**Mance, Rizzalyn**

**BSIT 32E1**

**Sample Assessment for Introduction to Programming**

This assessment is designed to evaluate your understanding of basic programming concepts in C#, HTML, CSS, and JavaScript.

**Part 1: C# (30 points)**

**(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.**

using System;

namespace TriangleAreaCalculator

{

    class Calculator

    {

        static void Main(string[] args)

        {

            double baseLength, height, area;

            //get baseLength

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

            if (!double.TryParse(Console.ReadLine(), out baseLength) || baseLength <= 0)

            {

                Console.WriteLine("Please enter a valid POSITIVE number for the base length.");

                return;

            }

            // get height

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

            if (!double.TryParse(Console.ReadLine(), out height) || height <= 0)

            {

                Console.WriteLine("Please enter a valid POSITIVE number for the height.");

                return;

            }

            // Calculate area

            area = 0.5 \* baseLength \* height;

            Console.WriteLine($"Area of the triangle w/ base {baseLength} and height {height} is: {area}");

            Console.ReadKey();

        }

    }

}

**(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.**

using System;

namespace LargestElementInArray

{

    class Compare

    {

        static void Main(string[] args)

        {

            int[] Arr1 = { 87, 45, 42, 21, 56 };

            Console.WriteLine("Numbers within the Array: ");

            foreach (int num in Arr1)

            {

                Console.WriteLine(num);

            }

            int bigger = Arr1[0];

            for (int i = 1; i < Arr1.Length; i++)

            {

                if (Arr1[i] > bigger)

                {

                    bigger = Arr1[i];

                }

            }

            Console.WriteLine("The largest element in the array is: " + bigger);

            Console.ReadKey();

        }

    }

}

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

using System;

namespace IterationLoop

{

    class Loop

    {

        static void Main(string[] args)

        {

            int num = 1;

            double sqr;

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

            {

                sqr = Math.Sqrt(num);

                Console.WriteLine(num + " " + sqr);

                num = num + 1;

            }

        }

    }

}

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

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

**<!DOCTYPE html>**

**<html>**

**<head>**

**<title>My Website</title>**

**<style>**

**body {**

**background-color: #ae9c8f;**

**}**

**h1, h2, h3 {**

**padding: 20px;**

**}**

**p {**

**font-size: 14px;**

**}**

**li {**

**list-style-type: disc;**

**}**

**</style>**

**</head>**

**<body>**

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

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

**<img src="coffeebooth..png">**

**<ol>**

**<li>Coffee</li>**

**<li>Non-Coffee</li>**

**<li>Frappe</li>**

**</ol>**

**<p>Visit for a<a href="https://www.boscoffee.com/"> Coffee Shop</a></p>**

**<h3 style="color: #ffffff;">The best way to start your day!! </h3>**

**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.

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

  <p id="result"></p>

  <script>

    function checkNumber() {

      var number = prompt("Enter number:");

      if (isNaN(number)) {

        document.getElementById("result").textContent = "Invalid input. Please enter a valid number.";

        return;

      }

      number = Number(number);

      if (number % 2 === 0) {

        document.getElementById("result").textContent = number + " is even.";

      } else {

        document.getElementById("result").textContent = number + " is odd.";

      }

    }

  </script>

</body>

</html>

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

Object-oriented programming (OOP) is integral to modern software development due to its ability to enhance code organization, reusability, and maintainability. The key principles of OOP—encapsulation, inheritance, polymorphism, and abstraction—play a crucial role in achieving these benefits.

1. **Encapsulation:** Encapsulation bundles data and methods into a single unit, the class. This protects the data from being accessed directly and ensures that the operations on the data are done through well-defined methods. For example, consider a class **Car** with private attributes like **speed** and **fuel**. Encapsulation ensures that these attributes can only be accessed and modified through methods like **accelerate** and **refuel**, maintaining the integrity of the **Car** object.
2. **Inheritance:** Inheritance allows a new class to inherit properties and behaviors from an existing class, promoting code reuse and reducing redundancy. For instance, in a software application for a school, a base class **Person** can be inherited by **Student** and **Teacher** classes. This allows both **Student** and **Teacher** to inherit common attributes and methods such as **name** and **age** from **Person**, avoiding the need to redefine them in each subclass.
3. **Polymorphism:** Polymorphism allows objects to be treated as instances of their parent class, enabling flexibility and extensibility in code. An example of polymorphism is a **Shape** class with subclasses like **Circle** and **Square**. Each subclass can have its own implementation of a method called **calculateArea()**, which is defined in the parent class **Shape**. This allows the same method **calculateArea()** to behave differently depending on the type of shape.
4. **Abstraction:** Abstraction focuses on hiding the complex implementation details and showing only the necessary features of an object. This simplifies the interface for interacting with objects. For example, consider a class **BankAccount** with methods like **deposit()** and **withdraw()**. The internal details of these methods, such as updating balances and transaction logs, are abstracted away from the user, who only needs to know how to use these methods.