# Behaviour of Class

# Storing Data in Procedural Way

We have made following Parallel arrays to store records of the students.

```
int array_size = 5;
string [] sname = new string[array_size];
float [] matricMarks = new float[array_size];
float [] fscMarks = new float[array_size];
float [] ecatMarks = new float[array_size];
float[] aggregate = new float[array_size];
```

# Operations on Data

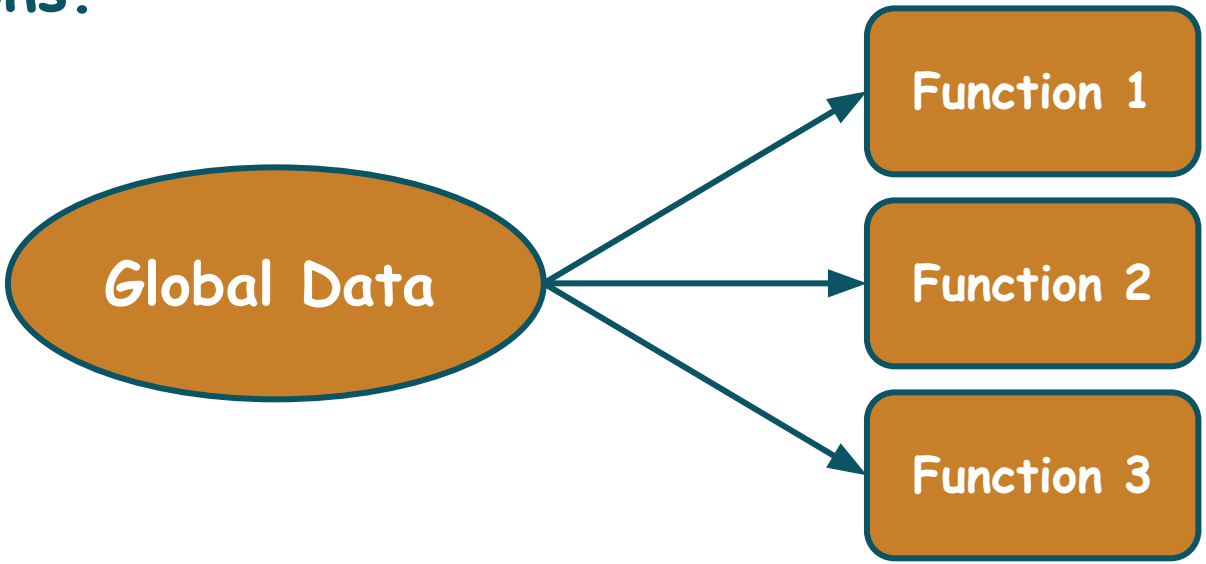
We define functions to perform operations on the Data but the function and data are highly decoupled.

```
aggregate[0] = calculateMerit(matricMarks[0], fscMarks[0], ecatMarks[0]);
```

```
static float calculateMerit(float mMarks, float fscMarks, float ecatMarks)
{
    float score;
    score = ((0.25F * (mMarks / 1050)) + (0.45F * (fscMarks / 550)) + (0.3F *
             (ecatMarks / 400))) * 100;
    return score;
}
```

# Operations on Data

Different functions use this global data. It is possible one type of global data is used by many different functions.



# Operations on Data

For example, during the Departmental Store Management System, we created a class product.

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;
}
```

# Operations on Data

For example, during the Departmental Store Management System, we created a class product and a function (separately) to calculate tax.

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;
}
```

```
static float calculateTax(Product p)
{
        float tax;
        if (p.category == "Grocery"){
            tax = p.price * 10 / 100F;
        }
        else if (p.category == "Fruit"){
            tax = p.price * 5 / 100F;
        }
        else{
            tax = p.price * 15 / 100F;
        }
        return tax;
}
```

# Operations on Data

For example, during the Departmental Store Management System, we created a class product and a function (separately) to calculate tax.

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;
}
```

```
static float calculateTax(Product p)
{
        float tax;
        if (p.category == "Grocery"){
            tax = p.price * 10 / 100F;
        }
        else if (p.category == "Fruit"){
            tax = p.price * 5 / 100F;
        }
        else{
            tax = p.price * 15 / 100F;
        }
        return tax;
}
```

# Operations on Data

The function calculateTax() is called by some program that is passed the record of object p.

```
List < Product > allProducts = new List<Product>();
…
if (option == 4)
{
    float tax;
    for (int x = 0; x < allProducts.Count; x++)
    {
        tax = calculateTax(allProducts[x]);
        Console.WriteLine("{0}\t\t{1}\t\t{2}\t\t{3}", allProducts[x].name, allProducts[x].category,
        allProducts[x].price, tax);
    }
}
```

# Operations on Data

We write the functions/code to perform operations on the Data but the functions/code and data are highly decoupled. It means data is placed somewhere else and the functions which are applying computations on that data are defined somewhere else.

# Operations on Data: Is it Problematic?

We write the functions/code to perform operations on the Data but the functions/code and data are highly decoupled. It means data is placed somewhere else and the functions which are applying computations on that data are defined somewhere else.

# Operations on Data: Is it Problematic?

With the growth of the software, these function increases.

# Operations on Data: Is it Problematic?

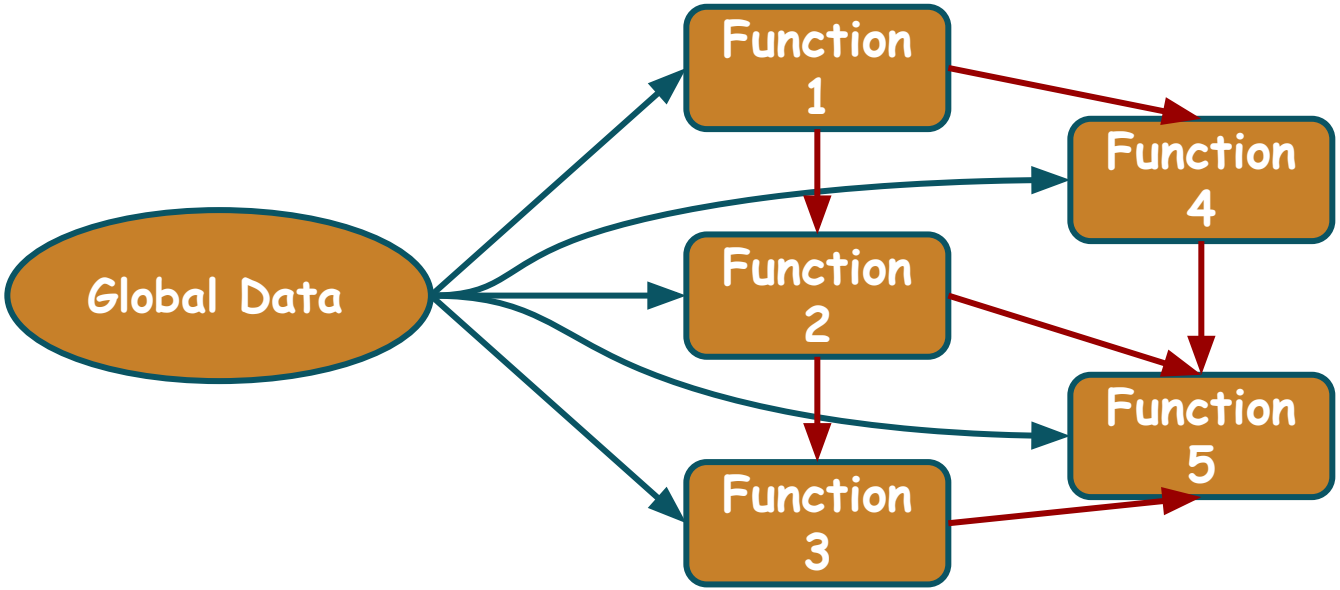
It gets difficult to keep track about all the locations (function and code) that are manipulating the data.

# Operations on Data: Is it Problematic?

Also, a lot of communication between functions increase the coupling and the dependences.

# Operations on Data: Is it Problematic?

Gradually, a situation could arise which is called Spaghetti code.



# Operations on Data: Maintainability Issue

If any change occurs in the data all other functions need to be changed but they are distributed all over the code and we may miss any change that leads to the crash of the software.

# Operations on Data: Maintainability Issue

For example, for some reason, we need to change or remove some attribute of class. Then, all the functions those are using that attribute of the class will give errors.

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;
}
```

```
class Product
{
    public string name;
    public string category;
    public float price;
    public int stock;
    public int minimumStock;
}
```

# Operations on Data: Maintainability Issue

So, what is the Solution?

# Object Oriented Philosophy

States that Data and functions which are related should be grouped together and that data and functions which are not related should not interfere with each other.

# Functions in Class

Lets see how to put data and functions in the same class but before recall current situation.

# Functions in Class

Lets see how to put data and functions in the same class but before recall current situation.

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;
}
```

```
static float calculateTax(Product p)
{
        float tax;
        if (p.category == "Grocery"){
            tax = p.price * 10 / 100F;
        }
        else if (p.category == "Fruit"){
            tax = p.price * 5 / 100F;
        }
        else{
            tax = p.price * 15 / 100F;
        }
        return tax;
}
```

# Functions in Class

We have both data (in class) and function separately. We pass the data in form of object to function so it can process.

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;
}
```

```
static float calculateTax(Product p)
{
        float tax;
        if (p.category == "Grocery"){
            tax = p.price * 10 / 100F;
        }
        else if (p.category == "Fruit"){
            tax = p.price * 5 / 100F;
        }
        else{
            tax = p.price * 15 / 100F;
        }
        return tax;
}
```

# Functions in Class

We have both data (in class) and function separately. We pass the data in form of object to function so it can process.

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;
}
```

```
static float calculateTax(Product p)
{
        float tax;
        if (p.category == "Grocery"){
            tax = p.price * 10 / 100F;
        }
        else if (p.category == "Fruit"){
            tax = p.price * 5 / 100F;
        }
        else{
            tax = p.price * 15 / 100F;
        }
        return tax;
}
```

# Functions in Class

Now, lets see how we can combine the data and functions both in the same class.

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;
}
```

```
static float calculateTax(Product p)
{
        float tax;
        if (p.category == "Grocery"){
            tax = p.price * 10 / 100F;
        }
        else if (p.category == "Fruit"){
            tax = p.price * 5 / 100F;
        }
        else{
            tax = p.price * 15 / 100F;
        }
        return tax;
}
```

# Functions in Class

Now, lets see how we can combine the data and functions both in the same class.

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;

public float calculateTax()
{
      float tax;
      if (category == "Grocery"){
         tax = price * 10 / 100F;
      }
      else if (category == "Fruit"){
         tax = price * 5 / 100F;
      }
      else{
         tax = price * 15 / 100F;
      }
      return tax;
}
}
```

# Functions in Class

Now, lets see how we can combine the data and functions both in the same class.

First important thing to note here is that we are not using static keyword.

Second important thing we will use public keyword so the function is visible in the main function of separate file.

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;

public float calculateTax()
{
      float tax;
      if (category == "Grocery"){
         tax = price * 10 / 100F;
      }
      else if (category == "Fruit"){
         tax = price * 5 / 100F;
      }
      else{
         tax = price * 15 / 100F;
      }
      return tax;
}
}
```

# Functions in Class

Now, lets see how we can combine the data and functions both in the same class.

Important thing to note here is that we have not passed an object to this function.

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;

public float calculateTax()
{
      float tax;
      if (category == "Grocery"){
         tax = price * 10 / 100F;
      }
      else if (category == "Fruit"){
         tax = price * 5 / 100F;
      }
      else{
         tax = price * 15 / 100F;
      }
      return tax;
}
}
```

# Functions in Class

Now, lets see how we can combine the data and functions both in the same class.

**Note:** we are accessing the variable without the dot operator.

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;

public float calculateTax()
{
      float tax;
      if (category == "Grocery"){
          tax = price * 10 / 100F;
      }
      else if (category == "Fruit"){
          tax = price * 5 / 100F;
      }
      else{
          tax = price * 15 / 100F;
      }
      return tax;
}
}
```

# Functions in Class

Now you can call the function as:

```
tax = p.calculateTax();
```

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;

public float calculateTax() {
      float tax;
      if (category == "Grocery"){
         tax = price * 10 / 100F;
      }
      else if (category == "Fruit"){
         tax = price * 5 / 100F;
      }
      else{
         tax = price * 15 / 100F;
      }
      return tax;
  }
}
```

# Functions in Class

Now you can call the function as:

```
tax = p.calculateTax();
```

calculateTax() function is now called the behaviour of the class.

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;

public float calculateTax()
{
      float tax;
      if (category == "Grocery"){
          tax = price * 10 / 100F;
      }
      else if (category == "Fruit"){
          tax = price * 5 / 100F;
      }
      else{
          tax = price * 15 / 100F;
      }
      return tax;
}
}
```

# Functions in Class

Now you can call the function as:

```
tax = p.calculateTax();
```

All the related functions of the class are generally known as the behaviours of the class.

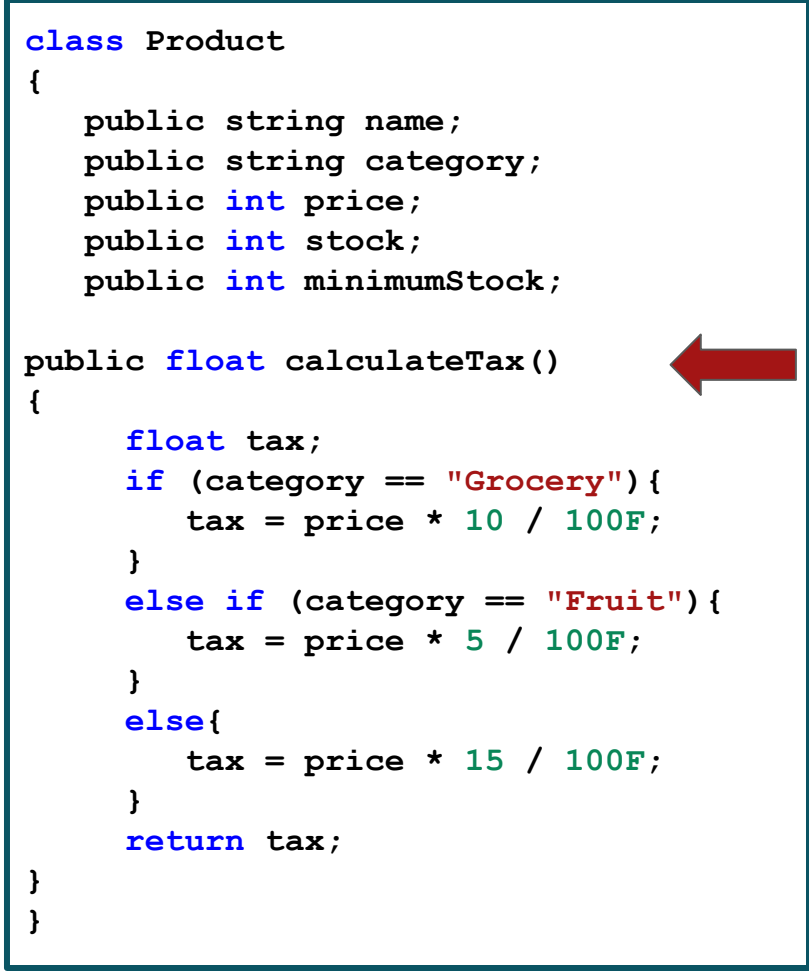
```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;

public float calculateTax()
{
      float tax;
      if (category == "Grocery"){
         tax = price * 10 / 100F;
      }
      else if (category == "Fruit"){
         tax = price * 5 / 100F;
      }
      else{
         tax = price * 15 / 100F;
      }
      return tax;
}
}
```

# Functions in Class

Now you can call the function as:

```
tax = p.calculateTax();
```

We call the behaviour of the class also with the dot (.) operator.



```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;

public float calculateTax()
{
      float tax;
      if (category == "Grocery"){
         tax = price * 10 / 100F;
      }
      else if (category == "Fruit"){
         tax = price * 5 / 100F;
      }
      else{
         tax = price * 15 / 100F;
      }
      return tax;
}
}
```

# How it Works?

Let's create an Object of Product class.

```
Product p = new Product();
```

p

| | |
|---|---|
| name | "" |
| category | "" |
| price | 0 |
| stock | 0 |
| minimumStock | 0 |

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;

public float calculateTax()
{
      float tax;
      if (category == "Grocery"){
         tax = price * 10 / 100F;
      }
      else if (category == "Fruit"){
         tax = price * 5 / 100F;
      }
      else{
         tax = price * 15 / 100F;
      }
      return tax;
}
}
```

# How it Works?

Let's create an Object of Product class.

```
Product p = new Product();
```

p

| | |
|---|---|
| name | "" |
| category | "" |
| price | 0 |
| stock | 0 |
| minimumStock | 0 |

When the constructor is called it allocates memory in the heap for the object attributes and set the default values.

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;

public float calculateTax()
{
      float tax;
      if (category == "Grocery"){
         tax = price * 10 / 100F;
      }
      else if (category == "Fruit"){
         tax = price * 5 / 100F;
      }
      else{
         tax = price * 15 / 100F;
      }
      return tax;
}
}
```

# How it Works?

Let's create an Object of Product class.

```
Product p = new Product();
p.name = "Milk";
p.category = "Grocery";
p.price = 92;
p.stock = 10;
```

p

| | |
|---|---|
| name | Milk |
| category | Grocery |
| price | 92 |
| stock | 10 |
| minimumStock | 0 |

We can change the values afterwards.

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;

public float calculateTax()
{
      float tax;
      if (category == "Grocery"){
         tax = price * 10 / 100F;
      }
      else if (category == "Fruit"){
         tax = price * 5 / 100F;
      }
      else{
         tax = price * 15 / 100F;
      }
      return tax;
}
}
```

# How it Works?

Let's create another Object of Product class.

```
Product p = new Product();
p.name = "Milk";
p.category = "Grocery";
p.price = 92;
p.stock = 10;
```

p

| | |
|---|---|
| name | Milk |
| category | Grocery |
| price | 92 |
| stock | 10 |
| minimumStock | 0 |

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;

public float calculateTax()
{
      float tax;
      if (category == "Grocery"){
         tax = price * 10 / 100F;
      }
      else if (category == "Fruit"){
         tax = price * 5 / 100F;
      }
      else{
         tax = price * 15 / 100F;
      }
      return tax;
}
}
```

# How it Works?

Let's create another Object of Product class.

```
Product p = new Product();
p.name = "Milk";
p.category = "Grocery";
p.price = 92;
p.stock = 10;
```

```
Product p1 = new Product();
```

This allocates different memory for second object

p

| | |
|---|---|
| name | Milk |
| category | Grocery |
| price | 92 |
| stock | 10 |
| minimumStock | 0 |

p1

| | |
|---|---|
| name | "" |
| category | "" |
| price | 0 |
| stock | 0 |
| minimumStock | 0 |

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;

public float calculateTax()
{
      float tax;
      if (category == "Grocery"){
         tax = price * 10 / 100F;
      }
      else if (category == "Fruit"){
         tax = price * 5 / 100F;
      }
      else{
         tax = price * 15 / 100F;
      }
      return tax;
}
}
```

# How it Works?

Let's create another Object of Product class.

```
Product p = new Product();
p.name = "Milk";
p.category = "Grocery";
p.price = 92;
p.stock = 10;
```

```
Product p1 = new Product();
p1.name = "Apple";
p1.category = "Fruit";
p1.price = 22;
p1.stock = 10;
```

p

| | |
|---|---|
| name | Milk |
| category | Grocery |
| price | 92 |
| stock | 10 |
| minimumStock | 0 |

p1

| | |
|---|---|
| name | Apple |
| category | Fruit |
| price | 20 |
| stock | 10 |
| minimumStock | 0 |

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;

public float calculateTax()
{
      float tax;
      if (category == "Grocery"){
         tax = price * 10 / 100F;
      }
      else if (category == "Fruit"){
         tax = price * 5 / 100F;
      }
      else{
         tax = price * 15 / 100F;
      }
      return tax;
}
}
```

# How it Works?

## Let's calculate tax.

```
Product p = new Product();
p.name = "Milk";
p.category = "Grocery";
p.price = 92;
p.stock = 10;
float tax = p.calculateTax();
```

```
Product p1 = new Product();
p1.name = "Apple";
p1.category = "Fruit";
p1.price = 22;
p1.stock = 10;
float tax = p1.calculateTax();
```

p

| | |
|---|---|
| name | Milk |
| category | Grocery |
| price | 92 |
| stock | 10 |
| minimumStock | 0 |

p1

| | |
|---|---|
| name | Apple |
| category | Fruit |
| price | 20 |
| stock | 10 |
| minimumStock | 0 |

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;

public float calculateTax()
{
      float tax;
      if (category == "Grocery"){
         tax = price * 10 / 100F;
      }
      else if (category == "Fruit"){
         tax = price * 5 / 100F;
      }
      else{
         tax = price * 15 / 100F;
      }
      return tax;
}
}
```

# How it Works?

## Let's calculate tax.

```
Product p = new Product();
p.name = "Milk";
p.category = "Grocery";
p.price = 92;
p.stock = 10;
float tax = p.calculateTax();
```

9.2

```
Product p1 = new Product();
p1.name = "Apple";
p1.category = "Fruit";
p1.price = 22;
p1.stock = 10;
float tax = p1.calculateTax();
```

p

| | |
|---|---|
| name | Milk |
| category | Grocery |
| price | 92 |
| stock | 10 |
| minimumStock | 0 |

p1

| | |
|---|---|
| name | Apple |
| category | Fruit |
| price | 20 |
| stock | 10 |
| minimumStock | 0 |

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;

public float calculateTax()
{
      float tax;
      if (category == "Grocery"){
         tax = price * 10 / 100F;
      }
      else if (category == "Fruit"){
         tax = price * 5 / 100F;
      }
      else{
         tax = price * 15 / 100F;
      }
      return tax;
}
}
```

# How it Works?

## Let's calculate tax.

```
Product p = new Product();
p.name = "Milk";
p.category = "Grocery";
p.price = 92;
p.stock = 10;
float tax = p.calculateTax();
```

9.2

```
Product p1 = new Product();
p1.name = "Apple";
p1.category = "Fruit";
p1.price = 22;
p1.stock = 10;
float tax = p1.calculateTax();
```

1.1

p

| | |
|---|---|
| name | Milk |
| category | Grocery |
| price | 92 |
| stock | 10 |
| minimumStock | 0 |

p1

| | |
|---|---|
| name | Apple |
| category | Fruit |
| price | 20 |
| stock | 10 |
| minimumStock | 0 |

```
class Product
{
    public string name;
    public string category;
    public int price;
    public int stock;
    public int minimumStock;

public float calculateTax()
{
      float tax;
      if (category == "Grocery"){
         tax = price * 10 / 100F;
      }
      else if (category == "Fruit"){
         tax = price * 5 / 100F;
      }
      else{
         tax = price * 15 / 100F;
      }
      return tax;
}
}
```

# Conclusion

- Object Oriented Philosophy is Related data and the functions that operate on the data should be stored within the same unit and that is class.
- The functions written in the class are generally known as behaviour of the class.
- The behaviour of the class are also called using the dot operator.

# Learning Objective

**Write class with appropriate behaviour on the data.**

# Self Assessment:

1. Extend your previous Circle Class with the following member functions/behaviours.

Three methods/behavoiurs:

1. getArea()
2. getDiameter()
3. getCircumference()

which return the area, diameter and circumference of the instances of Circle Class, respectively