INTRODUCTION TO FORCE.COM PLATFORM

Salesforce provides "Salesforce CRM" as part of SAAS Feature, which is used to automate all the existing and new Customers information at one place.

By using Salesforce CRM, we can Automate the sales, service and marketing processes of the organization.

While using Salesforce CRM, we have the below **Limitations**.

1. We can arrange the fields maximum in two columns inside the section in the Page Layout. i.e. We can't arrange the fields in more than 2 Columns in a Section.

2. We can't customize the background colour and image of the Standard User Interfaces.

3. We can create maximum of 100 Active Validation Rules per an Object.

4. We can't implement the De-Duplication Processes, to eliminate the Duplicate Rules based on the combination of fields in the object.

5. While sharing the records to the users, we can't select the users dynamically at runtime, and we can't select the level of access dynamically at runtime.

To avoid all the above limitations, we use "Force.com" platform.

Force.com is a Cloud Platform given by Salesforce as Platform as a Service, which is used to implement the customizations based on the need.

Force.com Platform has been introduced by Salesforce in Enterprise Edition, Unlimited Edition and Free Developer Edition.

By using Force.com Platform, we can perform the below Operations.

1. Customize the Existing Salesforce CRM Application features according to the business requirement.

2. Leverage the Existing Application features, by adding the Additional Enhancements.

3. Build the Custom Applications based on the requirement.

Ex: Banking, Finance, Insurance, HealthCare, Pharma, E-Commerce, Supply-Chain etc.

4. Integrate the Salesforce Applications with any External System.

Ex: Banking Systems - Payment Gateways, Finance Apps, Mobile Apps, Java / .Net Apps etc.

Force.com Platform provides 2 Programming Languages.

1. **Apex Programming**:

It is used to build the Custom Business Logics, perform the operations inside the application based on the need.

2. **Visualforce Programming**:

It is a Web development framework, used to design attractive and dynamic User Interfaces.

APEX PROGRAMMING

Apex is a Cloud-Based Programming Language, which doesn't require any installations inside the device. We can Access the Apex programming features using "Browser and Internet".

Apex is an Object-Oriented Programming, which supports the Object-Oriented Programming Principles.

In Apex programming, each Business Logic should be implemented in the form of "Class and Object".

By using Apex Programming, we can implement the Custom Validations, Complex Business Logics, De-Duplication Processes and Complex Transactional Flows.

Apex is Not a Case-Sensitive Programming.

Each Statement in Apex programming, should be ends with a Semicolon.

Apex programming allows us to perform the DML Operations on one or more records inside the Salesforce Objects.

(Ex: INSERT, UPDATE, DELETE, UNDELETE, UPSERT, MERGE.)

By using Apex Programming, we can fetch the records from one / more Salesforce Objects with the help of a Query Language called as "SOQL".

(SOQL ---> Salesforce Object Query Language)

By using Apex Programming, we can search for a particular content in multiple Salesforce Objects, using a Search Engine called as "SOSL".

(SOSL ---> Salesforce Object Search Language)

Apex provides "Batch Processing and Schedule Programming", through which we can process millions of records, which involves the complex operations.

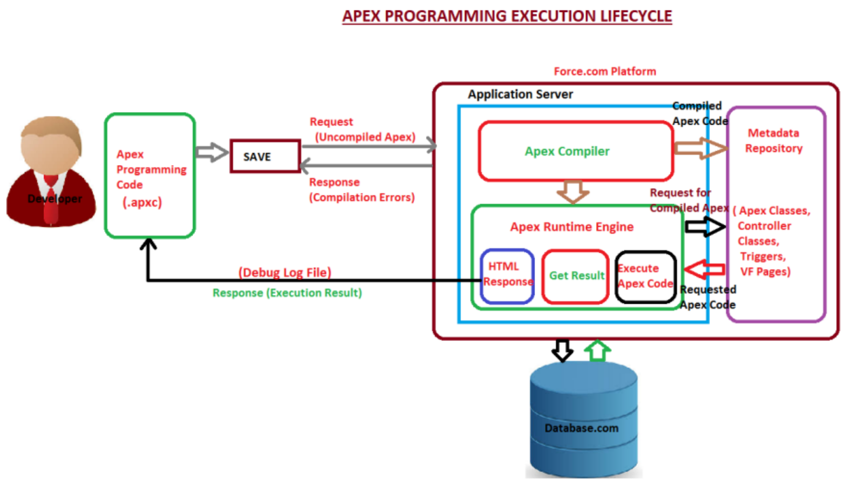
By using Apex Programming, we can Integrate the Salesforce Applications with the Third-Party Systems using REST API, and SOAP API.

(REST ---> Representational State Transfer

SOAP ---> Simple Object Access Protocol

API ---> Application Programming Interfaces)

<u>APEX PROGRAMMING EXECUTION LIFECYCLE</u>


APEX PROGRAMMING EXECUTION LIFECYCLE
Force.com Platform
Application Server
Developer
Apex Programming Code (.apxc)
SAVE
Request (Uncompiled Apex)
Response (Compilation Errors)
Apex Compiler
Compiled Apex Code
Metadata Repository
Request for Compiled Apex
(Apex Classes, Controller Classes, Triggers, VF Pages)
Apex Runtime Engine
HTML Response
Get Result
Execute Apex Code
Requested Apex Code
(Debug Log File)
Response (Execution Result)
Database.com

While writing the Apex Code, each Apex Programming Code will get compiled by "Apex Compiler", and executed by using a runtime Engine called as "Apex Run-time Engine".

Compiled Apex Code will get resides inside the "Metadata Repository" and a copy of the Code will be stored inside the "Apex Class" object.

Note: Each Apex Class Code will get stored in the file with the extension ".apxc".

We can write maximum of 60,00,000 apex characters code within the Organization.

Note: Once the code has been executed by the Apex runtime engine, results will be shown inside the "Debug Log File".

BUILDING BLOCKS OF APEX

Apex provides a set of fundamental building blocks, to implement the custom business logic based on the need.

1. Datatypes 2. Variables 3. Operators 4. Output Statements 5. Conditional Statements 6. Iterative Statements 7. Arrays

1. DATATYPES

Datatype describes

1. What type of data the variable can hold inside it.

2. How much memory needs to be reserved to store the value.

Apex provides the below two Categories of Datatypes.

i. PRIMITIVE DATATYPES

These are the Basic / Fundamental Datatypes, which has been given by Apex Programming Language.

Ex: Integer, Long, Double, Decimal, String, Boolean, ID, Date, Time, Date Time, Blob and Object.

String

It allows us to store either One / More Characters inside the variable.

i.e. We can store Alphanumerical Values along with special characters inside the variable.

String values should be enclosed with "Single Quote".

Each character inside the string variable occupies "1-Byte" of memory.

Note: Each Text Field, Text Area, Text Area Long, Text Area Rich, Text Encrypted, Auto Number, URL, Email, Phone, Picklist, Picklist-Multiselect, Geolocation field types will reference the "String DataType" by default.

Ex: 'M'

'Ravi Kumar'

'ravi.kumar@gmail.com'

'#104, RR Arcade, Kukatpally, Hyderabad-32'

ID

It allows us to store either "15/18-Characters Record Id" inside the variable.

Note: Each Id Field, Lookup Relationship, Master-Detail Relationship, External Lookup Relationship, Hierarchical Relationship fields will reference the "ID Datatype" by default.

Boolean

It allows us to store either "TRUE / FALSE" values inside the variable.

Note: Each Checkbox field will reference the "Boolean" datatype by default.

Date

It allows us to store the Date values inside the variable in the form of "MM/DD/YYYY" format.

Note: Each Date field will reference the "Date" datatype by default.

Time

It allows us to store the Time value inside the variable, either in 12-Hours / 24-Hours format.

Note: Each Time field will reference the "Time DataType" by default.

Date Time

It allows us to store the "Date and Time" stamp value inside a single variable.

Note: Each Date Time field will reference the "Date Time" datatype by default.

Ex: Created By, Last Modified By, Deletion Date etc.

Integer

It allows us to store the Numerical values without any decimal digits.

Note: Each Integer variable will occupy 4-Bytes of memory. Integer variable value can be between the range -2^31 to +2^31-1.

Ex: 1000

10

45000

1000000

Note: Each Number field, Percent field, Currency field will reference the Integer Datatype.

Long

It allows us to store the Larger Integer values, without any decimal digits.

We can store the values between the range -2^63 to +2^63-1.

Note: Each Long Datatype variable will occupy 8-Bytes of memory.

Decimal

It allows us to store the Numerical values along with the decimal digits.

Note: Each Currency field will reference the Decimal Datatype by default.

Ex: 5.1

5.10

15.5

1.5

12042.57

Double

It allows us to store the Numerical values with more number of decimal digits.

Ex: 10.450

Pi = 3.1416

10/3 = 3.33333333333

Blob (Binary Large Object)

It allows us to store the Binary Content of "images, Pictures, Audio, Video, PDF, Word etc.

Note: Each Attachment, Document Object, Images, pictures will reference the "Blob" datatype by default.

Object

It is a Generic Type, which allows us to store any type of value inside the variable.

Note: It requires explicit Type Casting, while managing the values.

ii. SOBJECT DATATYPES

These are used to store the Salesforce object records inside the variables.

Each Salesforce object name will be represented as a Datatype, which can store the associated object record inside the variable.

Ex: Account, Contact, Lead, Case, Campaign, User, Document, Profile, Opportunity, Task, Event, Position__C, Hiring_Manager__C, Employee__C, Candidate__C, Claim__C, Loan__C, SObject.

Note: SObject is a Generic Datatype, which can store any Salesforce object record inside the variable.

2. VARIABLES

Variables are nothing but the names, given to the memory location in order to store a value. Variables are used to store the values temporarily inside the application.

Apex is a tightly coupled programming language, which requires the variable definition first, before its usage. While defining the variable, we have to specify the Datatype, which allocates the required amount of memory.

Syntax: [Access Specifier] <DataType> <variableName>= [<Value>];

Rules: 1. Variable name should start with a "Character".

2. Variable name should be always a "single word". It should not have any spaces.

3. Variable name can include a special character "_"(Underscore).

4. It is a best practice to use the Camel Case to define the variables.

Ex:
```
Integer customerCode;
String customerName;
Date birthDate;
Decimal loanAmount;
Boolean isActive;
```

Note: While defining the variable, if the user didn't assign any value, the Apex will assign the default value as "Null" for each variable.

Note: We can define multiple variables of same Datatype in a single statement as below.

Ex:
```
Integer employeeCode, employeeAge;
String employeeName, designation, mailingAddress, emailId;
```

Date birthDate, joiningDate, relievingDate;

Decimal currentCTC, expectedCTC;

Assigning the Values: Once the variables has been defined, then we can assign the value for the required variable as below.

Syntax: <variableName> = <value>;

Ex: employeeCode = 400890021;

employeeName = 'Ram Kumar';

emailId = 'ram.kumar@gmail.com';

currentCTC = 1200000;

While defining the variable itself, we can assign the value as below.

Ex: Integer productCode = 9002134;

String productName = 'Washing Machine', manufacturer = 'LG Inc.';

Decimal unitPrice = 34000, discountedPrice = 31000;

3. OPERATORS

Operator is nothing but a Symbol, which performs the operations on the operands.

Apex provides the below 5 Categories of operators.

1. **Arithmetic / Mathematical Operators**:

+

-

*

/

Math.Mod()

Ex:

25000 + 9000 = 34000

'Welcome' + 12080 = Welcome12080

'Welcome' + 'Hyderabad' = Welcome Hyderabad

2. **Relational / Comparison Operators**:

<

<=

>

>=

== (Uses Text Comparison)

!=

Equals() (Uses Binary Comparison)

3. **Logical Operators**:

AND (&&)

OR (| |)

NOT (!)

4. **Increment / Decremental Operators**:

++ ---> Increment the Value by 1.

Ex: Integer customerAge = 24;

customerAge++; ---> O/p: 25

-- ---> Decrement the Value by 1.

Ex: Integer available Licenses = 2;

availableLicenses--; ---> O/p: 1

5. **Assignment Operators**:

=

+=

-=

*=

/=

Ex: Integer balanceAmount = 10000;

// Deposited 4,000 Rupees.

balanceAmount += 4000; ---> O/p: 14,000.

// Withdraw 2,000 Rupees.

balanceAmount -= 2000; ---> O/p: 12,000

// Multiply the Value.

balanceAmount *= 2; ----> O/p: 24,000

// Divide the Value.

balanceAmount /= 2; ----> O/p: 12,000.

4. OUTPUT STATEMENTS

By using Output Statements, we can Write / Print the message / results on the Debug Log File.

Syntax: System. Debug ('Message to Print.');

Ex: System. Debug ('Welcome to Apex Programming');

Ways to Write the Apex Code

Salesforce provides the below 4 ways to Write the Apex Code.

1. By using "Execute Anonymous Window".
2. By using "Standard Navigation".
3. By using "Developer Console".
4. By using "URL Format".

Ways to Execute the Apex Code

We can invoke the apex code from the below 10 ways.

1. By using "Execute Anonymous Window".
2. By using "Visualforce Pages".
3. By using "Apex Triggers".
4. By using "Batch Programming".
5. By using "Schedule Programming".
6. By using "Email Services".
7. By using "Process Builder".
8. By using "Webservices / API's".
9. By using "IDE's - Editors" (VS Code Editor)
10. By using "Lightning Flows / Flow Builder"

Execute Anonymous Window:

Click on "Setup" menu.

1. Click on "Developer Console" menu item.
2. Go to the Developer Console Editor.
3. Click on "Debug" menu.
4. Click on "Open Execute Anonymous window" menu item.

(OR)

From Developer Console, use the shortcut "CTRL+E".

5. Write the Code inside the Editor.
6. Select the Checkbox "Open Log".
7. Click on "Execute / Execute Highlighted" button.
8. View the Result in the Debug Log File.
    1. Go Back to the "Developer Console" Editor.
    2. Expand the Tab bar, and Click on "Logs" tab.
    3. Double Click on the "Recent Log File".
    4. Select the Checkbox "Debug Only".
    5. View the Results.

Use Case: Write an apex program, to Print the Welcome Message on the Debug Log File.

System. Debug('Welcome to Apex Programming');

Use Case: Write an apex program, to define Two Variables, and Assign the Values. And Print the Values on the Log File.

```
Integer customerCode;

String customerName;

System.debug(customerCode);

System.debug(customerName);

System.debug('Customer Code is....: ' + customerCode);

System.debug('Customer Name is....: '+ customerName);

customerCode = 11090188;

customerName = 'Merck Pharma Inc.';

System.debug('Customer Code.....: '+ customerCode);

System.debug('Customer Name.....: '+ customerName);
```

Use Case: Write an apex program, to define Two Numerical variables and Assign the Values. Perform all the Mathematical Calculations and represent the result on the Log File.

```
Integer firstNumber, secondNumber, result;

firstNumber = 12800;

secondNumber = 460;

result = firstNumber + secondNumber;

System. Debug ('Addition Result is....: '+ result);


result = firstNumber - secondNumber;

System. Debug ('Subtraction Result is...: '+ result);


result = firstNumber * secondNumber;

System. Debug ('Multiplication Result is....: '+ result);


result = firstNumber / secondNumber;

system. Debug ('Division Result is....: '+ result);


result = Math.mod (firstNumber, secondNumber);

System. Debug ('Modulus Result is....: '+ result);
```

Use Case: Write an apex program, to Calculate the Sum of First 100 Numbers.

Formula : $n * (n + 1)/2$

```
Integer maxNumber, result;

maxNumber = 100;

result = ( maxNumber * (maxNumber + 1) )/ 2;

System. Debug ('Sum of First '+ maxNumber + ' Numbers is....: '+ result);
```

Use Case: Write an apex program, to Calculate the Simple Interest and Print the result on the Log File.

Formula: (PTR) / 100

```
Integer principleAmount, rateOfInterest, tenure, interestAmount;

principleAmount = 25000;

rateOfInterest = 2;

tenure = 15;

interestAmount = (principleAmount * rateOfInterest * tenure) / 100;

System. Debug ('Interest Amount is....: '+ interestAmount);
```

Use Case: Write an apex program, to Calculate the Total Marks and Average Marks of the Student.

(Note: We have 5 Subjects for the Student, like English, Hindi, Maths, Physics & Chemistry)

```
Integer english, maths, hindi, physics, chemistry, total, average;

english = 72;

hindi = 76;

maths = 94;

physics = 82;

chemistry = 91;

total = english + hindi + maths + chemistry + physics;

System. Debug ('Total Marks are....: '+ total);

average = total / 5;

System. Debug ('Average Marks are....: '+ average);
```

Assignments

1. Write an apex program, to Calculate the Bill Amount based on the Quantity and Unit Price.

2. Write an apex program, to Convert the Temperature from Fahrenheit to Centigrade.

   (Formula: (32°F − 32) × 5/9 = 0°C)

3. Write an apex program, to Convert the Temperature from Centigrade to Fahrenheit.

   (Formula: (0°C × 9/5) + 32 = 32°F)

5. **CONDITIONAL STATEMENTS**

Conditional Statements are used to add one / more User Defined conditions, to be get verified before executing the Statements.

By using Conditional Statements, we can change the control Flow of the application execution.

By using the Conditional Statements, we can avoid the Runtime Errors / Exceptions.

Apex provides the below 3 Conditional Statements.

1. IF Condition.

2. Switch Statement.

3. Ternary Operator.

IF Condition:

By using "IF Condition", we can add one or more conditions, to be get verified before executing the statements. We can add the multiple conditions by using "Logical Operators".

We have the below 4 formats of "If Conditional Statement".

1. Simple IF Condition.

2. IF Else Condition.

3. Else IF Condition.

4. Nested IF Condition.

Simple IF Condition

Syntax:

```
IF(<Conditions>)
{
        // Write the Statements
}
```

In this approach, if the all the specified conditions are satisfied, then it will execute the immediate block of statements. If the conditions are failed, then it will come out of the block.

Example

```
Integer firstNumber, secondNumber, result;
        firstNumber = 12800;
        secondNumber = 0;
result = firstNumber + secondNumber;
System.debug('Addition Result is....: '+ result);


result = firstNumber - secondNumber;
System.debug('Subtraction Result is...: '+ result);
```

```
        result = firstNumber * secondNumber;

        System.debug ('Multiplication Result is....: '+ result);

        If (secondNumber > 0)

  {

      result = firstNumber / secondNumber;

      system.Debug ('Division Result is....: '+ result);


      result = Math.mod (firstNumber, secondNumber);

      System.Debug ('Modulus Result is....: '+ result);

  }
```

IF ELSE CONDITION

Syntax:

```
        If(<Conditions>)

        {

                // Write the Statements

                  (TRUE Block Statements)

        }

        Else

        {

                // Write the FALSE Block Statements

        }
```

In this approach, if all the specified conditions are satisfied, then it will execute the Immediate Block of Statements. If the conditions are failed, then "Else block statements" will get executed.

Example

```
        Integer firstNumber, secondNumber, result;

                firstNumber = 12800;

                secondNumber = 0;

        result = firstNumber + secondNumber;

        System. Debug ('Addition Result is....: '+ result);


        result = firstNumber - secondNumber;

        System.debug('Subtraction Result is...: '+ result);
        result = firstNumber * secondNumber;
        System.debug('Multiplication Result is....: '+ result);

        If (secondNumber > 0)
  {
      result = firstNumber / secondNumber;
      system.debug('Division Result is....: '+ result);

      result = Math.mod (firstNumber, secondNumber);
      System.debug('Modulus Result is....: '+ result);
  }
        else
  {
      System. Debug ('Division and Modulus Operations Cannot be Performed');
  }
```

Use Case: Write an apex program, to find out the Biggest Number from the Given 2 Numbers.

```
               Integer firstNumber, secondNumber;
               firstNumber = 9000;
               secondNumber = 17500;
               if(firstNumber > secondNumber)
        System. Debug ('First Number is the Biggest number');
      else
        System. Debug('Second Number is the Biggest number');
```

<u>Else IF Condition</u>

<u>Syntax</u>:

```
        If(<Conditions>)
        {
               // Write the Statements
        }
        else if(<Conditions>)
        {
               // Write the Statements
```

```
}
else if(<Conditions>)
{
        // Write the Statements
}
else
{
        // Default Block Statements
}
```

In this approach, we can prepare multiple branches of Conditions, to be get verified before executing the Statements.

It will execute the only one block statements at a time. If any of the branch conditions are not satisfied, then it will execute the "Else Block" statements by default.

Use Case: Write an apex program, to find the Biggest Number from the given Two Numbers.

```
Integer firstNumber, secondNumber;
        firstNumber = 27500;
        secondNumber = 17500;
        if(firstNumber == secondNumber)
 System.debug('Both the Numbers are Identical.');
        Else if(firstNumber > secondNumber)
System.debug('First Number is the Biggest One.');
else
System.debug('Second Number is the Biggest One.');
```

UseCase: Write an apex program, to find out the Season name based on the Month number as below.

Month Number	**Season Name**
1 - 4	Winter Season
5 - 8	Summer Season
9 - 12	Spring Season
<1 or >12	Invalid Month Number.

```
Integer monthNumber = 17;

if(monthNumber >= 1 && monthNumber <= 4)
 System.debug('Winter Season.');
            else if(monthNumber >= 5 && monthNumber <= 8)
 System.debug('Summer Season.');
            else if(monthNumber >= 9 && monthNumber <= 12)
 System.debug('Spring Season.');
            else
 System.debug('Invalid Month Number. Please Enter the Value between 1 - 12.');
```

Nested IF Condition

By using this approach, we can use one if condition inside an another, to implement the complex requirements.

Syntax:

```
IF(<Conditions>)
{
            // Write the Business Logic.
            if(<Conditions>)
            {
                        // Write the Business Logic.
            }
            else
            {
                        // Write the Statements.
                        if(<Conditions>)
                        {
                                    // Write the Business Logic.
                        }
            }
}
```

```
/*
        Write an apex program, to find out the Student Result and Grade as below.
                1. We have 5 Subjects for the Student.
                        (Ex: English, Hindi, Maths, Physics, and Chemistry)
                2. Calculate the Total Marks of the Student.
                3. Find out the Student Result as below.
                        3.1. PASS: If the Student got more than 40 Marks in each Subject.
                                3.1.1. Display the Student Result and Total Marks.

                                3.1.2. Calculate the Average Marks of the Student.
                                3.1.3. Find out the Student Grade based on the Average
Marks as below.
                                        Average Marks
        Grade
                                        -------------------------------
                                                >= 60%
                GRADE A.
                                                >=50 && <60
        GRADE B.
                                                >=40 && <50
        GRADE C.
                        3.2. FAIL: If the Student got less than 40 Marks in any of the Subject.
                                3.2.1. Display the Student Result and Total Marks.
*/

        Integer english, hindi, maths, physics, chemistry, total, average;

                english = 74;
                hindi = 76;
                maths = 26;
                physics = 81;
                chemistry = 87;

                total = english + hindi + maths + physics + chemistry;

if(english >= 40 && hindi >= 40 && maths >= 40 && chemistry >= 40 && physics >= 40)
```

```
{
    System.debug('Congratulations...!! You got PASSED.');
    System.debug('Total Marks are.....: '+ total);

    average = total / 5;
    System.debug('Average Marks are....: '+ average);

    if(average >= 60)
        System.debug('GRADE A.');
    else if(average >= 50 && average < 60)
        System.debug('GRADE B.');
    else if(average >= 40 && average < 50)
        System.debug('GRADE C.');
}
            else
{
    System.debug('You got FAILED...!! Better luck Next Time.');
    System.debug('Total Marks are....: '+ total);
}
```

SWITCH STATEMENT

Switch Statement is used to compare a variable / expression value with multiple branches. If any of the branch value is matching with the variable / expression, then it will execute the associated block of statements, else it will execute the "Else Block" statements.

Note: While comparing the variable value with the branches, it will use the "Binary comparison".

Syntax: Switch ON <Variable name / Expression>

```
            {
                    When <Value1>
                    {
                            // Write the Statements.
                    }
                    When <Value2>
                    {
                            // Write the Statements.
```

```
                }
                ....
                ....
                When Null
                {
                        // Write the Statements.
                }
                ...
                When Else
                {
                        // Default Block Statements.
                }
        }
```

<u>Use Case</u>: Write an apex program, to Print the Week day Name based on the Week day Number as below.

Week Day Number	Week Day Name
1	Monday
2	Tuesday
3	Wednesday
4	Thursday
5	Friday
6	Saturday
7	Sunday
<1 OR >7	Invalid Week Day Number.

```
    Integer weekdayNumber = 11;

            Switch ON weekdayNumber
  {
     When 1
     {
        System. Debug ('Today is Monday.');
     }
     When 2
     {
        System. Debug ('Today is Tuesday.');
     }
     When 3
     {
        System. Debug ('Today is Wednesday.');
     }
     When 4
     {
        System. Debug ('Today is Thursday.');
     }
     When 5
     {
        System. Debug ('Today is Friday.');
     }
     When 6
     {
        System. Debug ('Today is Saturday.');
     }
     When 7
     {
        System. Debug ('Today is Sunday.');
     }
     When Else
     {
       System. Debug ('Invalid WeekDay Number. Please Enter the value between 1 - 7.');
     }
  }
```

Use Case: Write an apex program, to find out the object name, to which the record is associated based on the "Record ID".

```
            String recordID = '00U5j000007K1dtEAC';

            System.debug('Object Id is....: '+ recordID.Left(3));

            Switch ON recordID.Left(3)

    {

        When '001'

        {

            System. Debug ('This is Account Record.');

        }

        When '003'

        {

            System.debug('This is Contact Record.');

        }

        When '006'

        {

            System.debug('This is Opportunity Record.');

        }

        When '005'

        {

            System.debug('This is User Record.');

        }

        When '00U'

        {

            System.debug('This is Event Record.');

        }

        When '701'

        {

            System.debug('This is Campaign Record.');

        }

        When '00Q'

        {

            System.debug('This is Lead Record.');
        }
        When Else
        {
           System.debug('Invalid Record Id.');
        }
    }
```

Use Case: Write an apex program, to find out the Season Name based on the Month Number as below.

Month Number	Season Name
1 - 4	Winter Season
5 - 8	Summer Season
9 - 12	Spring Season
<1 OR >12	Invalid Month Number.

```
          Integer monthNumber = 14;

          Switch ON monthNumber
    {
        When 1,2,3,4
        {
           System.debug('Winter Season.');
        }
        When 5,6,7,8
        {
           System. Debug ('Summer Season.');
        }
        When 9,10,11,12
        {
           System. Debug ('Spring Season.');
        }
        When Else
        {
            System. Debug ('Invalid Month Number. Please Enter the value between 1 - 12.');
        }
    }
```

ITERATIVE STATEMENTS

Iterative Statements are used to execute the Piece of Code iteratively/repeatedly till the required number of times.

By using Iterative Statements, we can avoid the redundant code i.e., we can make the code generic hence the application performance can be improved.

Apex provides 4 Iterative Statements.

1. While
2. Do-While
3. FOR
4. Enhanced FOR Loop

WHILE STATEMENT

While is a Pre-Checking Iterative Statement, which will execute the statements based on the condition results. i.e. it will check the conditions first, then it will execute the Statements. It will execute the Statements, till the conditions get satisfied.

Syntax:

```
While(<Conditions>)
{
        // Write the Business Logic.
                ....
                ....
}
```

Use Case: Write an apex program, to Print the Welcome Message 5 Times on the Log File.

```
    Integer counter = 1;
            While (counter <= 5)
        {
            System. Debug ('Welcome to Apex Programming.');
            Counter++;
        }
```

Use Case: Write an apex program, to Print the Numerical values between 1- 50.

(Ex: 1 2 3 4 ...... 50)

```
                Integer counter = 1;
                While (counter <= 50)
        {
            System. Debug ('Number Value is...: '+ counter);
            counter++;
        }
```

Use Case: Write an apex program, to Print the Even Numbers between 2 - 100.

(Ex: 2 4 6 8 10 .... 100)

```
                Integer counter = 2;
                While (counter <= 100)
            {
                System. Debug ('Even Number is....: '+ counter);
                counter += 2;
            }
```

Use Case: Write an apex program, to Print the Mathematical Table for the given Number as below.

```
            Ex:     5 * 1 = 5
                    5 * 2 = 10
                    ....
                    ....
                    5 * 10 = 50
```

```
            Integer counter = 1, maxNumber = 2945;
            While (counter <= 20)
        {
            System. Debug (maxNumber + '* ' + counter + '  =  '+ (maxNumber * counter));
             counter++;
        }
```

DO-WHILE STATEMENT

Do-While is a Post-Checking Iterative Statement, which will execute the statements once, and then it will check the conditions. It will execute the Statements repeatedly till the conditions get satisfied. Once the conditions get failed, then it will come out of the block.

Syntax:

```
        Do
        {
                // Write the Statements.
        }
While(<Conditions>);
```

Use Case:

Write an apex program, to Print the Odd Numbers between 1 - 100 in reverse order. (Ex: 99, 97, 95, 93...1)

```
        Integer counter = 99;
        do
      {
         System.debug('Odd Number is.....: '+ counter);
         counter -= 2;
      }
While (counter >= 1);
```

Use Case:

Write an apex program, to Print the Numerical Values between 50 - 100.

```
        Integer counter = 50;
        do
      {
         System.debug('Numerical Value is....: '+ counter);
         counter++;
      }
While(counter <= 100);
```

FOR ITERATIVE STATEMENT

It is a Pre-Checking Iterative Statement, which provides an optimized way of using Iterative Statements.

Syntax: for(<Initialization Part> ; <Conditions Part> ; <Increment / Decrement>)

```
        {
                // Write the Statements
                        ....
                        ....
        }
```

Use Case:

Write an apex program, to Print the Mathematical Table for the given Number as below.

Ex:

```
        5  *  1 = 5

        5  *  2 = 10

        ....

        ....

        5  *  10 = 50
```

```
    for(Integer maxNumber = 5, counter = 1; counter <= 10; counter++ )

  {

     System.debug(maxNumber + '  *  '+  counter + '  =  '+ (maxNumber * counter));

  }
```

ARRAYS

Arrays allows us to store a collection of homogeneous elements. i.e. Arrays stores a group of similar datatype elements inside it. It won't support to store the Heterogeneous Elements.

Arrays are Fixed Collections. Upon defining the array, we have to specify the size.

Arrays support Static Memory Allocation. Hence the size can't be grow/shrink at runtime.

Syntax: <DataType>[] <arrayName> = new <DataType>[<Size>];

```
Ex:     Integer[] customerCodes = new Integer[10];
                                          |
                                          ----> Holds 10 Integer Elements.

        String[] countryNames = new String[100];
                                    |
                                    ----> Holds 100 String elements.

        ID[] recordIds = new ID[50];
                         |
                         ----> Holds 50 Record Id's.

        Account[] accountRecords = new Account[20];
                                          |
                                          ----> Holds 20 Account Records.
        Opportunity[] opptyRecords = new Opportunity[10];
                                                  |
                                                  ----> Holds 10 Opportunity Records.


        Position__C[] positionRecords = new Position__C[50];
                                                  |
                                                  ----> Holds 50 Position Records.
```

Note: Each element inside the array, will get recognized by using an "Index Position", which always starts from "Zero".

Assigning the Values: We can store the values inside the array by using the Index Position as below.

Syntax: <ArrayName>[<IndexPosition>] = <Value>;

```
Ex:     customerCodes[0] = 45001;

        customerCodes[1] = 90023;

        customerCodes[2] = 80021;
```

Retrieving Values: We can fetch the elements from the array, by using Index Position as below.

Syntax: <arrayName>[<indexPosition>];

```
Ex:     System.debug('First Customer Code is...: '+ customerCodes[0]);

        System.debug('Second Customer Code is...: '+ customerCodes[1]);
```

Example 1

```
// Defining the Array

        Integer[] productCodes = new Integer[5];

// Print the Array Size

        System.debug('Array Size is....: '+ productCodes.size());

// Print the Array Elements

        System.debug('Array Elements Are.....: '+ productCodes);

// Assigning the Values

        productCodes[0] = 6002012;

        productCodes[1] = 9004567;

        productCodes[2] = 5003490;

        productCodes[3] = 8005678;

        productCodes[4] = 3006744;

        System.debug('After Assigning Values, Elements are....: '+ productCodes);
```

```
// Print the Elements in Separate Rows
        System.debug('First Element is.....: '+ productCodes[0]);
        System.debug('Second Element is....: '+ productCodes[1]);
        System.debug('Third Element is....: '+ productCodes[2]);
        System.debug('Fourth Element is....: '+ productCodes[3]);
        System.debug('Fifth Element is.....: '+ productCodes[4]);

// Print the Elements by using Iterative Statements
        for(Integer counter = 0; counter < productCodes.Size() ; counter++)
   {
      System.debug('Array Element is....: '+ productCodes[counter]);
   }
// Print the Elements in Reverse order
        System.debug('----------- PRINTING IN REVERSE ORDER --------------');
        for(Integer counter = productCodes.Size()-1; counter >= 0 ; counter--)
   {
      System.debug('Product Code is......: '+ productCodes[counter]);
   }
// Print the Elements by using Enhanced FOR Loop....
        for( Integer pCode :  productCodes )
   {
      System.debug('Product Code is....: '+ pCode);
   }
```

Example 2:

==========

```
// Defining the Array and Initializing the values..
String[] countryNames = new String[]{'India','United States','Germany','Japan','China',

                                        'Middle East','UK','Australia','Srilanka','Bangladesh'};

//Print the Collection Size....
System.debug('Collection Size is....: '+ countryNames.Size());
```

```
// Print the Elements...
        for (String cName : countryNames)
                {
                System.debug('Country Name is....: '+ cName);
                }
```