FLOBIZ

Q1. HTML Input Type — Password Field

Q: In an HTML form, which input type should be used for passwords to prevent visible text while typing?

```
html
CopyEdit
<input type="_____" id="password" name="password">
```

A. text
B. password
C. email
D. hidden

✅ **Answer: B. password**
ℹ It hides the characters as the user types.

Question 2:

Marks
20.00/60
Total Time
12 min 0 sec
Time Taken
12 min 0 sec

As a frontend developer for a popular e-commerce website, you are tasked with designing a responsive product gallery page that showcases a list of products in a grid layout. The page should be optimized for both desktop and mobile devices. The product gallery page should have the following layout:

- The header section should take up the full width of the page and have a height of 100 pixels.
- The product list section should be divided into three columns on desktop devices and two columns on mobile devices.

- Each product item should have a width of 30% on desktop devices and 45% on mobile devices.
- The product list section should have a 20-pixel gap between each column.
- The footer section should take up the full width of the page and have a height of 50 pixels.
  Write a CSS code snippet that achieves the above layout using Flexbox and Grid Layouts.

  You can use the following HTML structure as a starting point:

```
<header>Header Section</header>
<div class="product-list">
  <div class="product-item">Product 1</div>
  <div class="product-item">Product 2</div>
  <div class="product-item">Product 3</div>
  <!-- Add more product items here -->
</div>
<footer>Footer Section</footer>
```

```
/* Reset and basic styling */
* {
  box-sizing: border-box;
  margin: 0;
  padding: 0;
}

body {
  font-family: sans-serif;
}

/* Header */
header {
  height: 100px;
  background-color: #f5f5f5;
  width: 100%;
  display: flex;
  align-items: center;
  justify-content: center;
}

/* Footer */
footer {
  height: 50px;
  background-color: #ccc;
  width: 100%;
  display: flex;
  align-items: center;
  justify-content: center;
}

/* Product List using Grid */
.product-list {
  display: grid;
  grid-template-columns: repeat(3, 30%);
  gap: 20px;
  justify-content: center;
  padding: 20px;
}

/* Product Item */
.product-item {
  background-color: #eaeaea;
  padding: 20px;
  text-align: center;
  border-radius: 8px;
```

```
}


/* Responsive Design for Mobile */

@media (max-width: 768px) {

  .product-list {

    grid-template-columns: repeat(2, 45%);

  }

}
```

uestion 3:

Marks

0.00/20

Total Time

4 min 0 sec

Time Taken

2 min 41 sec

E-commerce website "ShopEasy" is facing an issue with its product page. The product description is not displaying properly when the user clicks on the "More Info" button. The development team has identified that the issue is due to incorrect DOM manipulation. The product description is stored in a HTML element with an ID "product-description" and the "More Info" button has an ID "more-info-button". When the button is clicked, the JavaScript function `displayProductDescription()` is triggered. The function is supposed to get the product description element, remove any existing child nodes, and then append a new paragraph element with the product description text. However, the function is not working as expected.
Write the correct implementation of the `displayProductDescription()` function to fix the issue.

```
// Assume the product description text is stored in a variable
'descriptionText'
function displayProductDescription() {
  // TO DO: implement the correct DOM manipulation
}
```

The implementation should ensure that the product description is displayed correctly when the "More Info" button is clicked.

```
// HTML structure
<div id="product-description">
  <!-- product description will be displayed here -->
</div>
<button id="more-info-button">More Info</button>
```

What is the correct implementation of the `displayProductDescription()` function?

```
•  function displayProductDescription() {
•    var descElement = document.getElementById("product-description");
•    descElement.innerText = descriptionText;
•  }
•
•  function displayProductDescription() {
•    var descElement = document.getElementById("product-description");
•    descElement.innerHTML = "<p>" + descriptionText + "</p>";
•  }
•
•  function displayProductDescription() {
•    var descElement = document.getElementById("product-description");
•    var paraElement = document.createElement("p");
•    paraElement.textContent = descriptionText;
•    descElement.appendChild(paraElement);
•  }
•
•  function displayProductDescription() {
•    var descElement = document.getElementById("product-description");
•    descElement.textContent = descriptionText;
•  }
```

The **correct implementation** of the `displayProductDescription()` function—based on the requirement to:

- **Get** the element with ID `product-description`,
- **Remove all existing child nodes**, and then
- **Append a new `<p>` element** containing the `descriptionText`,

is the following:

```
javascript
CopyEdit
function displayProductDescription() {
  var descElement = document.getElementById("product-description");
  // Remove all existing child nodes
  while (descElement.firstChild) {
    descElement.removeChild(descElement.firstChild);
  }

  // Create and append new paragraph
  var paraElement = document.createElement("p");
  paraElement.textContent = descriptionText;
  descElement.appendChild(paraElement);
}
```

Option:2

Question 4:

Marks

0.00/20

Total Time

4 min 0 sec

Time Taken

2 min 39 sec

A social media company, "ConnectMe", wants to develop a feature to allow users to like and comment on posts in real-time. The company has designed a simple web page with a post section, like button, and a comment input field. When a user clicks the like button, it should update the like count in real-time without reloading the page. The company wants to use JavaScript to achieve this functionality. The HTML code for the post section is given below:

```
<div id="post-section">
  <p id="post-text">This is a sample post.</p>
  <button id="like-btn">Like</button>
  <span id="like-count">(0)</span>
  <input id="comment-input" type="text" placeholder="Add a comment">
  <button id="comment-btn">Comment</button>
</div>
```

Write a JavaScript code to achieve the desired functionality when the user clicks the like button.

What should be the event handler attached to the like button to update the like count in real-time?

- addEventListener('click', updateLikeCount)
- attachEvent('onclick', updateLikeCount)
- addEventListener('mouseover', updateLikeCount)
- addEventListener('dblclick', updateLikeCount)

Option 1:

Question 5:

Marks

20.00/90

Total Time

20 min 0 sec

Time Taken

15 min 16 sec

A social media platform, "ConnectMe", wants to implement a feature that allows users to upload multiple images simultaneously. The platform has a restriction that it can only process a maximum of 3 image uploads at a time. The development team has decided to use asynchronous JavaScript to implement this feature. The team wants to ensure that the image uploads are done in a controlled parallel manner to avoid overwhelming the server.

You are required to write a JavaScript function `uploadImages` that takes an array of image objects as input. Each image object contains a `url` property and a `size` property. The function should upload the images in parallel, but not more than 3 at a time. Once all the images are uploaded, the function should return a promise that resolves with an array of booleans indicating the success or failure of each image upload.
For example, if the input array is `[img1, img2, img3, img4, img5]`, the function should upload `img1`, `img2`, and `img3` simultaneously, and then wait for their results before uploading `img4` and `img5` simultaneously.

You can assume that the `uploadImage` function, which uploads a single image, is already implemented and available to you. This function returns a promise that resolves with a boolean indicating the success or failure of the upload.
Implement the `uploadImages` function using promises and/or async/await to achieve the desired behavior.

```
// Example image object
let img1 = { url: 'https://example.com/img1.jpg', size: 1024 };
let img2 = { url: 'https://example.com/img2.jpg', size: 2048 };
let img3 = { url: 'https://example.com/img3.jpg', size: 512 };
let img4 = { url: 'https://example.com/img4.jpg', size: 768 };
let img5 = { url: 'https://example.com/img5.jpg', size: 1536 };

// Example usage
let images = [img1, img2, img3, img4, img5];
uploadImages(images).then((results) => {
  console.log(results); // [true, true, true, true, true]
});
```

Async JS — Limit Concurrent Uploads to 3

```
js
CopyEdit
async function uploadImages(images) {
  let results = [];

  for (let i = 0; i < images.length; i += 3) {
    const chunk = images.slice(i, i + 3);
    const uploads = chunk.map(img => uploadImage(img));
    const chunkResults = await Promise.all(uploads);
    results.push(...chunkResults);
  }

  return results;
}
```

✅ Uses async/await + chunking to limit parallel uploads to 3.

Question 6:

Marks

0.00/20

Total Time

4 min 0 sec

Time Taken

3 min 8 sec

The finance department of a company is developing a web application to help employees track their expenses. The lead developer decides to use JavaScript functions to handle actions like adding, updating, and deleting expense entries. Sanjay, a junior developer on the team, is responsible for creating the function that calculates the total expenses from an array of expense objects. Each object contains properties such as `amount` and `category`. When he uses a function named `calculateTotal`, he notices it's returning undefined instead of the expected total amount. After reviewing his code, he realizes that he might be having issues with variable scope because he attempted to reference variables defined outside the function without properly passing them in. What would be the correct way for Sanjay to ensure that his calculateTotal function correctly accesses and sums up these amounts? Below is part of his proposed code: `javascript let expenses = [{amount: 50}, {amount: 20}, {amount: 30}]; function calculateTotal() { // Structure here }` Given this context, which option correctly defines how Sanjay should implement this function?

- function calculateTotal(expenses) { return expenses.reduce((total, expense) => total + expense.amount, 0); }
- function calculateTotal() { let total = 0; for (let i = 0; i < expenses.length; i++) { total += expenses[i].amount; } return total; }
- function calculateTotal() { return sum(amounts); }
- function calculateTotal(expenses) { var total = []; for (var i in expenses)total += i.amount;} return total;

Answer is Option 1: Option 1 (Correct)

```
javascript
CopyEdit
function calculateTotal(expenses) {
  return expenses.reduce((total, expense) => total + expense.amount, 0);
}
```

- This uses `.reduce()` correctly.

- It passes the `expenses` array as a parameter.
- It starts with an initial `total` of `0`, and adds each `expense.amount`.
- Returns `100` correctly for the given input.

❌ Option 2

```
javascript
CopyEdit
function calculateTotal() {
  let total = 0;
  for (let i = 0; i < expenses.length; i++) {
    total += expenses[i].amount;
  }
  return total;
}
```

- This can work **only if** `expenses` is globally available.
- But the question mentions **scope issues**, so **passing `expenses` as a parameter** is better.

❌ Option 3

```
javascript
CopyEdit
function calculateTotal() {
  return sum(amounts);
}
```

- There is no `sum()` or `amounts` defined.
- This will throw a **ReferenceError**.

❌ Option 4

```
javascript
CopyEdit
function calculateTotal(expenses) {
  var total = [];
  for (var i in expenses) total += i.amount;
  return total;
}
```

- Wrong logic:
    - `i` is an index, not the expense object.
    - `i.amount` is undefined.
    - `total` is wrongly initialized as an array.
- This returns incorrect value or throws error.

✅ Final Answer:

```
javascript
CopyEdit
function calculateTotal(expenses) {
  return expenses.reduce((total, expense) => total + expense.amount, 0);
}
```

Question 7:

Marks

0.00/120

Total Time

20 min 0 sec

Time Taken

7 min 37 sec

Trapping Rainwater Problem

Given `n` non-negative integers representing an elevation map where the width of each bar is `1`. Compute how much water it can trap after rain.

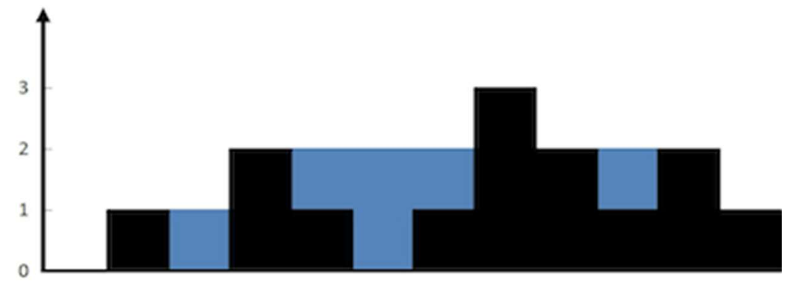
Input Format:

The first line contains one integer input `n`, the size of the array.
The second line contains n space-separated integers (arr[i]) that describe the width of each bar.

Output Format:

Prints a single integer value, which represents the amount of water it can hold.

Example 1:



Input:
```
12
0 1 0 2 1 0 1 3 2 1 2 1
```
Output:
```
6
```
Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.

Example 2:

Input:
```
6
4 2 0 3 2 5
```
Output:
```
9
```
Explanation: In this case, 9 units of rainwater will be trapped.

Constraints:

n == height.length

1 <= n <= 2 * 10^4

0 <= arr[i] <= 10^5

Response:

Answr:

#include <iostream>

```
#include <vector>
using namespace std;

int trappingRainWater(vector<int>& height) {
    int n = height.size();
    int left = 0, right = n - 1;
    int left_max = 0, right_max = 0;
    int water = 0;

    while (left < right) {
        if (height[left] < height[right]) {
            if (height[left] >= left_max) {
                left_max = height[left];
            } else {
                water += left_max - height[left];
            }
            left++;
        } else {
            if (height[right] >= right_max) {
                right_max = height[right];
            } else {
```

```
            water += right_max - height[right];
        }
        right--;
      }
   }

   return water;
}

int main() {
   int n;
   cin >> n;

   vector<int> height(n);
   for (int i = 0; i < n; ++i) {
      cin >> height[i];
   }

   cout << trappingRainWater(height) << endl;
   return 0;
}
```

E-Commerce Platform Optimization woith oop

ChatGPT said:

Creating an **E-Commerce Platform Optimization** system using **Object-Oriented Programming (OOP) in C++** involves organizing your system using **classes**, **encapsulation**, and **modularity**.

🛒💡 What We'll Implement:

A **basic e-commerce structure** with:

1. `Product` class – to hold product details
2. `Cart` class – to manage cart operations
3. `User` class – to simulate a user and place orders
4. **Optimization** – via reusable design (OOP), search by price or name

---

✅ C++ OOP Implementation

```
cpp
CopyEdit
#include <iostream>
#include <vector>
#include <string>
using namespace std;

class Product {
private:
    int id;
    string name;
    double price;

public:
    Product(int pid, string pname, double pprice) {
        id = pid;
        name = pname;
        price = pprice;
    }

    int getId() const { return id; }
    string getName() const { return name; }
    double getPrice() const { return price; }

    void display() const {
        cout << "ID: " << id << " | Name: " << name << " | Price: ₹" <<
price << endl;
    }
};

class Cart {
private:
    vector<Product> items;
```

```
public:
    void addToCart(Product p) {
        items.push_back(p);
        cout << "✔ Product '" << p.getName() << "' added to cart.\n";
    }

    void removeFromCart(int productId) {
        for (auto it = items.begin(); it != items.end(); ++it) {
            if (it->getId() == productId) {
                cout << "🗑 Removed: " << it->getName() << "\n";
                items.erase(it);
                return;
            }
        }
        cout << "❌ Product not found in cart.\n";
    }

    void viewCart() {
        cout << "\n🛒 Cart Items:\n";
        double total = 0;
        for (const auto& item : items) {
            item.display();
            total += item.getPrice();
        }
        cout << "💰 Total: ₹" << total << "\n";
    }

    bool isEmpty() {
        return items.empty();
    }

    void clearCart() {
        items.clear();
        cout << "□ Cart cleared.\n";
    }
};

class User {
private:
    string name;
    Cart cart;

public:
    User(string uname) {
        name = uname;
    }

    void addProduct(Product p) {
        cart.addToCart(p);
    }

    void removeProduct(int pid) {
        cart.removeFromCart(pid);
    }

    void viewMyCart() {
        cart.viewCart();
    }
    void checkout() {
        if (cart.isEmpty()) {
            cout << "🛑 Cart is empty. Cannot checkout.\n";
        } else {
            cout << "✅ Checkout successful. Thank you, " << name << "!\n";
            cart.clearCart();
        }
    }
};

void displayProducts(const vector<Product>& catalog) {
    cout << "\n📦 Available Products:\n";
    for (const auto& p : catalog) {
        p.display();
    }
}

Product searchProductById(const vector<Product>& catalog, int id) {
    for (const auto& p : catalog) {
        if (p.getId() == id)
            return p;
    }
    throw runtime_error("Product not found!");
}

int main() {
    vector<Product> catalog = {
        Product(1, "Phone", 15000),
        Product(2, "Laptop", 60000),
        Product(3, "Headphones", 2000),
        Product(4, "Smartwatch", 5000)
    };

    User user("Sanath");

    int choice, pid;

    while (true) {
        cout << "\n--- 🛍️ E-Commerce Menu ---\n";
        cout << "1. Show Products\n2. Add to Cart\n3. Remove from Cart\n4.
View Cart\n5. Checkout\n6. Exit\n";
        cout << "Enter choice: ";
        cin >> choice;

        try {
            switch (choice) {
                case 1:
                    displayProducts(catalog);
                    break;
                case 2:
                    cout << "Enter Product ID to add: ";
                    cin >> pid;
                    user.addProduct(searchProductById(catalog, pid));
                    break;
                case 3:
                    cout << "Enter Product ID to remove: ";
                    cin >> pid;
                    user.removeProduct(pid);
                    break;
                case 4:
                    user.viewMyCart();
                    break;
                case 5:
                    user.checkout();
                    break;
                case 6:
                    cout << "🚪 Exiting... Thank you!\n";
                    return 0;
                default:
                    cout << "⚠️ Invalid choice!\n";
            }
        } catch (exception& e) {
            cout << "❌ Error: " << e.what() << endl;
        }
    }

    return 0;
}
```

🔧 Features You Can Add for Optimization:

Feature	Benefit
Price filtering	Faster product selection
Product sorting	Better user experience
Stock count in `Product`	Helps limit inventory issues
Wishlist/Favorites	User personalization
Admin panel class	Manage product list
File I/O	Store catalog and cart data persistently

⚙️ OOP Concepts Used:

- ✅ **Classes and Objects** (`Product`, `Cart`, `User`)
- ✅ **Encapsulation** (private members with getters)
- ✅ **Composition** (`User` contains a `Cart`)
- ✅ **Exception Handling** (`searchProductById`)
- ✅ **Reusability** (modular functions and classes)