Domingo, Jhana Christina L.

**BSIT 32E2**

1. (10 points) Write a C# program that calculates the area of a triangle given its base and height. Include user input for both values and display the calculated area.

Source Code :

using System;

class Program

{

static void Main()

{

Console.WriteLine("Enter the base of the triangle:");

double baseLength = Convert.ToDouble(Console.ReadLine());

Console.WriteLine("Enter the height of the triangle:");

double height = Convert.ToDouble(Console.ReadLine());

double area = 0.5 \* baseLength \* height;

Console.WriteLine($"The area of the triangle is: {area}");

}

}

**OUTPUT :**

A screenshot of a computer

Description automatically generated

1. (10 points) Declare an array of 5 integers and fill it with values based on a user-defined formula (e.g., n^2). Then, print the largest element in the array.

**Source Code :**

using System;

class Program

{

static void Main()

{

int[] numbers = new int[5];

for (int i = 0; i < numbers.Length; i++)

{

numbers[i] = (i + 1) \* (i + 1); // Using the formula n^2

Console.WriteLine("Array elements:");

foreach (var number in numbers)

{

Console.WriteLine(number);

}

int max = numbers[0];

for (int i = 1; i < numbers.Length; i++)

{

if (numbers[i] > max)

{

max = numbers[i];

}

}

Console.WriteLine($"The largest element in the array is: {max}");

}

}

**OUTPUT :**

A screenshot of a computer

Description automatically generated

1. (10 points) Implement a simple for loop that iterates from 1 to 10 and prints each number along with its square root.

**Source Code :**

using System;

class Program

{

static void Main()

{

for (int i = 1; i <= 10; i++)

{

double squareRoot = Math.Sqrt(i);

Console.WriteLine($"Number: {i}, Square Root: {squareRoot}");

}

}

}

**Output :**

A screenshot of a computer program

Description automatically generated

**Part 2. HTML, CSS, and JavaScript (30 points)**

1. HTML (10 points): You are provided with the following incomplete HTML code snippet:

HTML

<!DOCTYPE html>

<html>

<head>

<title>My Website</title>

</head>

<body>

<h1>Welcome to...</h1>

<p>This is a paragraph...</p>

<ul>

<li>Item 1</li>

<li>Item 2</li>

</ul>

</body>

</html>

Complete the code snippet by adding the following elements:

* An image within the <body> tag with a relevant src attribute.

<!DOCTYPE html>

<html>

<head>

<title>Image Example</title>

</head>

<body>

<h1>Image Example</h1>

<p>Here is an image:</p>

<img src="image.jpg" alt="Description of the image">

</body>

</html>

* An ordered list (<ol>) with three items.

<ol>

<li>First item</li>

<li>Second item</li>

<li>Third item</li>

</ol>

* A hyperlink within a <p> tag that points to an external website.

<p>Visit <a href="https://www.example.com">Example</a> website.</p>

* A CSS styling rule using an inline style attribute to change the font color of the <h3> heading.

<!DOCTYPE html>

<html>

<head>

<title>Inline CSS Example</title>

</head>

<body>

<h3 style="color: red;">This is a heading with red font color</h3>

</body>

</html>

1. CSS (10 points): Create a CSS stylesheet that defines the following styles:

* Change the background color of the body element to light blue.

. body {

background-color: lightblue;

}

* Apply a padding of 20px to all headings (h1, h2, h3).

h1, h2, h3 {

padding: 20px;

}

* Set the font size of the <p> tag to 14px.

. p {

font-size: 14px;

}

* Make the list items (li) have a bullet point style instead of the default numbers.

ul {

list-style-type: disc; /\* Use disc for filled bullets, circle for hollow bullets \*/

}

1. JavaScript (10 points): Write a JavaScript function that takes a number as input and returns a string indicating whether the number is even or odd. Then, add a button to your HTML page that, when clicked, calls this function and displays the result (even or odd) in a paragraph element below the button.

HTML CODE

<!DOCTYPE html>

<html>

<head>

<title>Even/Odd Checker</title>

</head>

<body>

<button onclick="checkNumber()">Check Number</button>

<p id="result"></p>

<script src="script.js"></script>

</body>

</html>

JAVA

function isEvenOrOdd(num) {

return num % 2 === 0 ? 'even' : 'odd';

}

function checkNumber() {

const number = parseInt(prompt('Enter a number:'));

const result = document.getElementById('result');

if (isNaN(number)) {

result.textContent = 'Invalid input. Please enter a valid number.';

return;

}

result.textContent = `The number ${number} is ${isEvenOrOdd(number)}.`;

}

**Part 3**

1. **Part 3: Essay Question (40 points)**

Discuss the importance of object-oriented programming (OOP) concepts in software development. Explain the key principles of OOP (encapsulation, inheritance, polymorphism, abstraction) and provide examples of how they can be used to create more efficient, maintainable, and reusable code. Include real-world scenarios or cases where OOP is particularly valuable.

* Discuss the importance of object-oriented programming (OOP) concepts in software development.
* Object-oriented programming (OOP) is a programming paradigm that uses "objects" to design applications and computer programs. These objects can contain data in the form of fields (attributes or properties) and code in the form of procedures (methods or functions). Object-oriented programming (OOP) enables the modular design of software, enhancing organization, understanding, and maintenance by representing each object as a separate module. It also facilitates code reuse across different parts or programs, increases security through encapsulation of data and methods within objects, supports hierarchical relationships and code reuse through inheritance, and promotes flexibility and maintainability by allowing a single interface to represent various object types.
* Explain the key principles of OOP (encapsulation, inheritance, polymorphism, abstraction)
* Modularity: OOP allows for the modular design of software, where each object represents a separate module. This makes the code more organized, easier to understand, and easier to maintain.
* Reusability: Objects can be reused in different parts of a program or in different programs altogether, which can save time and effort in development.
* Encapsulation: OOP encapsulates data (attributes) and methods (functions) into objects, which restricts access to the inner workings of an object from outside code. This improves security and reduces the likelihood of bugs.
* Inheritance: OOP allows for the creation of new classes (objects) that inherit attributes and methods from existing classes. This promotes code reuse and allows for the creation of hierarchical relationships between classes.
* Polymorphism: OOP allows for the use of a single interface to represent different types of objects. This can make code more flexible and easier to maintain, as it can be written to work with a generic type rather than specific types.
* Examples of how they can be used to create more efficient, maintainable, and reusable code. Include real-world scenarios or cases where OOP is particularly valuable.
* Modularity

Example 3: Data Access Layer

Problem: You need to access data from a database in your application.

Modular Solution: Create a separate Data Access Layer module that handles all interactions with the database, such as querying and updating data.

Benefits:

Database access code is isolated and can be easily reused in other applications.

Changes to the database schema or data access logic can be made independently.

Example 4: Game Development

Problem: You are developing a game and need to manage different game objects and their interactions.

Modular Solution: Use an entity-component-system (ECS) architecture where each game object is composed of independent components (e.g., graphics, physics, AI) that can be added, removed, and updated independently.

Benefits:

Flexible and extensible architecture that allows for easy addition of new game objects and behaviors.

Components can be reused across different types of game objects.

In each of these examples, modularity allows for the creation of more efficient, maintainable, and reusable code by breaking down complex systems into smaller, manageable components that can be developed and tested independently.

* Inheritance

Example : Code Reuse - Inheritance allows you to define common behavior in a superclass and then reuse that code in multiple subclasses. For example, you could have a Vehicle superclass with properties and methods common to all vehicles, such as start engine() and stop engine(), and then create subclasses like Car, Truck, and Motorcycle that inherit these methods.

Efficiency: By using inheritance, you can avoid duplicating code in multiple classes. This makes your code more efficient and easier to maintain, as you only need to make changes in one place (the superclass) if the shared behavior needs to be modified.

Maintainability: Inheritance can help improve the maintainability of your code by organizing related classes into a hierarchy. This makes it easier to understand the structure of your code and makes it easier to make changes or add new functionality in the future.

* Encapsulation

Encapsulated data: Data is hidden and can only be accessed through defined methods, reducing the risk of accidental modifications.

Encapsulated methods: Methods that operate on the data are bundled with the data, making it easier to understand and maintain the code.

Code reusability: Encapsulation allows objects to be reused in different parts of the program or in different programs altogether, as the internal implementation details are hidden.

This approach enhances code maintainability by limiting the impact of changes to the internal structure of the object.

Shape hierarchy: Suppose you have a hierarchy of shapes such as Circle, Square, and Triangle, all inheriting from a common Shape class. Each shape class would implement a calculate Area() method. With polymorphism, you can treat all these shapes uniformly, for example, by iterating over a list of shapes and calling calculate Area() on each one without needing to know the specific type of each shape.

* Polymorphism

Example :

Sorting algorithms: Polymorphism can be used to implement sorting algorithms that can sort various types of data structures, such as arrays of integers, strings, or custom objects. By defining a common interface for comparison (e.g., a Comparator interface), sorting algorithms can work with any type of data as long as it implements that interface.

Graphic user interface (GUI) controls: In GUI programming, you might have different types of UI controls such as buttons, checkboxes, and text fields, all inheriting from a common Control class. Each control class might have methods like render() or handleClick(). With polymorphism, you can treat all these controls uniformly, allowing you to write generic code that can work with any type of UI control.