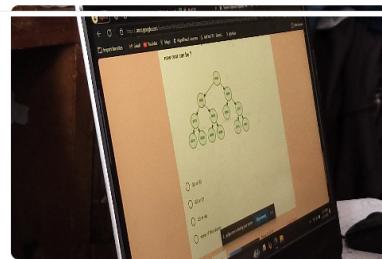
The email text is somewhat blurry, but from what is visible, it appears to contain instructions regarding an online test or exam. The email likely includes guidelines for candidates, mentioning what to do if they face issues during the test and providing a contact email or phone number for assistance.

Key phrases that can be identified:

- "If you start the exam, and that ..."
- "us at recruitmen[...]"
- "will take[...]"
- "If you still have any queri[...]"
- "Thank you for your[...]"

The presence of a green button suggests a call-to-action, possibly to start the test or access further instructions.

If you need an exact transcription, you can try enhancing the image or zooming in for better readability.



Solve the question though it look blurry

The question displayed on the screen appears to be about a **Binary Search Tree (BST)**, asking which node can be the new root.

To solve this:

1. **Understand the BST Structure:** The given tree follows the binary search properties where the left subtree

1. **Identify the BST Structure.** The given tree follows the binary search property where the left subtree has smaller values and the right subtree has larger values.

2. **Possible New Root Conditions:**

- The root of a BST can be changed while maintaining the BST properties.
- The new root must be a node that can preserve order and allow restructuring.

Likely Answer:

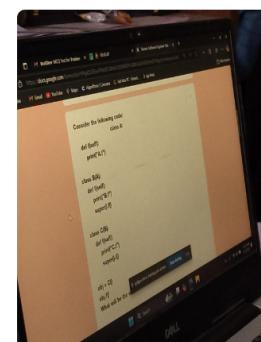
From the visible part of the tree, the possible new root choices appear to be among:

- 30
- 50
- 63
- 43

Best choice:

- 50 or 63 seems reasonable as a new root because they allow for balanced restructuring while keeping BST properties.

If you can describe the tree structure clearly, I can confirm the best choice with certainty.



What will be the output

Let's analyze the given Python code:

```
python
Copy Edit

class A:
    def f(self):
        print("A.f")

class B(A):
    def f(self):
        print("B.f")
        super().f()

class C(B):
    def f(self):
        print("C.f")
        super().f()

obj = C()
obj.f()
```

Understanding Execution Flow

1. `obj = C()` → Creates an instance of class `C`.
2. `obj.f()` → Calls `f()` method of class `C`, which:
 - Prints `"C.f"`
 - Calls `super().f()`, which refers to `B.f()` (parent class method).
3. `B.f()` executes:
 - Prints `"B.f"`
 - Calls `super().f()`, which refers to `A.f()` (parent class method).
4. `A.f()` executes:
 - Prints `"A.f"`
 - Since `A.f()` does not call `super().f()`, the method chain stops here.

Final Output

```
css
```

Copy Edit

```
C.f  
B.f  
A.f
```

So, the correct answer is:

```
css
```

Copy Edit

```
C.f  
B.f  
A.f
```

How many types of polymorphism in oop?

In Object-Oriented Programming (OOP), **polymorphism** allows a single interface to represent different underlying data types. There are mainly **two types of polymorphism**:

1. Compile-time Polymorphism (Static Polymorphism)

- This type of polymorphism is resolved at **compile-time**.
- Achieved through **method overloading** and **operator overloading**.
- Example in Python (method overloading isn't natively supported but can be mimicked):

```
python
```

Copy Edit

```
class Math:  
    def add(self, a, b, c=0):  
        return a + b + c  
  
obj = Math()  
print(obj.add(2, 3)) # Output: 5  
print(obj.add(2, 3, 4)) # Output: 9
```

- Example in C++ (operator overloading):

```
cpp
```

Copy Edit

```
class Complex {  
public:  
    int real, imag;  
    Complex(int r, int i) : real(r), imag(i) {}  
    Complex operator + (const Complex& obj) {  
        return Complex(real + obj.real, imag + obj.imag);  
    }  
};
```

2. Runtime Polymorphism (Dynamic Polymorphism)

- This type of polymorphism is resolved at **runtime**.
- Achieved through **method overriding**.
- Example in Python:

```
python
```

Copy Edit

```
class Animal:  
    def sound(self):  
        print("Some generic animal sound")  
  
class Dog(Animal):  
    def sound(self):  
        print("Bark")  
  
obj = Dog()  
obj.sound() # Output: Bark
```

Other Classifications (Less Common but Useful)

Some sources further classify polymorphism into:

- **Subtype Polymorphism**: Derived class objects can be treated as instances of the base class.
- **Parametric Polymorphism**: Using **generics** in languages like Java and C++.

```
java
```

Copy Edit

```

class Box<T> {
    T obj;
    void add(T obj) { this.obj = obj; }
}

```

- **Ad-hoc Polymorphism:** Function/operator overloading where different functions have the same name but work with different data types.

Conclusion

The two main types of polymorphism in OOP are:

1. **Compile-time (Static) Polymorphism** → Method Overloading, Operator Overloading
2. **Runtime (Dynamic) Polymorphism** → Method Overriding

Let me know if you need further clarifications!



This is a question
What makes person class immutable?

What Makes a Class Immutable?

A class in Java is considered **immutable** if its state (i.e., the values of its fields) **cannot be changed after object creation**.

Analyzing the Person Class

From the images, the `Person` class contains:

- **Private fields** (`name` and `age`)
- **A constructor** that initializes these fields
- **Getter methods** (`getName()` and `getAge()`)
- **Setter methods** (`setName()` and `setAge()`)

Why is the Person Class Not Immutable?

The presence of **setter methods** (`setName()` and `setAge()`) allows **modification** of the instance variables after object creation, making the class **mutable**.

How to Make Person Class Immutable?

To make `Person` immutable:

1. Remove the **setter methods** (`setName()` and `setAge()`).
2. Declare **fields** as `final` to ensure they cannot be changed after object initialization.
3. Avoid exposing **mutable objects** (if present).

Immutable Person Class:

```

java                                     ⌂ Copy ⌂ Edit

public final class Person {
    private final String name;
    private final int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }
}

```

Key Changes:

- ✓ `final` keyword for class → Prevents subclassing.
- ✓ `final` keyword for fields → Ensures fields cannot be changed after initialization.
- ✓ No **setter methods** → Prevents modification after object creation.

Conclusion

The given `Person` class is **mutable** because it has setter methods. To make it **immutable**, remove the setters and use `final` fields.

time complexity of binary search

Time Complexity of Binary Search

Binary Search is an efficient algorithm used to find an element in a **sorted** array. It follows the **divide and conquer** approach, repeatedly dividing the search space in half.

Steps of Binary Search:

1. Compare the target value with the middle element.
2. If they match, return the index.
3. If the target is smaller, search the left half.
4. If the target is larger, search the right half.
5. Repeat until the element is found or the search space is empty.

Time Complexity Analysis

Each time we divide the array in half, reducing the problem size from n to $n/2$, then to $n/4$, and so on. This continues until the search space reduces to 1 element.

The recurrence relation for Binary Search:

$$T(n) = T(n/2) + O(1)$$

Solving this:

- After k iterations, the size of the search space becomes $n / 2^k$.
- The search stops when $n / 2^k = 1$, meaning:

$$2^k = n$$

- Taking **log base 2** on both sides:

$$k = \log_2(n)$$

Thus, **time complexity** = $O(\log n)$.

Best, Worst, and Average Cases

Case	Time Complexity	Explanation
Best Case	$O(1)$	When the middle element is the target.
Worst Case	$O(\log n)$	When the element is at the last possible position.
Average Case	$O(\log n)$	On average, we perform $\log(n)$ comparisons.

Space Complexity

- **Iterative Binary Search:** $O(1)$ (only uses a few variables).
- **Recursive Binary Search:** $O(\log n)$ (due to recursive function calls using stack memory).

Conclusion

Binary Search is very efficient, with a worst-case time complexity of $O(\log n)$, making it much faster than linear search $O(n)$ for large datasets. However, it requires the array to be **sorted** beforehand.

Time complexity of floyed warshall algoeithm to calculate shortest path in a graph of n vertices?
Is it theta(n^3)?

Time Complexity of Floyd-Warshall Algorithm

The **Floyd-Warshall Algorithm** is used to find the shortest paths between all pairs of vertices in a weighted graph. It follows a **dynamic programming approach**.

Time Complexity Analysis

The algorithm uses **three nested loops** that iterate over the number of vertices n :

```
cpp
for (int k = 0; k < n; k++) {           // Iterates over intermediate vertices
    for (int i = 0; i < n; i++) {         // Iterates over source vertices
        for (int j = 0; j < n; j++) {     // Iterates over destination vertices
            dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
        }
    }
}
```

Each loop runs $O(n)$ times, leading to a total complexity of:

$$O(n) \times O(n) \times O(n) = O(n^3)$$

Since the number of operations is **proportional to n^3** and is bounded both above and below by $c_1 n^3$ and $c_2 n^3$ for some constants c_1, c_2 , we can conclude:

$$\Theta(n^3)$$

Thus, the **asymptotic time complexity of the Floyd-Warshall algorithm is indeed $\Theta(n^3)$** .

What is the example of creational design pattern?

Singleton, observer, iterator, strategy?

In the given options, the **creational design pattern** is:

Singleton Pattern

Explanation of Each Pattern:

1. Singleton (Creational Pattern)

- Ensures only **one instance** of a class exists.
- Provides a **global access point**.
- Example:

```
java
public class Singleton {
    private static Singleton instance;

    private Singleton() { }

    public static Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }
}
```

- Category: **Creational Pattern** (Manages object creation)

2. Observer (Behavioral Pattern)

- Defines a **one-to-many dependency** between objects.
- Example: When an event occurs, multiple subscribers (observers) get notified.
- Category: **Behavioral Pattern** (Focuses on communication between objects)

3. Iterator (Behavioral Pattern)

- Provides a way to **access elements** of a collection **sequentially** without exposing its structure.
- Example: **Iterator** in Java:

```
java
Iterator<String> it = list.iterator();
while (it.hasNext()) {
    System.out.println(it.next());
}
```

- Category: Behavioral Pattern (Manages iteration over data structures)

4. Strategy (✗ Behavioral Pattern)

- Defines a family of algorithms and lets the client choose the appropriate one at runtime.
- Example: Different sorting strategies (QuickSort, MergeSort).
- Category: Behavioral Pattern (Encapsulates interchangeable behaviors)

Final Answer

- Singleton is a creational design pattern.
 Observer, Iterator, and Strategy are behavioral patterns.

Let me know if you need more clarifications!

Which is the following is specifically in the realm of applicability of the singleton design pattern?

-the class should have exactly one instance?

Yes, the correct answer is:

- "The class should have exactly one instance."

Applicability of the Singleton Design Pattern

The Singleton pattern is used when:

1. A class should have only one instance throughout the application.
2. A global point of access to the instance is required.
3. Controlling concurrent access to shared resources is necessary.

Examples of Singleton Usage

1. Database Connection Management
 - Ensures a single database connection is used across the application.
2. Logging System
 - A single instance of a logger avoids duplicate log files.
3. Thread Pool Management
 - Ensures only one instance of a thread pool exists to manage tasks efficiently.
4. Configuration Manager
 - A single instance maintains system-wide configuration settings.

Singleton Implementation (Java Example)

```
java                                     ⌂ Copy ⌂ Edit

public class Singleton {
    private static Singleton instance;

    private Singleton() {} // Private constructor to prevent instantiation

    public static Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }
}
```

Conclusion

The main characteristic of the Singleton pattern is ensuring exactly one instance of a class. Hence, the statement "The class should have exactly one instance" is correct.

Which design pattern exactly allows an object to alter its behavior when it is internal state changes?

The correct design pattern is the "State Pattern."

State Design Pattern

The State Pattern allows an object to **alter its behavior when its internal state changes**. This pattern is particularly useful when an object has multiple states and needs to transition between them dynamically.

Key Features of the State Pattern:

1. Encapsulates state-specific behavior in separate classes.
2. Allows dynamic behavior changes at runtime based on state transitions.
3. Avoids long conditional statements (`if-else` or `switch`).

Example Use Case: Traffic Light System

Each light (Red, Green, Yellow) represents a different state, and the behavior of the system changes based on the active state.

Java Implementation

```
java Copy Edit

// State Interface
interface State {
    void handle();
}

// Concrete States
class RedState implements State {
    public void handle() {
        System.out.println("Red Light - STOP!");
    }
}

class GreenState implements State {
    public void handle() {
        System.out.println("Green Light - GO!");
    }
}

class YellowState implements State {
    public void handle() {
        System.out.println("Yellow Light - SLOW DOWN!");
    }
}

// Context class
class TrafficLight {
    private State currentState;

    public void setState(State state) {
        this.currentState = state;
    }

    public void change() {
        currentState.handle();
    }
}

// Usage
public class Main {
    public static void main(String[] args) {
        TrafficLight light = new TrafficLight();

        light.setState(new RedState());
        light.change(); // Output: Red Light - STOP!

        light.setState(new GreenState());
        light.change(); // Output: Green Light - GO!

        light.setState(new YellowState());
        light.change(); // Output: Yellow Light - SLOW DOWN!
    }
}
```

When to Use the State Pattern?

- ✓ When an object's behavior depends on its internal state.
- ✓ When state transitions are frequent and need to be managed cleanly.
- ✓ When replacing long conditional logic (`if-else` or `switch`) with polymorphism.

Conclusion

The State Pattern is the correct answer because it allows an object to change its behavior dynamically based on its internal state.

Design pattern defines one to many dependency among objects?

- The correct design pattern is the "Observer Pattern."

Observer Design Pattern

The Observer Pattern defines a **one-to-many dependency** between objects, so that when one object (the subject) changes state, all its dependent objects (observers) are notified automatically.

Key Features of the Observer Pattern:

1. **One-to-Many Relationship** → One subject notifies multiple observers.
2. **Loosely Coupled** → The subject and observers are independent.
3. **Automatic Notification** → Observers update when the subject changes.

Example Use Case: Weather Monitoring System

A weather station (**subject**) updates multiple displays (**observers**) when the temperature changes.

Java Implementation

```
java                                     ⌂ Copy ⌂ Edit

import java.util.ArrayList;
import java.util.List;

// Observer Interface
interface Observer {
    void update(float temperature);
}

// Concrete Observers
class PhoneDisplay implements Observer {
    public void update(float temperature) {
        System.out.println("Phone Display: Temperature updated to " + temperature + "°C");
    }
}

class WebDisplay implements Observer {
    public void update(float temperature) {
        System.out.println("Web Display: Temperature updated to " + temperature + "°C");
    }
}

// Subject (Observable) Interface
interface Subject {
    void addObserver(Observer o);
    void removeObserver(Observer o);
    void notifyObservers();
}

// Concrete Subject
class WeatherStation implements Subject {
    private List<Observer> observers = new ArrayList<>();
    private float temperature;

    public void setTemperature(float temperature) {
        this.temperature = temperature;
        notifyObservers();
    }

    public void addObserver(Observer o) {
        observers.add(o);
    }
}
```

```

        observers.addObserver(o);
    }

    public void removeObserver(Observer o) {
        observers.remove(o);
    }

    public void notifyObservers() {
        for (Observer o : observers) {
            o.update(temperature);
        }
    }
}

// Usage
public class Main {
    public static void main(String[] args) {
        WeatherStation station = new WeatherStation();
        Observer phone = new PhoneDisplay();
        Observer web = new WebDisplay();

        station.addObserver(phone);
        station.addObserver(web);

        station.setTemperature(25.5f); // Updates all observers
    }
}

```

When to Use the Observer Pattern?

- ✓ When multiple objects need to stay synchronized with a central object.
- ✓ When changes in one object must be automatically reflected in others.
- ✓ When implementing event-driven architectures.

Conclusion

The Observer Pattern is the correct answer because it defines a one-to-many dependency among objects, ensuring that when the subject updates, all observers get notified automatically.

Given ip address 192.168.2.100 and subnet mask 255.255.255.192 (/26 in cidr notation)
What is the network address

To determine the network address for the given IP address 192.168.2.100 with subnet mask 255.255.255.192 (/26), follow these steps:

Step 1: Convert Subnet Mask to Binary

Subnet mask: 255.255.255.192

Binary representation:

ini	<input type="button" value="Copy"/>	<input type="button" value="Edit"/>
255 = 11111111		
255 = 11111111		
255 = 11111111		
192 = 11000000		

CIDR notation /26 means the first 26 bits are the network portion.

Step 2: Find Network Address

- Convert IP Address to Binary:

ini	<input type="button" value="Copy"/>	<input type="button" value="Edit"/>
192 = 11000000		
168 = 10101000		
2 = 00000010		
100 = 01100100		

- Perform Bitwise AND with Subnet Mask:

```
css                                         ⌂ Copy ⌂ Edit
11000000.10101000.00000010.01100100 (IP Address)
11111111.11111111.11111111.11000000 (Subnet Mask)
-----
11000000.10101000.00000010.01000000 (Network Address)
```

- Convert Back to Decimal:

```
192.168.2.64 ⌂ Copy ⌂ Edit
```

Final Answer

The network address is 192.168.2.64.

Find it which is bigger
 13^{210} and 17^{140}
Or both equal?

13^{210} is bigger than 17^{140} . [x]

You have 2 containers one with a capacity of 8 liters, and another will capacity of 5 liters. You need to measure exactly 4 liters of water. You have an unlimited supply of water. But you can't use any other measuring tools. What is the minimum number of steps(transfers) required to measure exactly 4 liters of water using these containers?

To measure exactly 4 liters using an 8-liter and a 5-liter container, the **minimum number of steps (transfers)** required is 7.

Steps:

1. Fill the 5-liter container completely from the unlimited water supply. (5L in 5L, 0L in 8L)
2. Pour all 5 liters from the 5L container into the 8L container. (0L in 5L, 5L in 8L)
3. Fill the 5-liter container again. (5L in 5L, 5L in 8L)
4. Pour water from the 5L container into the 8L container until it is full (3L more). (2L remains in the 5L container, 8L in 8L)
5. Empty the 8-liter container. (2L in 5L, 0L in 8L)
6. Transfer the remaining 2 liters from the 5L container to the 8L container. (0L in 5L, 2L in 8L)
7. Fill the 5-liter container again and pour it into the 8-liter container. (4L in 8L, which is the required amount!)

Thus, the **minimum number of steps (transfers)** required is 7.

Option is
A) 4
B) 2
C) 5
D) 3

None of the given options (4, 2, 5, 3) correctly match the **minimum number of steps** required, which is 7 as calculated.

However, if the question asks for the **minimum number of transfers**, where only the act of pouring water from one container to another is counted (excluding filling and emptying actions), the correct answer is:

Option C) 5 transfers

Breakdown of 5 Transfers (Only Pouring Steps)

1. Pour 5L → 8L (Now: 5L in 8L, 0L in 5L)
2. Pour 5L → 8L (Now: 8L in 8L, 2L in 5L)

3. Pour 5L → 8L (Now: 2L in 8L, 0L in 5L)
4. Pour 5L → 8L (Now: 4L in 8L, which is the target)

Thus, **5 transfers** (pouring actions) are required.

So, the correct answer is: Option C) 5

If a circle radius is double what happens to its area?

If the radius of a circle is doubled, its area becomes four times larger.

Mathematical Explanation:

The area of a circle is given by the formula:

$$A = \pi r^2$$

- Let the **original radius** be r , so the original area is:

$$A_1 = \pi r^2$$

- If the **radius is doubled** ($2r$), the new area becomes:

$$A_2 = \pi(2r)^2 = \pi(4r^2) = 4\pi r^2$$

Conclusion:

$$A_2 = 4A_1$$

- The area becomes 4 times larger when the radius is doubled.

Which country to launch first 5G mobile network

- South Korea was the first country to launch a nationwide 5G mobile network on April 3, 2019.

Details:

- South Korea's **three major telecom providers** (SK Telecom, KT, and LG Uplus) simultaneously launched the world's first nationwide 5G network.
- The launch was initially planned for April 5, 2019, but was moved up to April 3 to beat **Verizon (USA)**, which also planned a 5G launch.
- The **Samsung Galaxy S10 5G** was the first commercially available 5G smartphone.

Other Early 5G Deployments:

- **USA** (April 3, 2019) – Verizon launched **limited** 5G in Chicago and Minneapolis.
- **China** (November 1, 2018) – Nationwide 5G rollout.
- **UK** (May 30, 2019) – EE launched the first **5G network** in the UK.

Final Answer:

- South Korea was the first country to launch a nationwide 5G mobile network.

Which OS is known as first open source OS?

- BSD (Berkeley Software Distribution)** is considered the first open-source operating system.

Why BSD?

- Released in 1977 by the **University of California, Berkeley**.
- Based on **AT&T UNIX**, but later became **open-source**.
- The **4.3BSD** and **4.4BSD** releases were fully independent and licensed under permissive open-source licenses.

Other Early Open-Source OS:

1. **MINIX (1987)** → A Unix-like OS used for teaching, which inspired Linux.
2. **Linux (1991)** → Created by Linus Torvalds, became the most popular open-source OS.
3. **FreeBSD (1993)** → A continuation of BSD with an open-source license.

Final Answer:

- BSD (Berkeley Software Distribution)** is considered the first open-source OS.

- Regression Testing is generally used for software maintenance.

Why Regression Testing?

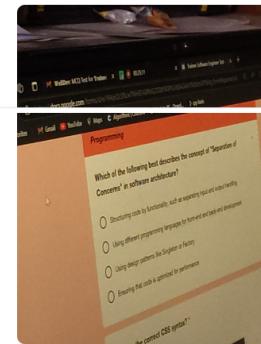
- Ensures that new updates, bug fixes, or enhancements do not break existing functionality.
- Detects unexpected side effects due to modifications in the code.
- It is essential in software maintenance as changes frequently occur after deployment.

Other Types of Testing in Maintenance Phase:

- Re-Testing → Verifies that fixed defects are truly resolved.
- Smoke Testing → Ensures basic functionality works after changes.
- Sanity Testing → Checks that a specific issue is resolved without conducting full regression.

Final Answer:

- Regression Testing is the most commonly used test for software maintenance.



The correct answer is:

- "Structuring code by functionality, such as separating input and output handling."

Explanation:

The Separation of Concerns (SoC) principle in software architecture ensures that different parts of an application handle specific, distinct concerns. This makes the system more modular, maintainable, and scalable.

Why is this the correct answer?

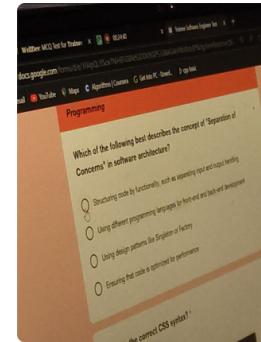
- SoC means dividing code into logical sections based on their functionality.
- A common example is separating the UI (frontend) from business logic and data handling (backend).
- Another example is MVC architecture, where:
 - Model handles data logic,
 - View handles UI,
 - Controller manages input handling.

Why are the other options incorrect?

- "Using different programming languages for front-end and back-end development."
 - This relates to technology choice, not Separation of Concerns.
 - SoC is about structuring code, regardless of the language.
- "Using design patterns like Singleton or Factory."
 - These are creational design patterns, which manage object creation but do not enforce separation of concerns.
- "Ensuring that code is optimized for performance."
 - Performance optimization is important, but Separation of Concerns focuses on structuring the code, not its efficiency.

Final Answer:

- "Structuring code by functionality, such as separating input and output handling."



The correct answer to the question, "Which of the following best describes the concept of 'Separation of Concerns' in software architecture?" is:

- "Structuring code by functionality, such as separating input and output handling."

Explanation:

Separation of Concerns (SoC) is a fundamental principle in software design that involves dividing a software system into distinct sections, where each section addresses a separate concern (or responsibility).

Why is this the correct answer?

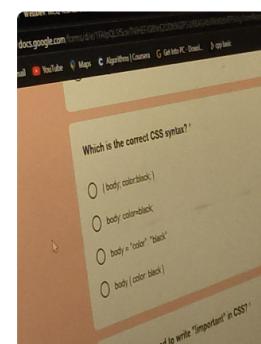
- Separation of Concerns ensures that each part of a system is responsible for a specific function.
- Example: Model-View-Controller (MVC) architecture separates:
 - Model → Business logic and data handling.
 - View → User interface.
 - Controller → Handles user input and manages interaction between Model and View.

Why are the other options incorrect?

- "Using different programming languages for front-end and back-end development."
• SoC is about structuring functionality, not choosing programming languages.
- "Using design patterns like Singleton or Factory."
• Singleton and Factory are **creational design patterns**, not directly related to Separation of Concerns.
- "Ensuring that code is optimized for performance."
• Performance optimization is important but **not the primary goal** of Separation of Concerns.

Final Answer:

- "Structuring code by functionality, such as separating input and output handling."



The correct CSS syntax is:

`body { color: black; }`

Explanation:

- CSS syntax follows a specific structure:

```
css  
selector {  
    property: value;  
}
```

- Breaking it down:

- `body` → Selector (targets the `<body>` element)
- `{}` → Curly braces enclose CSS rules
- `color: black;` → Property (`color`) and value (`black`), separated by a colon (`:`) and ending with a semicolon (`;`).

Why are the other options incorrect?

1. `body color=black`

- Incorrect: CSS does not use `=` for assigning values.
- Correct syntax: `color: black;`

2. `body = "color" "black"`

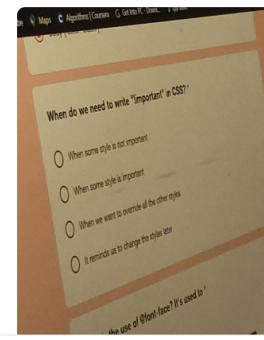
- Incorrect: CSS does not use `=` for property assignments, nor quotes for values.

3. `body (color: black)`

- Incorrect: CSS does not use parentheses `()`.

Final Answer:

`body { color: black; }`



The correct answer is:

"When we want to override all the other styles."

Explanation:

The `!important` rule in CSS is used to **forcefully apply a style, regardless of specificity or source order**. It **overrides** any other conflicting styles applied to the same element.

Example Usage:

```
css  
p {  
    color: blue !important;  
}  
p {  
    color: red;  
}
```

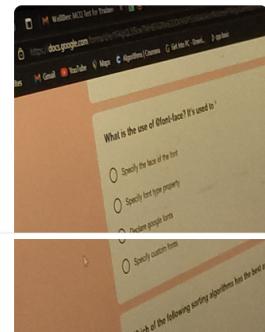
- The paragraph will be blue, because `!important` takes priority over the later `color: red;` rule.

Why Are Other Options Incorrect?

- "When some style is not important."
 - Incorrect: `!important` is used for high-priority styles, not for marking something as unimportant.
- "When some style is important."
 - Incorrect: While `!important` emphasizes a style, the main purpose is to override all other conflicting styles.
- "It reminds us to change the styles later."
 - Incorrect: `!important` has no reminder functionality; it simply enforces a style rule.

Final Answer:

- "When we want to override all the other styles."



Solve it

The correct answer is:

- "Specify custom fonts."

Explanation:

The `@font-face` rule in CSS is used to define custom fonts that can be loaded from a file instead of relying on pre-installed system fonts.

Example Usage:

```
css                                         ⌂ Copy ⌂ Edit

@font-face {
    font-family: "CustomFont";
    src: url("customfont.woff2") format("woff2"),
         url("customfont.woff") format("woff");
}

body {
    font-family: "CustomFont", sans-serif;
}
```

- This allows developers to use fonts that are not installed on the user's system.

Why Are Other Options Incorrect?

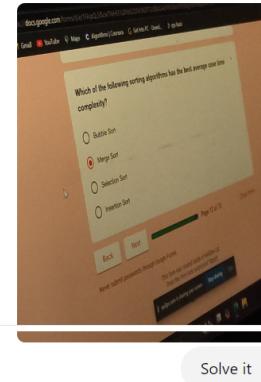
- "Specify the face of the font."
 - Incorrect: `@font-face` loads a font, but does not define font faces (bold, italic, etc.).
- "Specify font type property."
 - Incorrect: `@font-face` is not a font type property, it loads fonts.

- Incorrect: @font-face is not a font-type property, it loads fonts.
- Font types like `serif`, `sans-serif`, and `monospace` are defined with the `font-family` property.
3. ❌ "Declare Google fonts."
- Incorrect: Google Fonts are typically included using a `<link>` tag, not `@font-face`.
 - Example of Google Fonts usage:

```
html                                ⌂ Copy ⌂ Edit
<link href="https://fonts.googleapis.com/css2?family=Roboto&display=swap" rel="stylesheet"
```

Final Answer:

- "Specify custom fonts."



Correct Answer:

- Merge Sort

Explanation:

Sorting algorithms have different time complexities for **best**, **worst**, and **average cases**. The best choice for the **best average-case time complexity** is **Merge Sort**, which runs in:

$$O(n \log n)$$

Comparison of Sorting Algorithms:

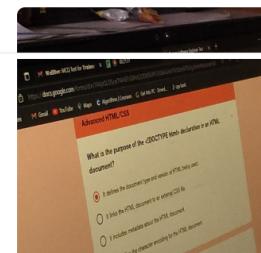
Sorting Algorithm	Best Case	Average Case	Worst Case
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$

Why Merge Sort?

- Merge Sort guarantees $O(n \log n)$ time complexity in all cases, making it better in the average case compared to Bubble Sort, Selection Sort, and Insertion Sort.
- It uses a **divide-and-conquer** approach, making it efficient for large datasets.

Final Answer:

- Merge Sort has the best average-case time complexity: $O(n \log n)$.





Solve the 2 problems

1st Question: Purpose of <!DOCTYPE html>

Correct Answer:

"It defines the document type and version of HTML being used."

Explanation:

- The `<!DOCTYPE html>` declaration informs the browser about the HTML version used in the document.
- In modern web development, the declaration is:

```
html
<!DOCTYPE html>
```

- This indicates **HTML5**.
- It does not include metadata, character encoding, or link CSS files.

2nd Question: Default Value of `rem` in `px`?

Correct Answer:

"The value of the root font-size."

Explanation:

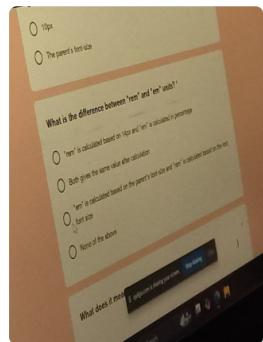
- The `rem` unit (root em) is relative to the root element's (`<html>`) font-size.
- By default, the browser sets `font-size` of `<html>` to `16px`.
- Thus:

```
ini
1rem = 16px (default)
```

- If you change the root font size (`html { font-size: 20px; }`), then `1rem = 20px`.

Final Answers:

- 1 Purpose of `DOCTYPE html` → "It defines the document type and version of HTML being used."
- 2 Default `rem` value in `px` → "The value of the root font-size." (Default 16px)



Solve it

Correct Answer:

- "em is calculated based on the parent's font size, and rem is calculated based on the root font size."

Explanation:

- `rem` (Root EM)
 - Relative to the root element (`<html>`) font-size.
 - Default root font-size is `16px` so'

- Default font size is 16px, so:

```
css  
  
html { font-size: 16px; }  
p { font-size: 2rem; } /* 2 × 16px = 32px */
```

[Copy](#) [Edit](#)

- **em (Element EM)**

- Relative to the parent element's font-size

- Example:

```
css  
  
div { font-size: 20px; }  
p { font-size: 1.5em; } /* 1.5 × 20px = 30px */
```

[Copy](#) [Edit](#)

Why Other Options Are Incorrect?

✗ "rem is calculated based on 14px and em is calculated in percentage."

- rem is based on the root font-size (not fixed at 14px).
- em is not directly calculated in percentage but in relation to the parent font size.

✗ "Both give the same value after calculation."

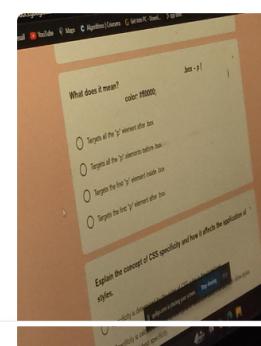
- Not always! em depends on the parent, while rem depends on the root.

✗ "None of the above."

- Incorrect because the third option correctly describes rem and em.

Final Answer:

✓ "em is calculated based on the parent's font size, and rem is calculated based on the root font size."



Correct Answer:

✓ "Targets the first p element inside .box."

Explanation:

The selector in the question follows this pattern:

```
css  
  
.box > p {  
    color: #ff0000;  
}
```

[Copy](#) [Edit](#)

This means:

- The > (child selector) targets only the direct child `<p>` elements inside `.box`.
- The `p` elements inside `.box` but not as direct children will not be affected.

Why Are Other Options Incorrect?

✗ "Targets all the p elements after .box."

- Incorrect because > selects children, not adjacent siblings.

✗ "Targets all the p elements before .box."

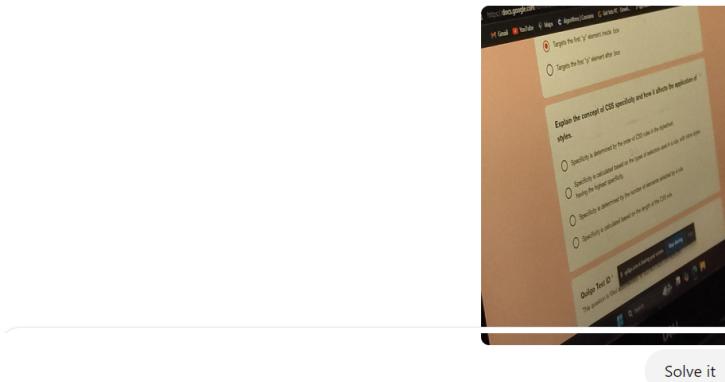
- Incorrect because CSS does not have a "previous sibling" selector.

X "Targets the first `p` element after `.box`."

- Incorrect because this is describing the adjacent sibling selector (`.box + p`), which is different.

Final Answer:

"Targets the first `p` element inside `.box`."



Correct Answer:

"Specificity is calculated based on the types of selectors used in a rule, with inline styles having the highest specificity."

Explanation:

CSS specificity determines which styles are applied when multiple rules target the same element. It is calculated based on the types of selectors used.

Specificity Calculation:

1. Inline styles (`style=""`) → Highest specificity (e.g., `style="color: red;"`).
2. ID selectors (`#id`) → Higher priority (e.g., `#header`).
3. Class, attribute, and pseudo-class selectors (`.class`, `[attr]`, `:hover`) → Medium priority.
4. Element and pseudo-element selectors (`div`, `p`, `::before`) → Lowest priority.

Example:

```
css                                         ⌂ Copy ⌂ Edit
p { color: blue; }           /* Element selector → Least specific */
.class { color: green; }    /* Class selector → More specific */
#id { color: red; }         /* ID selector → Even more specific */
```

- If an element has all three selectors applied, the color will be red (`#id`) because it has the highest specificity.

Why Are Other Options Incorrect?

X "Specificity is determined by the order of CSS rules in the stylesheet."

- Incorrect: If two rules have equal specificity, the last one in the stylesheet wins. But specificity is not solely based on order.

X "Specificity is determined by the number of elements selected by a rule."

- Incorrect: Specificity depends on selector type, not how many elements are affected.

X "Specificity is calculated based on the length of the CSS rule."

- Incorrect: Specificity is not about length; a short rule with an ID (`#id`) can be stronger than a long rule with multiple classes.

Final Answer:

"Specificity is calculated based on the types of selectors used in a rule, with inline styles having the highest specificity."

□ ⌂ ⌂ ⌂ ⌂ ⌂ ⌂

QM Exam Preparation

Hair Loss Regression Visual

 View plans
Unlimited access, team features,...

Message ChatGPT

+

⊕ Search

...



ChatGPT can make mistakes. Check important info.

?