Introduction to programming C#

Overview

The following activities are designed to be done in parallel with the course of 10975 Introduction to programming. This course is a starting point to the world of programming. The main objective of this course is to learn the basics of Microsoft Visual Studio and C#. In this course, it is assumed that the students have no prior experience and introduce the concepts needed to progress to the intermediate course on programming. With this in mind, the main objective of these challenges for the students is to start developing the mindset of a developer and to find solution to programming challenges by themselves.

It's very important to notice that all challenges are connected and intended to be done sequentially. The first challenge is needed to complete the second, and the second is needed to complete the third. In the end, the code should be able to calculate the area, perimeter to given geometric figures. The first challenge's focus is to get started, consolidate the learning from the module one to five.

With that said, the students will be asked to verify, test and most importantly, write some code to advance in the challenges. But if needed, base files will be provided with errors and missing parts for the students.

Requirements

- Ability to use computers to start programs, open and save files, navigate application menus and interfaces.
- Ability to understand logical concepts such as comparisons.
- Understand number theory.
- Ability to create, understand, and follow structured directions or step-by-step procedures.
- Ability to understand and apply abstract concepts to concrete examples.
- Visual Studio and C# installed.

General Objectives

Challenge 1:

Lab setup:

45-60 mins

Introduction:

This challenge is designed to start thinking about the logic of programming. Creating an App in a procedural form is key to start thinking about how a computer program is written and interact with external components such as keyboard, files, etc. And within the code itself, such as functions and etc…

Objectives:

In this exercise, students will write functions that calculate specific outputs and create a simple logic that will help to select the right function according to the user inputs. Of course, the base code contains several syntax errors that the student should correct.

- Create functions.
- Use of If-else, or other logical/conditional expression.
- Use of repetition expression.
- Error handling and code compilation.

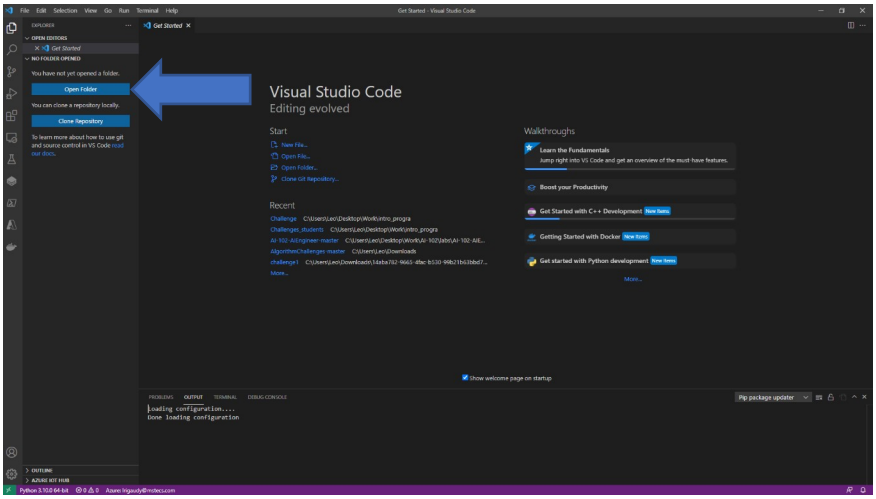
Exercise:

In this challenge, the goal for the student is to verify and complete the code. In this first challenge, the code is supposed to calculate the Area of three different geometric figures. Triangle, circles, and rectangles. The user interacts with the program via the console and based on the figures the user wishes to calculate, the program will ask for the needed variables as it follows:

- Triangle: 3 sides needed.
- Rectangle (or square): 2 sides needed.
- Circle: only radius is needed.

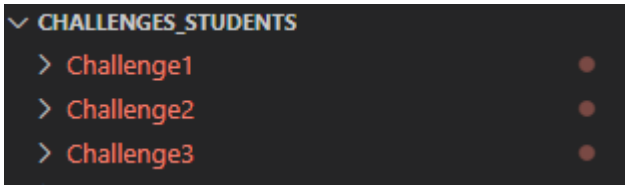
Open labs files on “Visual Studio Code”

1. Open “Visual Studio Code”.
2. Click on “Open folder”

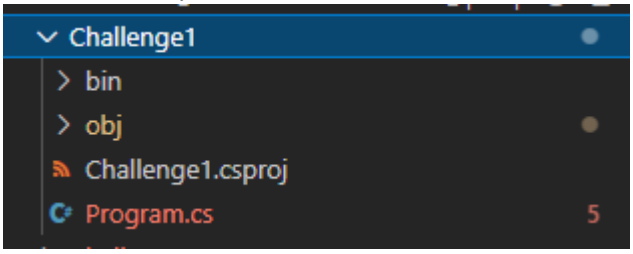


3. Select the folder provided. (You should select the general folder, not the specific folder of each exercise.)

4. Now, you should see this on the left side of your screen when you click on “explorer” (ctrl+shift+E).



5. Now that we open the folder, this I what we should see inside of them.

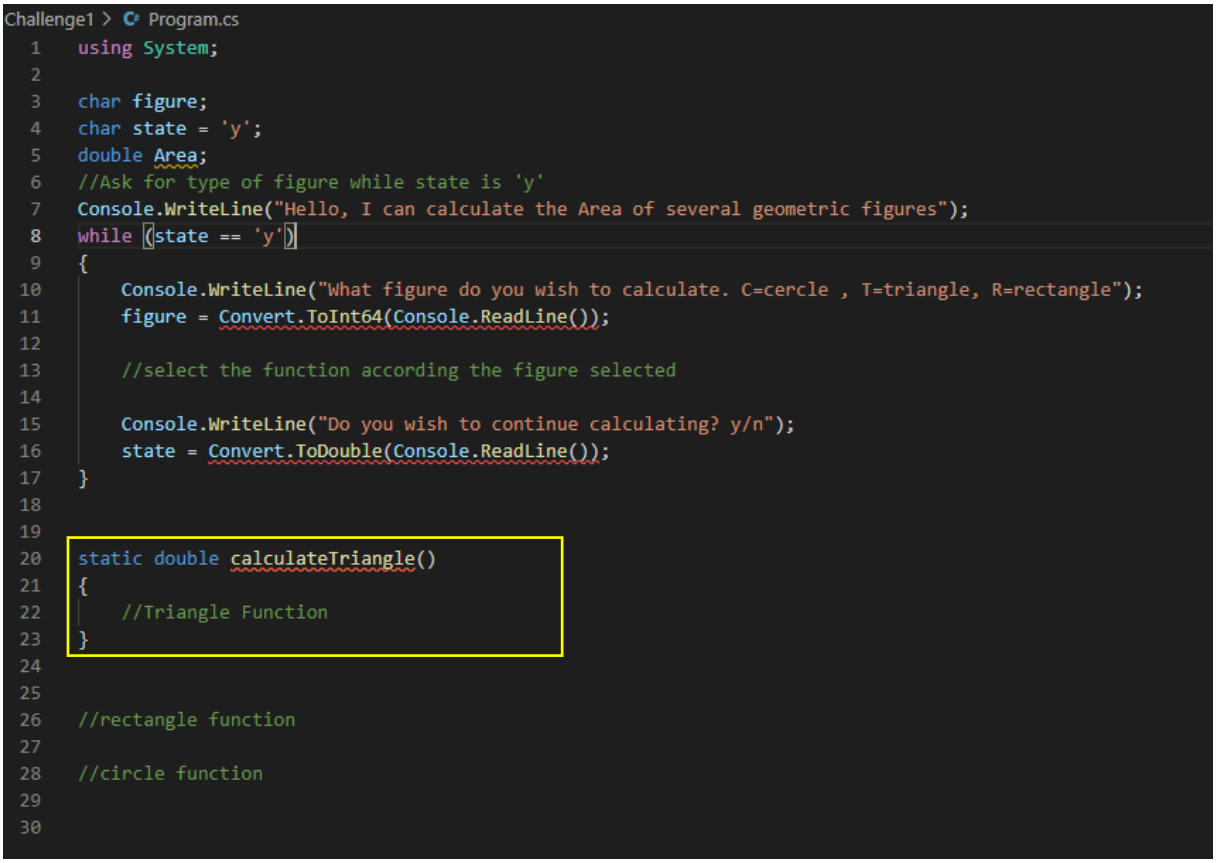


The only file that matters to us is the “Program.cs”. This file is where we are going to write our code.

To open it, simply click on it and the code should open on your visual studio code.

Code writing on Visual Studio Code

1. Now, we have several things to look at to run our code. It’s clear that several parts are missing, and some others are incorrect. So, first, let’s create the functions that are going to calculate the area of each figure. The triangle function is already created but nothing is inside yet. So, let’s fill it.



2. Since the code we are writing need 3 sides in order to calculate the area of a triangle (based on Heron's formula). Let's ask for them to the user. With the following two lines of code and repeat it for the other two sides.

```
Console.WriteLine("Side a");
a = Convert.ToDouble(Console.ReadLine());
```

In the first line, we output "side a" to the user, so he can know we are asking for the side a. And the second line assign to "a" the input from the user on the console.

3. Now that we have our sides, we have to implement the formula to calculate the area. Based on Heron's formula. We need:

$s=\frac{a+b+c}{2}$ and $Area=\sqrt{s(s-a)(s-b)(s-c)}$

The code should look like that:

```
//Heron's formula
s = Convert.ToDouble((a+b+c)/2);
result = Convert.ToDouble(Math.Sqrt(s*(s-a)*(s-b)*(s-c)));
```

4. Now, we have our "CalculateTriangle" function almost ready, We only have to return the value of the calculated area and help the user that he is inside the triangle function. To do that, we'll output to the console the message "Triangle selected" and "please, provide me the variables needed" (this part is not mandatory, but it will help the user and us if the code doesn't run as intended.) You're function should look more or less like that:

```
//Triangle Function
static double calculateTriangle()
{
      double  a, b, c, s, result;
      Console.WriteLine("Triangle selected");
      Console.WriteLine("Please provide me the variables needed");
      //ask for sides
      Console.WriteLine("Side a");
      a = Convert.ToDouble(Console.ReadLine());
      Console.WriteLine("Side b");
      b = Convert.ToDouble(Console.ReadLine());
      Console.WriteLine("Side c");
      c = Convert.ToDouble(Console.ReadLine());

      //Heron's formula
      s = Convert.ToDouble((a+b+c)/2);
      result = Convert.ToDouble(Math.Sqrt(s*(s-a)*(s-b)*(s-c)));
      return result;
}
```

Perfect! Now we have our Triangle function. Now let's create the other two needed, circle and square (or rectangle, it's basically the same).

5. Now, let's get started with the circle function. We only need the radius to calculate the area of a circle. So, let's do the same process as the triangle. But, for only one variable, radius. We create a function called calculateCircle like that:

```
static double calculateCercle()
{
      //code Inside the function
}
```

6. Perfect! Now, let's ask for the radius!

```
Console.WriteLine("Cercle selected");
Console.WriteLine("Please provide me the variables needed");
//ask for radius
Console.WriteLine("What is your radius?");
radius = Convert.ToDouble(Console.ReadLine());
```

7. And the last part is to calculate it and return it!

```
//results = Pi*R^2
result = Math.PI*(Math.Pow(radius,2));
return result;
```

8. In the end, your function should look like this:

```
static double calculateCercle()
{
    double result, radius;
    Console.WriteLine("Cercle selected");
    Console.WriteLine("Please provide me the variables needed");
    //ask for radius
    Console.WriteLine("What is your radius?");
    radius = Convert.ToDouble(Console.ReadLine());

    //results = Pi*R^2
    result = Math.PI*(Math.Pow(radius,2));
    return result;
}
```

Now we have two functions ready, we are missing one function yet, the square one. Let's get to it!

9. Remember, we create the function, then we populate it with the logic.

```
static double calculateRectangle()
{
      //code inside the function
}
```

10. Now, we ask for what we need:

```
Console.WriteLine("Rectangle selected");
Console.WriteLine("Please provide me the variables needed");
//ask for both sides needed
Console.WriteLine("What is your 1st side?");
side1 = Convert.ToDouble(Console.ReadLine());
```

```
Console.WriteLine("What is your 2nd side?");
side2 = Convert.ToDouble(Console.ReadLine());
```

11. And the last part, we calculate and return the Area:

```
result = side1*side2;
return result;
```

Well done! Now that we have our functions ready to be used. We need to call them accordingly to the user's needs. To do that, we will populate with code the "main" part of the code.

12. Our base code already has a while loop created for us. If we look at the code, the "while" loop is basically running while the state variable is equal to "y". But if wee look closely, after asking to the user if he wishes to continue, we assign the value entered by the user to the variable "state". But this line of code is wrong. Do you know what's wrong? Let's review it together:

```
Console.WriteLine("Do you wish to continue calculating? y/n");
state = Convert.ToDouble(Console.ReadLine());
```

Found it? If yes congratulation, If no, don't worry.

We are converting the state value to double. But we are expecting a letter. So we should change the conversion to "Conver.ToChar(.....)". And your code should look like that:

```
Console.WriteLine("Do you wish to continue calculating? y/n");
state = Convert.ToChar(Console.ReadLine());
```

13. The last part we need to write, is how we are going to select the right function depending on the user needs. There are several ways to do that. But here we will use the "If" and "else" statements.

First, let's ask to the user, what figure he wishes to calculate:

```
Console.WriteLine("What figure do you wish to calculate. C=cercle ,
T=triangle, R=rectangle");
figure = Convert.ToChar(Console.ReadLine());
```

We assign the user input inside the "figure" variable in Char.

And now we select according to the letter and of course, we call our functions and the output the result in the same time:

```
if (figure == 'C'|| figure == 'c')
{
    Area = calculateCercle();
    Console.WriteLine("The area is equal to {0}", Area);
}
else if (figure == 'R' || figure == 'r')
{
    Area = calculateRectangle();
    Console.WriteLine("The area is equal to {0}", Area);
}
else
{
    Area = calculateTriangle();
    Console.WriteLine("The area is equal to {0}", Area);
```

```
}
```

Your main code should look like this:

```
while (state == 'y')
{
    Console.WriteLine("What figure do you wish to calculate. C=circle ,
T=triangle, R=rectangle");
    figure = Convert.ToChar(Console.ReadLine());

    //select the function according the figure selected
    if (figure == 'C'|| figure == 'c')
    {
        Area = calculateCercle();
        Console.WriteLine("The area is equal to {0}", Area);
    }
    else if (figure == 'R' || figure == 'r')
    {
        Area = calculateRectangle();
        Console.WriteLine("The area is equal to {0}", Area);
    }
    else
    {
        Area = calculateTriangle();
        Console.WriteLine("The area is equal to {0}", Area);
    }

    Console.WriteLine("Do you wish to continue calculating? y/n");
    state = Convert.ToChar(Console.ReadLine());
}
```

14. If you followed well this exercise, you should have notice that we are missing one important part. How do we declare our variables?

For the main area, we used three variables. “figure”, “state”, and “Area”. “figure” and “state” are char variables since we store inside them a single letter. As for “Area” it should be at least int, but since we are calculating Areas, and most of times, the area is not a integer number. So we declare it as a double.

```
char figure;

char state = 'y';

double Area;
```

15. And for the functions. We should create and declare the variable we are going to use inside each function. So, for the triangle, we have the variables declared as double for the result, the “s” and each side of the triangle(a,b,c).

```
double  a, b, c, s, result;
```

16. Same for the square, we declare:

```
double side1, side2, result;
```

17. And of course, the circle:

```
double result, radius;
```

Well done! You've completed this exercise. Your code should look like this:

```
using System;

char figure;
char state = 'y';
double Area;
//Ask for type of figure while state is 'y'
Console.WriteLine("Hello, I can calculate the Area of several geometric
figures");
while (state == 'y')
{
    Console.WriteLine("What figure do you wish to calculate. C=cercle ,
T=triangle, R=rectangle");
    figure = Convert.ToChar(Console.ReadLine());

    //select the function according the figure selected
    if (figure == 'C'|| figure == 'c')
    {
        Area = calculateCercle();
        Console.WriteLine("The area is equal to {0}", Area);
    }
    else if (figure == 'R' || figure == 'r')
    {
        Area = calculateRectangle();
        Console.WriteLine("The area is equal to {0}", Area);
    }
    else
    {
        Area = calculateTriangle();
        Console.WriteLine("The area is equal to {0}", Area);
    }

    Console.WriteLine("Do you wish to continue calculating? y/n");
    state = Convert.ToChar(Console.ReadLine());
}

//Triangle Function
static double calculateTriangle()
{
    double  a, b, c, s, result;
    Console.WriteLine("Triangle selected");
    Console.WriteLine("Please provide me the variables needed");
```

```
    //ask for sides
    Console.WriteLine("Side a");
    a = Convert.ToDouble(Console.ReadLine());
    Console.WriteLine("Side b");
    b = Convert.ToDouble(Console.ReadLine());
    Console.WriteLine("Side c");
    c = Convert.ToDouble(Console.ReadLine());

    //Heron's formula
    s = Convert.ToDouble((a+b+c)/2);
    result = Convert.ToDouble(Math.Sqrt(s*(s-a)*(s-b)*(s-c)));
    return result;
}

//Cercle function
static double calculateCercle()
{
    double result, radius;
    Console.WriteLine("Cercle selected");
    Console.WriteLine("Please provide me the variables needed");
    //ask for radius
    Console.WriteLine("What is your radius?");
    radius = Convert.ToDouble(Console.ReadLine());

    //results = Pi*R^2
    result = Math.PI*(Math.Pow(radius,2));
    return result;
}

//rectangle function
static double calculateRectangle()
{
    double side1, side2, result;
    Console.WriteLine("Rectangle selected");
    Console.WriteLine("Please provide me the variables needed");
    //ask for both sides needed
    Console.WriteLine("What is your 1st side?");
    side1 = Convert.ToDouble(Console.ReadLine());
    Console.WriteLine("What is your 2nd side?");
    side2 = Convert.ToDouble(Console.ReadLine());

    //result = a* b
    result = side1*side2;
    return result;
}
```

And Output similar to this:

```
Hello, I can calculate the Area of several geometric figures
What figure do you wish to calculate. C=cercle , T=triangle, R=rectangle
c
Cercle selected
Please provide me the variables needed
What is your radius?
12
The area is equal to 452,3893421169302
Do you wish to continue calculating? y/n
y
What figure do you wish to calculate. C=cercle , T=triangle, R=rectangle
t
Triangle selected
Please provide me the variables needed
Side a
1
Side b
1
Side c
1
The area is equal to 0,4330127018922193
Do you wish to continue calculating? y/n
y
What figure do you wish to calculate. C=cercle , T=triangle, R=rectangle
r
Rectangle selected
Please provide me the variables needed
What is your 1st side?
2
What is your 2nd side?
3
The area is equal to 6
Do you wish to continue calculating? y/n
n
```

Challenge 2:

lab setup:

lab time

Introduction

This challenge is designed to start thinking about the logic of programming an app in Object Oriented Programming. The modules 6 and 7 focus on an introduction of OOP and teaches students about inheritance, polymorphism in classes and function overloading.

Objectives:

The students have to adapt the previous code to OOP. Students will have to create classes based on an abstract class. Based on those classes, the student will implement the methods needed to solve the problem and implement a basic logic in the “main” part.

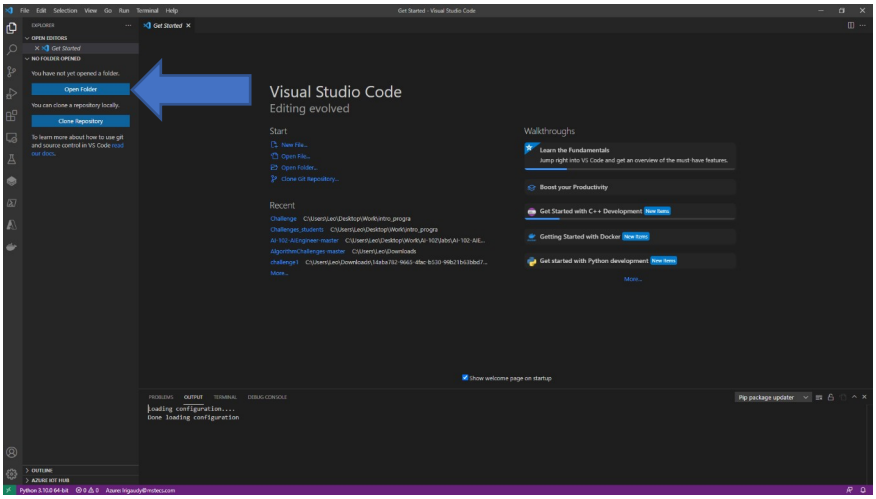
- Create and use basic class files
- Introduction to Inheritance
- Introduction to Polymorphism

Exercise:

In this challenge, the main goal is to adapt our previous code from a procedural form to OOP. The code will not do more. It just needs to be adapted to OOP. In order to do that, the students have to use an abstract class called “Shape”. And then create the classes needed with all the methods needed to make run the code the same way as the previous exercise. The program will calculate the area of three geometric figures based on their sides. The program should ask the user which of the three figures he wishes to calculate and then ask the variables needed.

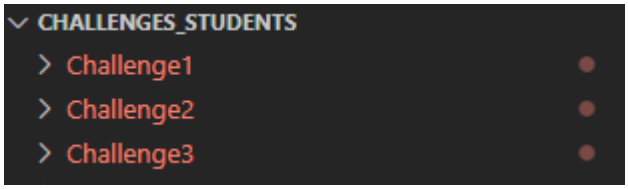
Open labs files on “Visual Studio Code”

1. Open “Visual Studio Code”.
2. Click on “Open folder”

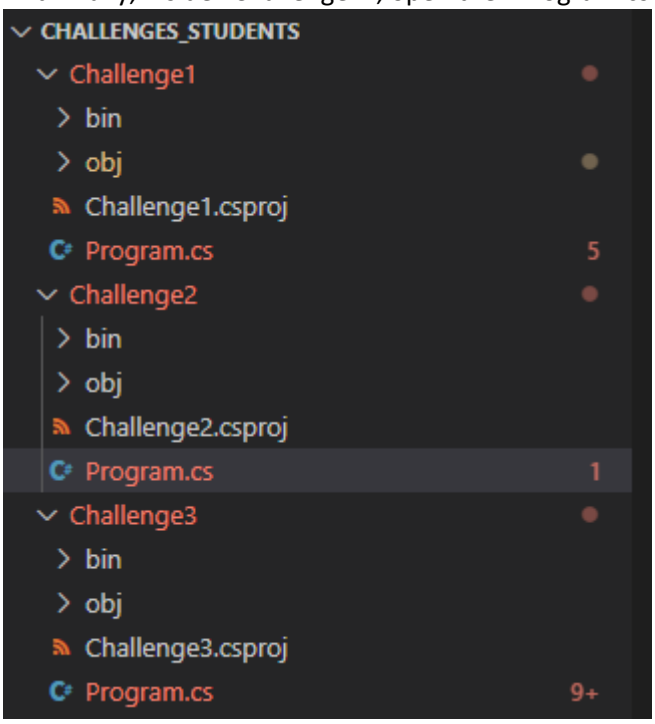


3. Select the folder provided. (You should select the general folder, not the specific folder of each exercise.)

4. Now, you should see this on the left side of your screen when you click on “explorer” (ctrl+shift+E).



5. And finally, inside “Challenge2”, open the “Program.cs” file to open the C# code file.



Code writing on Visual Studio Code

1. Perfect, now let's get starting. Our program needs to do the same as the previous one, but the logic inside the code will be very different. We will create or review first the abstract class. This class will help us when we'll create classes for each figure. On your file, the class should already be created and look like this:

```
abstract class Shape
{
    public abstract double CalculateArea();

}
```

Remember: abstract classes are created by using the 'abstract' keyword.

2. After that, we should create classes for each type of figure we wish to calculate. So, first, let's start to review the square class to understand and repeat easily the same process for the other two classes.

Our Square class should look like that:

```
class Square : Shape
{
    private double side1;
    private double side2;
    public Square() { }
    public Square(double side1, int side2)
    {
        this.side1 = side1;
        this.side2 = side2;
    }
    public override double CalculateArea()
    {
        int area = this.side1 * this.side2;
        return area;
    }
}
```

If you remember, we create a class by writing:

```
class Square : Shape
{
      //Code here
}
```

And the first thing we need to do is to declare the variables inside the class "Square". In order to calculate the area of a square or rectangle, we need at least two sides. So we declare two private variables in doubles called "side1" and "side2".

After that we specify two things, the constructors. One empty and one overloading wich we will use to pass in the two sides we need.

And finally, we inherit the from the base "Shape" class the method "CalculateArea()". And of course we populate the method with the right process to calculate the area of a square or rectangle so we can return the result.

Remember: any functions inherited from an abstract or virtual class MUST contain the 'override' keyword.

Perfect! Now that we've just reviewed the class "square". Let's create the other two.

3. Let's go with the triangle class which is a bit harder since we need three sides and to implement the heron's formula. We should create the class the same way as the square:

```
class Square : Shape
{
      //Code here
}
```

4. After that if we remember. We need to declare our variables inside the class:

```
private double tside1;
private double tside2;
private double tside3;
```

5. And now we need the constructor. Both overloaded and empty:

```
public Triangle() { }
public Triangle(double tside1, double tside2, double tside3)
{
       this.tside1 = tside1;
       this.tside2 = tside2;
       this.tside3 = tside3;
}
```

6. And finally, the inherited method from the abstract class:

```
public override double CalculateArea()
{
      double s = Convert.ToDouble((this.tside1+this.tside2+this.tside3)/2);
      double area = Convert.ToDouble(Math.Sqrt(s*(s-this.tside1)*(s-
this.tside2)*(s-this.tside3)));
      return area;
}
```

7. In the end, your Triangle class should look like that:

```
class Triangle : Shape
    {
    private double tside1;
    private double tside2;
    private double tside3;
    public Triangle() { }
    public Triangle(double tside1, double tside2, double tside3)
    {
        this.tside1 = tside1;
        this.tside2 = tside2;
        this.tside3 = tside3;
```

```
    }
    public override double CalculateArea()
    {
        double s = Convert.ToDouble((this.tside1+this.tside2+this.tside3)/2);
        double area = Convert.ToDouble(Math.Sqrt(s*(s-this.tside1)*(s-
this.tside2)*(s-this.tside3)));
        return area;
    }
}
```

Well done! We have now two classes ready to be used. But we need the last one, the circle one. So, let's get to it.

8. Same process here, but a bit shorter since we only need one variable, the radius. We create the class:

```
class Circle : Shape
{
      //code here
}
```

9. Then we create the variable needed as it follows:

```
private double radius;
```

10. After that, the constructors. Both empty and overloaded as well:

```
public Circle() { }
public Circle(double r)
{
      this.radius = r;
}
```

11. And finally, the method inherited:

```
public override double CalculateArea()
{
      double area = (double)(Math.Pow(this.radius, 2) * Math.PI);
      return area;
}
```

12. In the end. Your "circle" class should look similar to this:

```
class Circle : Shape
{
      private double radius;
      public Circle() { }
      public Circle(double r)
      {
            this.radius = r;
      }
      public override double CalculateArea()
      {
            double area = (double)(Math.Pow(this.radius, 2) * Math.PI);
            return area;
      }
```

```
}
```

Perfect! We have now all our classes needed! If we remember, now we need to create a logic main code to handle the creation of objects and methods we want to invoke.

13. First, we need to create a class “App” or “Program”. This is where the main is going to run. The “App” class should be specified as it follows:

```
class App
{
      //code here
}
```

14. We’ve created the Class “App”. Now wee need to specify the main area of our code. And we do it like this:

```
class App
{
      static void Main()
      {
            //main code here
      }
}
```

15. The logic of how we are going to select a figure to calculate is going to be the same since the goal here was to create, use classes and use their methods. So, we are going to have a while loop where we’re going to ask to the user which figure he wishes to calculate and then we will ask for the variables needed according to the figure selected. This time, the variables will be asked directly in the main code because we will create figures by overloading the constructor created in each class.

```
char figure;
char state = 'y';
Console.WriteLine("Hello, I can calculate the Area of several geometric
figures");
while (state == 'y')
{
      //code to select and construct the figures
}
```

16. The same process is applied to ask the user which figure he wishes to calculate. We aske the question to the user and wait for a Caracter in the terminal.

```
Console.WriteLine("What figure do you wish to calculate. C=cercle ,
T=triangle, R=rectangle");
figure = Convert.ToChar(Console.ReadLine());
```

17. Then, some simple “if else” will help us select the right constructors and ask the right number of variables. Let’s first create the first if:

```
if (figure == 'C'|| figure == 'c')
            {
                double radius;
                Console.WriteLine("Cercle selected");
                Console.WriteLine("Please provide me the variables needed");
                //We ask for radius
                Console.WriteLine("What is your radius?");
                radius = Convert.ToDouble(Console.ReadLine());
            }
```

18. And now, let’s populate it with the right constructor:

```
if (figure == 'C'|| figure == 'c')
{
      double radius;
      Console.WriteLine("Cercle selected");
      Console.WriteLine("Please provide me the variables needed");
      //We ask for radius
      Console.WriteLine("What is your radius?");
      radius = Convert.ToDouble(Console.ReadLine());

      Shape circle = new Circle(radius);
      Console.WriteLine("The area is equal to {0}", circle.CalculateArea());
}
```

19. Perfect, now we need and “else if” statement to select the next figure:

```
else if (figure == 'R' || figure == 'r')
{
      //code here
}
```

20. And let’s populate it also, but this time with the Rectangle(or square) variables assignations and constructor:

```
else if (figure == 'R' || figure == 'r')
{
      double side1, side2;
      Console.WriteLine("Rectangle selected");
      Console.WriteLine("Please provide me the variables needed");
      //We ask for both sides needed
      Console.WriteLine("What is your 1st side?");
      side1 = Convert.ToDouble(Console.ReadLine());
      Console.WriteLine("What is your 2nd side?");
      side2 = Convert.ToDouble(Console.ReadLine());
      Shape square = new Square(side1,side2);
      Console.WriteLine("The area is equal to {0}", square.CalculateArea());
}
```

21. And for the last part, just an “else” will do the job. A good practice to make your code more robust to error handling would be to use another “if else” with the specific character. But since this is a practice code, a simple “else” statement will do the job.

```
else
{
      //code here
}
```

22. Same here, we populate it with our variable’s assignations and the correct constructors:

```
else
{
      double  a, b, c;
      Console.WriteLine("Triangle selected");
      Console.WriteLine("Please provide me the variables needed");
      //We ask for 3 sides
      Console.WriteLine("Side a");
      a = Convert.ToDouble(Console.ReadLine());
      Console.WriteLine("Side b");
      b = Convert.ToDouble(Console.ReadLine());
      Console.WriteLine("Side c");
      c = Convert.ToDouble(Console.ReadLine());
      Shape triangle = new Triangle(a,b,c);
      Console.WriteLine("The area is equal to {0}", triangle.CalculateArea());
}
```

Well done! You’ve just completed to write your code in OOP. The code will do the same as the previous one. But this time you used a completely different approach. Your final code should more or less look like this:

```
using System;
abstract class Shape//this abstract Shape class is our base class to inherit
from
    {//abstract classes are created by using the 'abstract' keyword
        public abstract double CalculateArea();
       /*abstract classes may or may not have abstract  functions/fields, in
this case we'll make an abstract function called CalculateArea.*/
    }
//This is a subclass called Square with its own unique characteristics, it
inherits from Shape
    class Square : Shape
    {
        private double side1;
        private double side2;
        public Square() { }//empty constructor
        public Square(double side1, double side2)//overloading, we will pass
in a length & width (we'll use this to populate with data)
        {
            this.side1 = side1;
            this.side2 = side2;
        }
```

```
        public override double CalculateArea()//pay close attention here: any
functions inherited from an abstract or virtual class MUST contain the
'override' keyword
        {
            double area = this.side1 * this.side2;
            return area;            //and must return the same type
        }
    }
//the same is done for the next classes that inherit from Shape
    class Triangle : Shape
    {
        private double tside1;
        private double tside2;
        private double tside3;
        public Triangle() { }
        public Triangle(double tside1, double tside2, double tside3)
        {
            this.tside1 = tside1;
            this.tside2 = tside2;
            this.tside3 = tside3;
        }
        public override double CalculateArea()
        {
            double s =
Convert.ToDouble((this.tside1+this.tside2+this.tside3)/2);
            double area = Convert.ToDouble(Math.Sqrt(s*(s-this.tside1)*(s-
this.tside2)*(s-this.tside3)));
            return area;
        }
    }
//Same here, we inherit from 'Shape'
    class Circle : Shape
    {
        private double radius;
        public Circle() { }
        public Circle(double r)
        {
            this.radius = r;
        }
        public override double CalculateArea()
        {
            double area = (double)(Math.Pow(this.radius, 2) * Math.PI);
            return area;
        }
    }
class App
{
     static void Main()
    {
```

```
        char figure;
        char state = 'y';
        Console.WriteLine("Hello, I can calculate the Area of several
geometric figures");
        while (state == 'y')
        {
            Console.WriteLine("What figure do you wish to calculate.
C=cercle , T=triangle, R=rectangle");
            figure = Convert.ToChar(Console.ReadLine());

            //We select the right constructor according the figure selected
            if (figure == 'C'|| figure == 'c')
            {
                double radius;
                Console.WriteLine("Cercle selected");
                Console.WriteLine("Please provide me the variables needed");
                //We ask for radius
                Console.WriteLine("What is your radius?");
                radius = Convert.ToDouble(Console.ReadLine());
                Shape circle = new Circle(radius);
                Console.WriteLine("The area is equal to {0}",
circle.CalculateArea());
            }
            else if (figure == 'R' || figure == 'r')
            {
                double side1, side2;
                Console.WriteLine("Rectangle selected");
                Console.WriteLine("Please provide me the variables needed");
                //We ask for both sides needed
                Console.WriteLine("What is your 1st side?");
                side1 = Convert.ToDouble(Console.ReadLine());
                Console.WriteLine("What is your 2nd side?");
                side2 = Convert.ToDouble(Console.ReadLine());
                Shape square = new Square(side1,side2);
                Console.WriteLine("The area is equal to {0}",
square.CalculateArea());
            }
            else
            {
                double  a, b, c;
                Console.WriteLine("Triangle selected");
                Console.WriteLine("Please provide me the variables needed");
                //We ask for 3 sides
                Console.WriteLine("Side a");
                a = Convert.ToDouble(Console.ReadLine());
                Console.WriteLine("Side b");
                b = Convert.ToDouble(Console.ReadLine());
                Console.WriteLine("Side c");
                c = Convert.ToDouble(Console.ReadLine());
```

```
                Shape triangle = new Triangle(a,b,c);
                Console.WriteLine("The area is equal to {0}",
triangle.CalculateArea());
            }
            Console.WriteLine("Do you wish to continue calculating? y/n");
            state = Convert.ToChar(Console.ReadLine());
        }
    }
}
```

And the output should look like that:

```
Hello, I can calculate the Area of several geometric figures
What figure do you wish to calculate. C=cercle , T=triangle, R=rectangle
c
Cercle selected
Please provide me the variables needed
What is your radius?
12
The area is equal to 452,3893421169302
Do you wish to continue calculating? y/n
y
What figure do you wish to calculate. C=cercle , T=triangle, R=rectangle
t
Triangle selected
Please provide me the variables needed
Side a
1
Side b
1
Side c
1
The area is equal to 0,4330127018922193
Do you wish to continue calculating? y/n
y
What figure do you wish to calculate. C=cercle , T=triangle, R=rectangle
r
Rectangle selected
Please provide me the variables needed
What is your 1st side?
1
What is your 2nd side?
2
The area is equal to 2
Do you wish to continue calculating? y/n
n
```

Challenge 3:

lab setup:

lab time

introduction:

This activity is designed to challenge the mind of the students in the end of their learning process. The last modules focus on inputs and outputs from the console, files and converting types. Which is an important concept to know because applications work with the files on the disk systems on computers.

objectives:

In this exercise, the student will complete the same code from the second activity. But this time, we will change the input process and add more capabilities to our code. Based on a file, the code will read and output on the console the results.

- Read and write text files.
- Input and Output to console.
- Converting types.
- Regular expressions.

Before we get to the code, a small introduction to regular expression is needed in order to understand what comes next:

Regular expressions provide a powerful, flexible, and efficient method for processing text. The extensive pattern-matching notation of regular expressions enables you to quickly parse large amounts of text to:

- Find specific character patterns.
- Validate text to ensure that it matches a predefined pattern (such as an email address).
- Extract, edit, replace, or delete text substrings.
- Add extracted strings to a collection to generate a report.

For many applications that deal with strings or that parse large blocks of text, regular expressions are an indispensable tool. To know more about regular expressions, see .NET Regular Expressions.

exercise:

In this activity, our main challenge is to adapt our code to accept inputs via a text file. Now that our code is in OOP. As said, we will change the input. With this rises a new challenge, how can we know that the text we read in the file is which value or figure. To do that, as part of the challenge, we will use and introduce regular expressions.

Our code will calculate the three different figures at once. And will calculate either “doubles” value or “int” values. The floating point in a variable will be written with points and variables need to be separated by comas. And those values will need to be specified on the “input.txt” file provided in the same folder as the file for the C# code.

Example of the input text file:

```
figure,var1,var2,var3

triangle,12.2,12.1,12.2
rectangle,12.1,11.2
circle,12.2
```

1. If we review the top part of our code. We will see that our “abstract class” is containing two more methods. The “CalculatePerimeter” and the “GetSides”. Calculate perimeter is straight forward. But the getsides will be the method used to read the variables from a .txt file.

So, now let’s review the “getsides” method inside the Square class.

```
public override void GetSides()
        {
            string pattern = @"^[Rr]ectangle,";
            Regex rg = new Regex(pattern);
            //creating an object of NumberFormatInfo
            NumberFormatInfo provider = new NumberFormatInfo();
            provider.NumberDecimalSeparator = ".";
```

In this part, we are creating a Regex pattern or a string pattern so we can recognize the word “Rectangle,” or “rectangle,”.

2. After we specified the pattern to recognize, let’s open the “input.txt” file. So, we “try” to open the file and we assign each new line read to the variable “line”. We create an instance of the Streamreader and since we are using “using”. The StreamReader will also be closed after being used.

```
try {
                // Create an instance of StreamReader to read from a file.
                // The using statement also closes the StreamReader.
                using (StreamReader sr = new StreamReader("input.txt")) {
                    string line;
```

3. Now that we have the pattern ready and the file also ready to be read. Let’s try to recognize an read the values.

```
while ((line = sr.ReadLine()) != null) {
     if(Regex.Match(line , pattern).Success){
                Console.WriteLine(line);
```

```
                try
{
  //try for variables with floating points. Only accepts the same type of
variables.
                                  //rectangle,12.3,12.4
  string [] sides = Regex.Split(line, @"(^[rR]ectangle),([\d]+\.[\d]+),([\d]
+\.[\d]+)");
  this.side1 = Double.Parse(sides[2],
System.Globalization.CultureInfo.InvariantCulture);
  this.side2 = Double.Parse(sides[3],
System.Globalization.CultureInfo.InvariantCulture);
}
catch
{
//or rectangle,12,13
  string [] sides = Regex.Split(line, @"(^[rR]ectangle),([\d]+),([\d]+)");
  this.side1 = Double.Parse(sides[2],
System.Globalization.CultureInfo.InvariantCulture);
  this.side2 = Double.Parse(sides[3],
System.Globalization.CultureInfo.InvariantCulture);
  }
 }
}
```

Let's review together what's happening here. The while part is here for us to read each line of the file. After that, we have an "If", which is here to recognize the pattern previously set("Rectangle or rectangle"). If there is a match, we write it on the console so we can track it later and now where our code is currently running and help the user understand the output.

If a line matches the pattern, we review the lines so we can extract the value. But to do that. We need to be sure of what we are extracting. So, we check to different structures. If we remember, we said that our code will only accept double or int value. But not mixed. Either we have:

- Rectangle,12,12 - *^[rR]ectangle,([\d]+),([\d]+)*

Or

- Rectangle,12.1,12.1 - *^[rR]ectangle,([\d]+\.[\d]+),([\d]+\.[\d]+)*

\d: matches any decimal digit

\.: matches points

+: Matches the previous element one or more times.

With that in mind, we try to split the line in the two different ways using *Regex.Split(line, @"(^[rR]ectangle),([\d]+),([\d]+)")* and we assign those variables to our sides of our figure.

Keep in mind that when we split our line, we parse the matches to a list.

4. Perfect! We've just completed the first class of our code. Now, based on what we've just done. Let's do the other two classes. Why don't we start doing the triangle class, which is a bit longer since we need three variables.

```
class Triangle : Shape
{
      //code here
}
```

We create the class "triangle" based on the abstract class "Shape".

5. Then we populate it with the variables and constructors:

```
private double tside1;
private double tside2;
private double tside3;
public Triangle() { }
public Triangle(double tside1, double tside2, double tside3)
{
      this.tside1 = tside1;
      this.tside2 = tside2;
      this.tside3 = tside3;
}
```

6. If we still remember, we need to create the "CalculateArea" method just like the previous exercise.

```
public override double CalculateArea()
{
      double s = Convert.ToDouble((this.tside1+this.tside2+this.tside3)/2);
      double area = Convert.ToDouble(Math.Sqrt(s*(s-this.tside1)*(s-
this.tside2)*(s-this.tside3)));
      return area;
}
```

7. And of course, the new method "CalculatePerimeter". Which is simpler to implement than the area, but it will make look our code fancier.

```
public override double CalculatePerimeter()
{
      double perimeter = this.tside1+this.tside2+this.tside3;
      return perimeter;
}
```

8. And now, the good part. We need one last part in this class. The "GetSides" method. This time, our patter to match will be "Triangle" or "triangle".

```
public override void GetSides()
{
      string pattern = @"^[Tt]riangle,";
      Regex rg = new Regex(pattern);
      //creating an object of NumberFormatInfo
      NumberFormatInfo provider = new NumberFormatInfo();
      provider.NumberDecimalSeparator = ".";
```

9. Then we try to read the “input.txt” file:

```
try {
      // Create an instance of StreamReader to read from a file.
      // The using statement also closes the StreamReader.
      using (StreamReader sr = new StreamReader("input.txt")) {
            string line;
            // Read and display lines from the file until
            // the end of the file is reached.
```

10. Now we iterate through each line and write it on the console for the user:

```
while ((line = sr.ReadLine()) != null) {
      if(Regex.Match(line , pattern).Success){
            Console.WriteLine(line);
```

11. And finally, we try both patterns. Floating and integer values to then split the line:

```
try
{
      //try for variables with floating points
      string [] sides = Regex.Split(line, @"(^[Tt]riangle),([\d]+\.[\d]+),([\
d]+\.[\d]+),([\d]+\.[\d]+)");
      this.tside1 = Double.Parse(sides[2],
System.Globalization.CultureInfo.InvariantCulture);
      this.tside2 = Double.Parse(sides[3],
System.Globalization.CultureInfo.InvariantCulture);
      this.tside3 = Double.Parse(sides[4],
System.Globalization.CultureInfo.InvariantCulture);
}
catch
{
      string [] sides = Regex.Split(line, @"(^[Tt]riangle),([\d]+),([\d]+),([\
d]+)");
      this.tside1 = Double.Parse(sides[2],
System.Globalization.CultureInfo.InvariantCulture);
      this.tside2 = Double.Parse(sides[3],
System.Globalization.CultureInfo.InvariantCulture);
      this.tside3 = Double.Parse(sides[4],
System.Globalization.CultureInfo.InvariantCulture);
}
```

Perfect, now wee simply need the last class and to check our main code.

12. Same as the other two classes. We first create it:

```
class Circle : Shape
{
      //code here
}
```

13. Then we specify the variables and constructors:

```
private double radius;
public Circle() { }
public Circle(double r)
{
      this.radius = r;
}
```

14. Now the methods to calculate the area and perimeter:

```
public override double CalculateArea()
{
      double area = (double)(Math.Pow(this.radius, 2) * Math.PI);
      return area;
}
public override double CalculatePerimeter()
{
      double perimeter = Math.PI*2*this.radius;
      return perimeter;
}
```

15. And finally, the method to get our radius, the same way as the others two. So first, we create it.

```
public override void GetSides()
{
      //code here
}
```

16. Then we specify our new pattern:

```
string pattern = @"^[Cc]ircle,-?\d+(?:\.\d+)?";
Regex rg = new Regex(pattern);
//creating an object of NumberFormatInfo
NumberFormatInfo provider = new NumberFormatInfo();
provider.NumberDecimalSeparator = ".";
```

17. Of course, we try to open our input file:

```
try {
      using (StreamReader sr = new StreamReader("input.txt")) {
      string line;
```

18. Just after that, we iterate through each line and output the line read if it matches the pattern:

```
while ((line = sr.ReadLine()) != null) {
      if(Regex.Match(line , pattern).Success){
      Console.WriteLine(line);
```

19. And finally, now we try for both patterns and split the line to assign our radius value to his variable.

```
try
{
      //try for variables with floating points
      string [] sides = Regex.Split(line, @"(^[Cc]ircle),([\d]+\.[\d]+)");
      this.radius = Double.Parse(sides[2],
System.Globalization.CultureInfo.InvariantCulture);
}
Catch
{
      string [] sides = Regex.Split(line, @"(^[Cc]ircle),([\d]+)");
      this.radius = Double.Parse(sides[2],
System.Globalization.CultureInfo.InvariantCulture);
}
```

You've done it! We have all our classes ready to be used. But if we remember well. We miss one last part. The main area. Since our code drastically change the way of reading our figures and sides. We need to change the main part.

20. First, we need to create the class of our main:

```
class App
{
      //code here
}
```

21. Then we create the main:

```
static void Main(string[] args)
{
      //code here
}
```

22. So our code look a bit fancier, let's output something like "Hello, I will calculate Perimeter and Area of several geometric figures":

```
Console.WriteLine("Hello, I will calculate Perimeter and Area of several
geometric figures");
```

23. If we remember the second challenge. We first need to create our object:

```
Shape square = new Square();
```

But now, we don't need to ask to the user for the sides again. Since we implemented a new method that will read and assign our sides from a text file. We need to call the method and then we call the method to calculate the area and perimeter of our figure

24. Let's get our sides:

```
square.GetSides();
```

As simply as that

25. And now the methods to calculate:

```
Console.WriteLine("The area of the square/recangle is equal to {0}",
square.CalculateArea());
Console.WriteLine("The perimeter of the square/recangle is equal to {0}",
square.CalculatePerimeter());
```

Well done, now we have basically created our object, call it's method to find values, calculated the area and perimeter and give to the user as an output the results.

Now, let's do the same for the other two.

26. For our triangles:

```
//triangle calculation
Shape triangle = new Triangle();
triangle.GetSides();
Console.WriteLine("The area of the triangle is equal to {0}",
triangle.CalculateArea());
Console.WriteLine("The perimeter of the triangle is equal to {0}",
triangle.CalculatePerimeter());
```

27. And now for our circles:

```
//circle calculation
Shape circle = new Circle();
circle.GetSides();
Console.WriteLine("The area of the circle is equal to {0}",
circle.CalculateArea());
Console.WriteLine("The perimeter of the circle is equal to {0}",
circle.CalculatePerimeter());
```

We've done it! Our code is completed. Let's review possible inputs and outputs:

- Input:
  - triangle,12.2,12.1,12.2
  - rectangle,12.1,11.2
  - circle,12.2
- Output:

```
Hello, I will calculate Perimeter and Area of several geometric figures
The perimeter of the square/recangle is equal to 46,599999999999994
triangle,12.2,12.1,12.2
The area of the triangle is equal to 64,09503173998748
The perimeter of the triangle is equal to 36,5
circle,12.2
The area of the circle is equal to 467,59465056030473
The perimeter of the circle is equal to 76,65486074759094
```

- Input:
    - Triangle,12,12,12
    - Rectangle,12,11
    - Circle,12
- Output: