[Apex HandBook]

WiseQuarter Education

What is Programing Language?

A programming language is a type of written language that tells computers what to do. “Put simply, programming is giving a set of instructions to a computer to execute. If you’ve ever cooked using a recipe before, you can think of yourself as the computer and the recipe’s author as a programmer. The recipe author provides you with a set of instructions which you read and then follow. The more complex the instructions, the more complex the result!”

Apex Programing Language

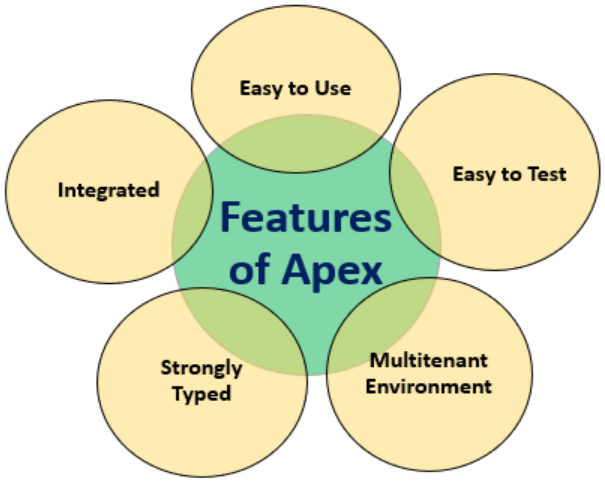
What is APEX?

Apex is a strongly typed programing language means you need to type data types when you create variables.

Apex is a case-insensitive programing language means System and system are same.

Apex is an Object-Oriented Programming Language (OOP). In OOP we will focus on objects. While creating objects we should be careful about feature and functions.

1) Feature (Variaabe) : passive feature called variable (size, color, shape)
2) Functionality (Method): Active feature called functions.



Example: We want to create car object passive feature will be Model, Make, Wheel, Body, etc. and active feature will be other features, like Reversing Assistant, Blind-Spot Information System, Adaptive Cruise Control, Night Vision Assist etc.

After creating all objects, we will put them together to have application. The benefit of OOP is consisted of small parts. Since it formed by small part error will be less.

When Should I Use Apex?

The Salesforce prebuilt applications provide powerful CRM functionality. In addition, Salesforce provides the ability to customize the prebuilt applications to fit your organization. However, your organization may have complex business processes that are unsupported by the existing functionality. In this case, Lightning Platform provides various ways for advanced administrators and developers to build custom functionality.

Use Apex if you want to:

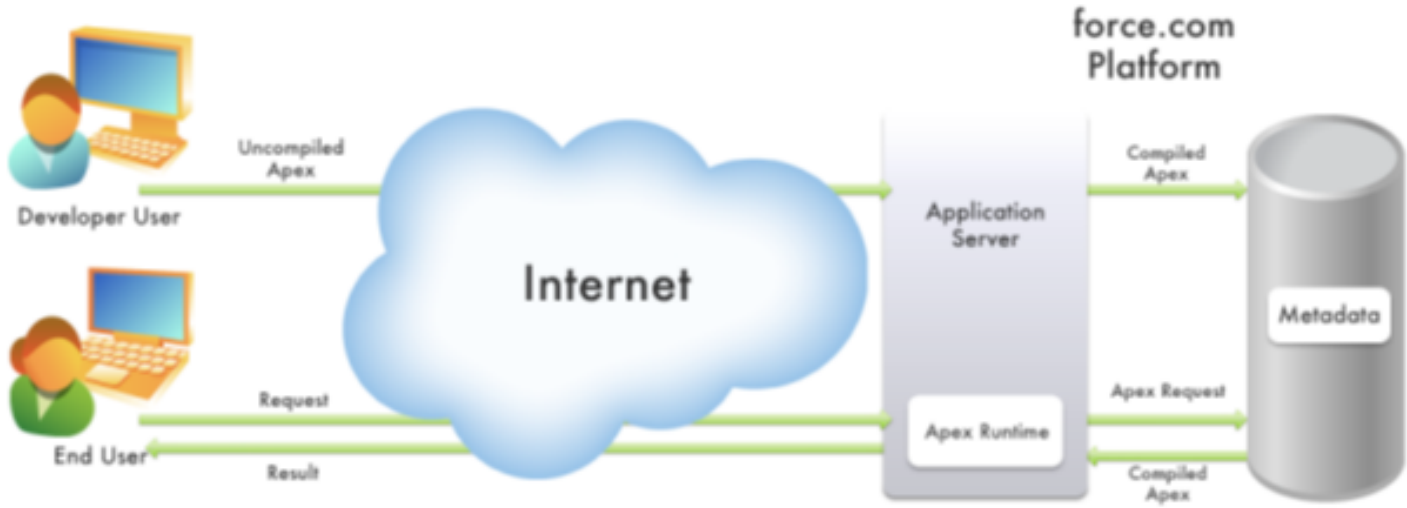
Create Web services.

Create email services.

Perform complex validation over multiple objects.
Create complex business processes that are not supported by workflow.
Create custom transactional logic (logic that occurs over the entire transaction, not just with a single record or object).
Attach custom logic to another operation, such as saving a record, so that it occurs whenever the operation is executed, regardless of whether it originates in the user interface, a Visualforce page, or from SOAP API.

How Does Apex Work?

All Apex runs entirely on-demand on the Lightning Platform. Developers write and save Apex code to the platform, and end users trigger the execution of the Apex code via the user interface.
Apex is compiled, stored, and run entirely on the Lightning Platform



When a developer writes and saves Apex code to the platform, the platform application server first compiles the code into an abstract set of instructions that can be understood by the Apex runtime interpreter, and then saves those instructions as metadata.

When an end user triggers the execution of Apex, perhaps by clicking a button or accessing a Visualforce page, the platform application server retrieves the compiled instructions from the metadata and sends them through the runtime interpreter before returning the result. The end user observes no differences in execution time from standard platform requests.
Statements: Instructions can also be called commands but are usually referred to as statements.
Different languages have differing syntax, but in many languages, the syntax for a statement is to terminate (end) each statement with a semicolon (;).
Sample Statements:
String firstName = 'Mike';

Date todaysDate;
Integer five = 5;

Comments: Single-line comments are marked by using two forward slashes //. Multiline comments are marked by using a forward slash and asterisk /* to open the comment block, and closing the block is done with an asterisk followed by a forward slash */.

Sample of Comments:

```
/*This is an example of a multiline comment block.
Notice that only the first
and last lines have the markings.
Multiline comments can also be used to comment out sections of code, by
turning them into a comment
   <commented code here>
*/
String firstName = 'Mike'; // single-line comments can be made after
 //statements on the same line
//they can also be made on separate lines
```

Class: Classes are templets to create an object. In real life we call class templets. Everything in object must be in class. In class we will have variables and functions.

Data: Data is information processed or stored by a computer. Everything Apex uses or everything Apex produces is data.

Bit: A bit is the smallest unit of data in a computer. A bit has a single binary value, either 0 or 1. Note: 8 bits are called 1 byte.

VARIABLES in APEX

Variables is the name of reserved area allocated in memory. In other words, it is a name of memory location. A variable is a container which holds the value while the java program is executed. Null are empty object in Apex.

1. The Apex language is strongly typed so every variable in Apex will be declared with the specific data type.
2. All apex variables are initialized to null initially.
3. It is always recommended for a developer to make sure that proper values are assigned to the variables. Otherwise, such variables when used, will throw null pointer exceptions or any unhandled exceptions

4. Apex supports the following data types

a) **Primitive** (Integer, Decimal, Double, Long, Date, Datetime, String, ID, or Boolean)

b) **Collections** (Lists, Sets and Maps)

c) **sObject**

d) **Enums**

e) **Classes, Objects and Interfaces**

In modern programming languages, words can be used to make it easy to understand what value is stored in a given variable. For example, if you were writing a form, you might store the value of the person's first name in a variable called firstName. Likewise, you could store their last name in lastName, email in email, date of birth in birthday, and so on. Storing a value in a variable is done by assigning the value, and assignment is done using the equals sign (=). For example, using the assignment statement, firstName = 'Mike';.

Variable Declaration:

```
Data Type variableName;
Integer age;
String name;
```

Note: if you will declare more than one variable with the same data type:

```
Data Type variableNames:
Integer age, height, distance;
Example: Create 3 Integer variables and assign values to them

//1.Way:
Integer num1 = 12;
Integer num2 = 13;
Integer num3 = 23;
//2. Way: Recommended
Integer num1 = 12, num2 = 13, num3 = 23;
System.debug(num1);//12
System.debug(num2);//13
System.debug(num3);//23
```

Tip naming your variables in such a manner that is easy to understand is beneficial to everyone. It allows you to develop quickly without pausing to remember what is stored in a given variable, and it also allows other developers to understand your code faster. using a variable name of asdf to store the value of someone's first name from a form would be rather cryptic.

camelCase: After first word all other start with upper case letter called camelCase.

Primitive Data Types

Primitive data types are basic pieces of a programming language that include built-in support. In most languages, these types cannot be modified and are used as basic building blocks to construct application logic. The primitive data types in Apex are as follows:

Data Type	Description and Example
Integer	A positive or negative number that doesn't have a decimal point. `Integer num = 12;`
Decimal	A positive or negative number that has a decimal point. `Decimal num = 12.22222;`
String	A series of characters surrounded by single quotes. This can include any text as short as one letter to sentences. `String whatAmI = 'A String';`
Boolean	Typically either true or false. In Apex, null (empty) is also a valid value. Boolean is commonly used with checkboxes. `Boolean doSomething = False;`
ID (Salesforce ID)	Any valid 18-character Salesforce record ID. `Id contactId = '00300000003T2PGAA0';`

1. **Integer:** A 32-bit number that does not include any decimal point which can range in value from -2,147,483,648 through 2,147,483,647.

Example1: Create an integer variable for your age and print it on console like "My age is 31".

Integer age = 31;

System.debug('My age is ' + age); ➔ *My age is 31*

Example2: Create 3 integer variables and name them as num1, num2, num3, then multiply the sum of n1 and n3 by n2.

```
//1.Way:
Integer num1 = 10;
Integer num2 = 12;
Integer num3 = 8;
//2. Way: Recommended
Integer num1 = 10, num2 = 12, num3 = 8;
System.debug('The Result ' + ((num1+num3)*n2));
```

2. **Double:** A very large 64-bit number which can include a decimal, with a minimum value of -2^{63} and maximum value of $2^{63}-1$. Should only be used for very large numbers and complex calculations.

Example1: Calculate the area of circle

Double pi = 3.14;

Double r = 2.5;

System.debug('The area of a circle:' + pi*r*r); ➔

Note: If you use "Double" as data type you get result everytime as decimal. For example, for 25 you will get 25.0

Example2:

```
Double priceOfBook = 12;
System.debug('The Book price is ' + priceOfBook);
```

3. **Decimal:** A number with a decimal point. The default type for Currency fields.

Example1: Create 2 decimal variables for eggs and milk with corresponding price then print total value on the console.

Decimal eggs = 4.38, milk =1.98;

System.debug (' Total Value is ' + (eggs+milk)); // Way1 not recommended

Decimal totalValue= eggs + milk;

System.debug('Total Value is ' + totalValue);// Way2 Best practice

4. **Long:** This is a 64-bit number without decimal point; it's the same as an integer but can be used to represent much larger numbers than an integer. Use this if you need to represent a number outside the range of integers only. Long values can range from -2^{63} through $2^{63}-1$ (that is a really big number!).

Example1:

Long googolplex = 10000000000000000000000000L ➔ (1 x 10 ^10 ^100) ➔ Even long will not save that number since its too long.

```
Long populationOFTheWorld= 7000000000L;

System.debug (populationOFTheWorld);
```

Note1: Put "L" or "l" to the end of the long vale

Note2: When you select "Long" as Data Type for the values in the Integer Range Apex will accept it as Integer, even you typed "Long" as Data Type. Apex will do that one to save memory. Because Integers use memory less than Longs.

Example2: Create a Long variable that in the range of the Integer.

```
Long num= 100000L;

System.debug(num);
```

5. **Boolean:** variables of this type can only be assigned as **true**, **false,** or **null** (see Null).

Null: Although null is technically not a primitive type per se, it is a special constant value that is used to represent **"nothing".** All variables declared, but not assigned a value, have the value of null assigned automatically.

Number Variables Practices

Practice1: Create two integer variables and print sum on the console with a label.

```
Integer int1= 12;

Ineteger int2 = 24;

System.debug('The sum of int1 and int2 is '+ (a+b));
```

Practice2: Create three decimal variables for three items that you bought recently with their price then print total that you paid for them.

```
Decimal juice = 2.89;
Decimal meat = 6.49;
Decimal bread = 1.49;
System.debug('Total of the items value is ' + (juice+meat+bread)); ➔
```

Practice3: Type a code to calculate a simple interest for 2500 of principal, 5 years and rate of 3.99. (simple Interest formula is (principal*rate*numOfYears/100)

```
Decimal principal = 2500;
Decimal rate = 3.99;
Integer numOfYear = 5;
System.debug('The Simple Interest is '+ (principal* rate* numOfYear/100));
```

Note: If you use multiple Data Types in an operation, result data type will be in the largest Data Type as above practice.

Practice4: Create 2 Decimal variables for the prices of laptop and TV then print the total value for 2 laptops and 3 TVs with a label

```
Decimal laptopPrice = 849;
Decimal tvPrice = 350;
//1.Way: Not recommended
System.debug('Total Price is ' + (2*laptopPrice + 3*tvPrice));//Total Price is 4652.00

//2.Way:
Decimal totalPrice = 2*laptopPrice + 3*tvPrice;
System.debug('Total Price is ' + totalPrice);
```

6. **ID:** Any valid 18-character Force.com record ID. To find id of the record go to object of your selection then click on the record that you want to see its id ➔ In the URL you will see the id of the record:

➔https://d5f000005vrf7eae-dev-ed.lightning.force.com/lightning/r/Account/**0015f00000AzqzAAAR**/view

Example:

```
ID id='0015f00000AzqzAAAR';
System.debug( ' The value of ID ' + id)
```

7. **String:** Any set of characters, such as a word, surrounded by quotes. String is arguably the most versatile type (other than Object) as it can be cast to most other types and is easy to store and manipulate. Strings have no limit on a maximum length, but the Heap Size Limit is used to control the program's size.

Example:

```
String str='  I love Coding. ';
System.debug( ' The value of str is  ' + str);

String str1=' ';
System.debug( ' The value of str is  ' + str1);// The value of String str is

String str2=null;
System.debug( ' The value of str is  ' + str2); // The value of String str is
null
```

```
Note: if you want to print Apostrophe (') symbol in your string
//you can negatete its meaning by backward slash (\).
String stdName = ' Ahmet Balli \'Age\' is  30';
System.debug(stdName);//Ahmet Balli 'Age' is  30
System.debug('Ahmet\'s performance at Apex has improved' );
//Ahmet's performance at Apex has improved
```

String Class Methods in Apex

1) **capitalize():** Returns the current String with the first letter changed to uppercase. It does not touch other charcters.

Example: Convert first character of the given string to uppercase

```
String str = 'hello Everyone';
String str2 = str.capitalize();
System.debug(str);// hello Everyone
System.debug(str2);//Hello Everyone ==> Only first character will be
capitalized
```

2) **toLowerCase():** Converts all of the characters in the String to lowercase using the rules of the default locale.

Example: Convert all characters of the given string to lower case.

```
String str= 'HELLO WORLD';
String str2 = str.toLowerCase();
System.debug(s2); //hello world==> All characters will be lowercased
```

3) **toUpperCase():** Converts all of the characters in the String to uppercase using the rules of the default locale.

Example: Convert all characters of the given string to upper-case.

```
String str= 'Wisequarter Education LLC';
```

```
String str2 = str.toLowerCase();
System.debug(s2);  //WISEQUARTER EDUCATION LLC==> All characters will be uppercase
```

4) **trim() :** Returns a copy of the string that no longer contains any leading or trailing white space characters. User can put space character at the beginning and at the end of the string by mistake. We use trim() method to remove those kind of space characters.

Example: Remove all spaces from beginning and end of the given string.

```
String str= ' Enjoy Coding    ';
String str2 = str.trim();
System.debug(s2);  //Enjoy Coding==> All space characters removed from beginning and end of the string.
```

5) **Length():** That method will count the number of characters that used in string.

Example: Count numbers of characters that used in given string.

```
 String str= 'Enjoy Coding';
Integer numOFChars = str.length();
System.debug(int); //12
```

6) **Split(‘ ’):** Returns a list that contains each substring of the String that is terminated by either the regular expression regExp or the end of the String.

Example1: Divede given string by space.

```
String str = 'I love coding with Dervis';
List <String> listOFWord = str.split(' ');
System.debug(listOfWord); // (I, love, coding, with, Dervis)
```

Example2: Divede given string by space and get the first word of it.

```
//Way1:
String str = 'Lets do some practice';
List <String> listOFWord = str.split(' ');
String firstWord = listOFWord[0];
System.debug (firstWord); //Lets

//Way2:
String str = 'Lets do some practice';
String firstWord = = str.split(' ')[0];
```

System.debug (firstWord); //Lets

Example3: Check if the user put his name and surname in correct format.

There should not be any space in front of name
There shouldn't be any space at the end of the surname
There should be only one space between name and surname
Only the first letter of name and surname should be in Uppercase

```
String str = '  JACK Jones   ';
String trimmedStr= str.trim();
System.debug(trimmedStr); //JACK jones
//1. Way:
//Fixed first name
String firstName = split(' ')[0];
System.debug (firstWord); //Jack
//Firstname format fixed
String firstName = s.split(' ')[0];
String lowerCasedFirstName = firstName.toLowerCase();
String capitalizedLowerCasedFirstName = lowerCasedFirstName.capitalize();
//Lastname format should be fixed as well
String lastName = s.split(' ')[1];
String lowerCasedLastName = lastName.toLowerCase();
String capitalizedlowerCasedLastName = lowerCasedLastName.capitalize();
System.debug(capitalizedLowerCasedFirstName + ' ' + capitalizedlowerCasedLastName);
//2.Way:
String expectedFormatOfFirstName = s.split(' ')[0].toLowerCase().capitalize();
String expectedFormatOfLastName = s.split(' ')[1].toLowerCase().capitalize();
System.debug(expectedFormatOfFirstName + ' ' + expectedFormatOfLastName);
```

7) deleteWhiteSpace(): To remove all spaces from string include the white spaces in middle use that method.

Example: Type code to count the number of characters except spaces in your name.

```
String str = '   Dervis   Ozay   ';
Integer noSpaceStr = s.deleteWhiteSpace().length();
System.debug('Your name has ' + noSpaceStr + ' characters');
```

8) indexOf(): to get the index of any character use indexOf() method. That will give the index of first occurrence.

Note: index start counting at 0. I mean the index of first character is 0.

Example: Find the index of @ in the email address.

```
String email = 'dozaywisequarter@gmail.com'
Ineteger indx = email.indexOf('@');
System.debug(indx);
```

9) **substring(startIndex):** Returns a new String that begins with the character at the specified zero-based startIndex and extends to the end of the String. Also, **substring(startIndex, endIndex)** Returns a new String that begins with the character at the specified zero-based startIndex and extends to the character at endIndex - 1.

Example1: Create another string from your name at the 3rd index of it.

```
String name = 'Dervis';
String subName = srt.substring(3);
System.debug(subStr)//vis
```

Example2: Write a code to create substring from given string at 1 – 5 indexes.

```
String str = 'smile';
String subStr = srt.substring(1,5);
System.debug(subStr)//mile
```

Example3: Extract what kind of email your user is using (is it gmail, Hotmail, Yahoo etc.)

```
//To get the company name I will use substring() method
//To be able to use substring() method I need index of first character of the
company name
//index of '@' + 1
//To be able to find the index of last character of the company name use ➔index of
'.' – 1
String myEmail = 'dozaywisequarter@gmail.com';
Integer startingIndex = myEmail.indexOf('@')+1;
Integer endingIndex = myEmail.indexOf('.');
String companyName = myEmail.substring(startingIndex, endingIndex);
System.debug(companyName); //gmail
```

10) remove(substring): Removes all occurrences of the specified substring and returns the String result.

Example1: Extract what kind of email your user is using (is it gmail, Hotmail, Yahoo etc.)

```
String myEmail = 'dozaywisequarter@gmail.com';
String companyName = myEmail.split('@') [1].remove('.com');
System.debug(companyName); //gmail
```

Example2:

```
String s1 = 'Salesforce and force.com';
String s2 = s1.remove('force');
System.assertEquals('Sales and .com', s2); // that will compare two string to check
if they equal or not.
```

11) **lastIndexOf():** Returns the index of last occurrence of any character or characters.

Example: Check the index first occurrence and last occurance of force in following string.

```
String s1 = 'Salesforce and force.com';
Integer firstIndex = s1.indexOf('force');
Integer lastIndex = s1.lastIndex('force');
System.debug(firstIndex);
System.debug(lastIndex);
```

Note: If you get '-1' from indexOf() or lastIndexOf() methods, it means the character/s you are looking for does not exist inside the String.

Example:

```
String str = 'Salesforce and force.com';
Integer indx = str.index('xyz');
System.debug(indx); //-1
```

12) contains () : Check if a specific character or characters exists in a String or not. Will return Boolean value (true or false)

Example: Check if str1 contain str2

```
String str1 = 'Apex is easy';
String str2 = 'Apex';
Boolean str1ContainStr2 = str1.contains(str2);
System.debug(str1ContainStr2); //True
```

13) containsIgnoreCase(substring): Returns true if the current String contains the specified sequence of characters without regard to case; otherwise, returns false.

Example1:

```
String str = 'hello';
Boolean bln = s.containsIgnoreCase('HE');
System.debug(b); //True
```

Example2: check if a word exists in a string ignoring cases

```
String str = 'Apex is fun, Enjoying apex class';
Boolean bln = s.containsIgnoreCase('APEX');
System.debug(b); //True
```

Example3:

```
String s1 = 'Albania';
String s2 = 'ban';
System.debug(s1.contains(s2)); // true
```

Example4:

```
String s = 'ApeX is good';
System.debug(s.containsIgnoreCase(GOOD)); //true
System.debug(s.containsIgnoreCase(apex)); // true
```

14) containsNone(secondString): Returns true if the current String doesn't contain any of the characters in the specified String; otherwise, returns false.

Example:

```
String s = 'Apex is good';
System.debug(s.containsNone('manE')); // true
```

15) containsWhiteSpace(secondString): Returns true if the current String contains any white space characters; otherwise, returns false.

Example:

```
String s1 = 'Apex is good';
String s2 = 'Apex ';
String s3 = ' Apex ';
String s4 = 'Apex';
System.debug(s1.containsWhiteSpace()); //true
System.debug(s2.containsWhiteSpace()); //true
System.debug(s3.containsWhiteSpace()); // true
System.debug(s4.containsWhiteSpace()); // false
```

16) equals(string): This method is used to compare 2 Strings and return Boolean data type. Note: Use this method to perform case-sensitive comparisons.

Example2: Check if given 2 strings are equal each other or not.

```
String str1 = 'Apex';
String str2 = 'APEX';
Boolean result = str1.equals(str2);
System.debug(result); // False
```

Example2:

```
String str1 = 'Apex is apex';
String str2 = 'Apex is APEX';
String str3 = '';
String str4 = '';
String str5 = '';
String str6 = null;
```

```
boolean b1 = str1.equals(str2);
boolean b2 = str3.equals(str4);
boolean b3 = str5.equals(str6);
System.debug(b1); // false
System.debug(b2); //true
System.debug(b3); // false
```

17) equalsIgnoreCase(string): If you want to check equality by ignoring cases use that method

Note: Use this method to perform case-sensitive comparisons.

Example:

```
String str1 = 'Apex is apex';
String str2 = 'Apex is APEX';
boolean b1 = str1.equalsIgnoreCase(str2);
System.debug(b1); // true
```

18) replace(target, replacement): Replaces each substring of a string that matches the literal target sequence target with the specified literal replacement sequence replacement.

Example: Change a specific character to another one.

```
String str1 = 'Apex is Apex';
String resultStr = str1.replace('Apex','Java');
System.debug(resultStr); //Java is Java
```

Note1: If you use ''(nothing) in the second parameter, it means first character will be removed

Note2: Instead many characters you can put a single or more characters.

Note3: There is no any restriction about the number of characters

Example2: Remove 'ex' from string

```
String str1 = 'Apex is Apex';
String resultStr = str1.replace('ex','');
System.debug(resultStr); //Ap is Ap
```

Example3: Remove spaces from given string.

```
String str1 = 'Apex is Apex';
String resultStr = str1.replace(' ','');
System.debug(resultStr); //ApexisApex
```

19) ValueOf(): Return string that represent any other data type.

Example1: Convert following integers to string and combine them with a single space between them.

```
Integer int1 = 12;
Integer int2 = 22;
String str1 = String.valueOf(int1);
String str2 = String.valueOf(int2);
System.debug(str1 +' '+str2);
```

Example2: Find the total price of book and pen which given in string format.

```
String bookPrice = $2.99;
String penPrice = $0.49;
String book= bookPrice.replace('$','');
String pen= penPrice.replace('$','');
Decimal priceOfBook = Decimal.valueOf(book);
Decimal priceOfPen = Decimal.valueOf(pen);
System.debug('The total price is '+ (priceOfBook + priceOfPen));
```

Example3: Remove all letter characters from a given string (both lower and upper case)

```
String password = 'ab123FE%*/67C';
String str =
password.replace('a','').replace('b','').replace('F','').replace('E','').
replace('C','');
System.debug(str); //123%*/67
```

➔ The above practice not recommended for that we can use **replaceAll()** method with **Regex.**

20) replaceAll(regExp, Replacement); Replaces each substring of a string that matches the regular expression regExp with the replacement sequence replacement.

➔ Regex is used to get a group of data.

Example1: Remove all upper-case letter from a given string

```
String password = 'ab123FE%*/67C';
String noUpperCase =password.replaceAll('[A-Z]', '');
System.debug(noUpperCase);
```

Example2: Remove all lower-case letter from a given string

```
String password = 'ab123FE%*/67C';
String nolowerCase =password.replaceAll('[a-z]', '');
System.debug(noLowerCase);
```

Example3: Remove all lower-case and upper cases letter from a given string

```
String password = 'ab123FE%*/67C';
String noLetter =password.replaceAll('[A-Za-z]', '');
System.debug(noLetter);
```

Example4: Remove all number characters from a given string

```
String password = 'ab123FE%*/67C';
String nolowerCase =password.replaceAll('[0-9]', '');
System.debug(noLowerCase);
```

Example5: Remove all characters except upper-case letters from given string

```
String password = 'ab123FE%*/67C';
String onlyUpperCases =password.replaceAll('[^A-Z]', '');
System.debug(onlyUpperCases);
```

Example6: Remove all characters except lower-case letters from given string

```
String password = 'ab123FE%*/67C';
String onlyLowerCases =password.replaceAll('[^a-z]', '');
System.debug(onlyLowerCases);
```

Example7: Remove all characters except numbers letters from given string

```
String password = 'ab123FE%*/67C';
String onlyNumber=password.replaceAll('[^0-9]', '');
System.debug(onlyNumber);
```

Example8: Keep only symbols in given string

```
String password = 'ab123FE%*/67C';
String onlyNumber=password.replaceAll('[A-Za-z0-9]', '');
System.debug(onlyNumber);
```

Example9: Remove all characters different from alphabetical characters from a String.

```
String password = 'ab123FE%*/67C';
String onlyLetter=password.replaceAll('[^A-Za-z]', '');
System.debug(onlyLetter);
```

Example10: Remove all characters different from ‘a’ and 1 from a String.

```
String password = 'ab123FE%*/67C';
String newStr=password.replaceAll('[^a1]', '');
System.debug(newStr);
```

Example11: Change all character of password to * .

```
String password = 'ab123FE%*/67C';
```

```
String asterix=password.replaceAll('[^ ]', '*'); //* has one space on the right
side
System.debug(asterix);
```

Regular Expressions Table:

Character classes	
`[abc]`	a, b, or c (simple class)
`[^abc]`	Any character except a, b, or c (negation)
`[a-zA-Z]`	a through z or A through Z, inclusive (range)
`[a-d[m-p]]`	a through d, or m through p: `[a-dm-p]` (union)
`[a-z&&[def]]`	d, e, or f (intersection)
`[a-z&&[^bc]]`	a through z, except for b and c: `[ad-z]` (subtraction)
`[a-z&&[^m-p]]`	a through z, and not m through p: `[a-lq-z]` (subtraction)

Predefined character classes	
`.`	Any character (may or may not match line terminators)
`\d`	A digit: `[0-9]`
`\D`	A non-digit: `[^0-9]`
`\s`	A whitespace character: `[ \t\n\x0B\f\r]`
`\S`	A non-whitespace character: `[^\s]`
`\w`	A word character: `[a-zA-Z_0-9]`
`\W`	A non-word character: `[^\w]`

Practice: Check a given password by using following rules.

1)It has at least 10 characters

2)It has at least 1 uppercase

3)It has at least 1 lowercase

4)It has at least 1 digit

5)It has at least 1 symbol

```
String pwd = 'Ab429*l$8910';

1)It has at least 10 characters
Boolean length = pwd.length()>=10;

2)It has at least 1 uppercase
Boolean upper = pwd.replaceAll('[^A-Z]','').length()>0;

3)It has at least 1 lowercase
Boolean lower = pwd.replaceAll('[^a-z]','').length()>0;

4)It has at least 1 digit
Boolean digit = pwd.replaceAll('[^0-9]','').length()>0;
```

```
5)It has at least 1 symbol
Boolean symbol = pwd.replaceAll('[A-Za-z0-9]','').length()>0;

System.debug(length && upper && lower && digit && symbol);
```

21) chartAt(index): Returns the **ASCII** value of the character at the specified index.

Note: ASCII abbreviated from **American Standard Code for Information Interchange**, is a character encoding standard for electronic communication. **ASCII** codes represent text in computers, telecommunications equipment, and other devices.

Example1: Find the sum of ASCII values of first and second characters in a string.

```
String str = 'I love Apex';
Integer asciiOfFirst = s.charAt(0);
Integer asciiOfSecond = s.charAt(1);
System.debug(asciiOfFirst + asciiOfSecond);
```

Example2: Find ASCII value of third and last character of given string.

```
String str = 'Apex is easy';
Integer asciiOfThird = s.charAt(3);
Integer asciiOfLast = s.charAt(str.length()-1);
System.debug(asciiOfThird + asciiOfLast);
```

Example3: Check if string starting with 'A' and end with 'x'; ASCII value of A = 65, ASCII value of x = 120
Apex ➔ starting with A, end with x ➔True
Java ➔ False

```
String str = 'Apex';
Integer asciiOfFirst= s.charAt(0);
Integer asciiOfLast = s.charAt(str.length()-1);
System.debug(asciiOfThird==65 && asciiOfLast==120);
```

Note: The above method not recommended Apex have method for that.

22) endsWith(suffix): Returns true if the String that called the method ends with the specified suffix.
23) startsWith(prefix): Returns true if the String that called the method begins with the specified prefix

Example1: Check if string starting with 'A' and end with 'x';

```
String str = 'Apex';
Boolean first = str.startsWith('A');
Boolean end= str.endssWith('j');
System.debug(first && end);
```

Example2:

```
String str = 'Apex is Fun';
Boolean b1 = str.startsWith('Apex');
System.debug(b1) //True
Boolean b2= str.endssWith('Fun');
System.debug(b1) //True
Boolean b3= str.endssWith('ap');
System.debug(b3) //False
Boolean b4= str.endssWith('fUN');
System.debug(b4) //False
```

24) **endsWithIgnoreCase(suffix):** Returns true if the String that called the method ends with the specified suffix.

25) **startsWithIgnoreCase(prefix):** Returns true if the String that called the method begins with the specified prefix

Example 3: Check if a String is starting and ending with 'A' by ignoring cases

```
String str = 'Apex is Fun';
Boolean b1 = str.startsWithIgnoreCase('aPEX');
System.debug(b1) //True
Boolean b2= str.endssWithIgnoreCase('UN');
System.debug(b2) //True
```

26) **difference(secondString):** Returns the difference between the current String and the specified String.(it will start get the difference from the first different character).

Example:

```
String s1 = 'Apex is language';
String s2 = 'Apex is easy';
String diff1 = s1.difference(s2);
String diff2 = s2.difference(s1);
System.debug(diff1); //language
System.debug(diff2); //easy
```

27) **countMatches(secondString):** Returns the number of times the specified substring occurs in the current String.

Example:

```
String s = 'Apex is Apex';
Integer count = s.countMatches('Apex');
System.debug(count); //2
```

28) compareTo(secondString): Compares two strings lexicographically, based on the Unicode value of each character in the Strings

```
String s1 = 'Alabama';
String s2 = 'Alabama';
String s3 = 'alabama';
System.debug(s1.compareTo(s2)); // 0
System.debug(s2.compareTo(s3)); // Returns -32 ‘-’ means s2 is before s3, 32 means ASCII value of A - ASCII value of a = 65-97 = -32
System.debug(s3.compareTo(s2)); //32
```

FYI: You can reach full list of string method by following link: https://developer.salesforce.com/docs/atlas.en-us.apexref.meta/apexref/apex_methods_system_string.htm

8. **Date :** represents a date and contains no time data. For saving the date along with time, we will need to store it in variable of DateTime.
1. First way of creating date variable in Apex is using today() method.

Example: Get current date

```
Date todayDate = date.today(); // will create date of today
System.debug('Today Date is ' + todayDate ); // Today Date is: 2022-11-28 00:00:00
```

2. Second way to create date variable in Apex is to use newInstance(Y, M, D) method.

Example: Create variable called lastClassDate.

```
Date lastClassDate = date.newInstance(2022, 09, 03); // that method will create date
System.debug('Last class date is ' + lastClassDate);
```

Date Class Methods:

1) **Year(), month(), and day() method:** You can get year, month or day of given date variable by using these methods.

Example: Get year, month and day of current date.

```
Date todayDate = date.today(); // will create date of today
Integer currentYear = todayDate.Year();
```

```
System.debug(currentYear);
Integer currentMonth = todayDate.Month();
System.debug(currentMonth);
Integer currentDay = todayDate.Day();
System.debug(currentDay);
```

Note: Some methods can be used with class name. Like today() method, we used today() with date class to get today date. This kind of method called **static method.**
Some methods can be used with variable name. Like year(), month(), day() methods in previous example. This kind of method called **non-static methods**.

2) **addDays(), addMonths(), and addYears() Method:** We use these method to add year, months and days to a date variable.

Example1:

```
Date myDate = date.newInstance(1990, 11, 21); // date.newInstance() method
will create date.
Date actualDate = myDate.addDays(3);
Date expectedDate = date.newInstance(1990, 11, 24);
System.debug('Actual date is ' + actualDate); //Actual date is 1990-11-24
System.debug('Expected date is ' + expectedDate); //Expected date is 1990-
11-24
System.assertEquals(expectedDate, actualDate);// ==> will check if
expectedDate and actualDate are same.
```

```
Example2:
Date myDate = date.newInstance(1990, 11, 21);
Date actualDate = myDate.addMonths(3);
Date expectedDate = date.newInstance(1991, 02, 21);
System.debug('Actual date is ' + actualDate); // Actual Date is 1991-02-21
00:00:00
System.debug('Expected date is ' + expectedDate); //Expected date is 1991-
02-21 00:00:00
System.assertEquals(expectedDate, actualDate); //==>That will check if
expected and actual date are same.
```

```
Example3:
Date myDate = date.newInstance(1990, 11, 21);
Date actualDate = myDate.addYears(3);
Date expectedDate = date.newInstance(1993, 11, 21);
```

```
System.debug('Actual date is ' + actualDate); // Actual Date is 1993-11-21
00:00:00
System.debug('Expected date is ' + expectedDate); //Expected date is 1993-
11-21 00:00:00
System.assertEquals(expectedDate, actualDate); //==> That will check if
expected and actual date are same.
```

Example4: Find the date 2 years, 3 months, and 20 days after the current date

```
Date todayDate = date.today();
Date futureDate = todayDate.addYears(2).addMonths(3).addDays(20);
System.debug(futureDate);
```

Example5: Find the date 5 years, 6 months, and 15 days before the current date

```
Date todayDate = date.today();
Date pastDate = todayDate.addYears(-5).addMonths(-6).addDays(-15);
System.debug(pastDate);
```

3) day() : Returns the day-of-month component of a Date

1.Way:

```
Date myDate = date.newInstance(1990, 11, 21);
Integer numOfDayInMonth = myDate.day(); //this code will pull number of days in
myDate which is 21.
System.debug('The number of day in a month is ' + numOfDayInMonth); //The number of
day in a month is 21
```

2.Way:

```
Date currentDate = Date.today();
Integer numOfDayInMonth = currentDate.day();
System.debug('The number of day in a month is ' + numOfDayInMonth); //The number of
day in a month is 28
```

4) month() : Returns the month component of a Date

```
Date myDate = Date.today();
Integer currentMonth = myDate.month();
System.debug('The current month is ' + currentMonth); // The Current month is 11
System.assertEquals(08, currentMonth);
```

5) dayOfYear() : Returns the day-of-year component of a Date

1.Way:

```
Date myDate = date.newInstance(1990, 11, 21);
Integer numOfDayInYear = myDate.dayOfYear();
System.debug('The number of day in a year is ' + numOfDayInYear); // The
number of day in a year is 325

2.Way:
Date currentDate = Date.today();
Integer numOfDayInYear = currentDate.dayOfYear();
System.debug('The number of day in a year is ' + numOfDayInYear); // The
number of day in a year is 332
```

6) **daysInMonth()** : Returns the number of days in the month for the specified year and month

Example:

```
Integer numberDays = date.daysInMonth(1960, 2);
System.debug('The number of days in 1960 on February is ' + numberDays); //
The number of days in 1960 on February is 29
```

7) **daysBetween() :** Returns the number of days between the Date that called the method and the specified date

Example:

```
Date dob = Date.newInstance(2008, 1, 1);
Date currentDate = Date.today();
Integer numberDays = dob.daysBetween(currentDate);
System.debug('Lived days: ' + numberDays); // Lived days: 5080
```

Example2 : Create a Date Value for your birth date and create Date Value for one of your friend birth date then calculate the difference in days.

```
Date myBirth = Date.NewInstance (1990,04,15);
Date myFirendBirth = Date.NewInstance (1991,08,15);
Integer numberOfDaysBetween = myBirth.daysBetween(myFirendBirth);
System.debug(numberOfDaysBetween);
```

Note: When you use daysBetween() method, use the smaller date first otherwise you will gate negative number.

8) **monthsBetween() :** Returns the number of months between the Date that called the method and the specified date, ignoring the difference in days.

Example:

```
Date dob = Date.newInstance(2006, 12, 2);
Date currentDate = Date.today();
Integer ageInMonths = dob.monthsBetween(currentDate);
System.debug('Age in months is ' + ageInMonths); // Age in months is 179
System.assertEquals(179, ageInMonths); ==> Do assertion after learning the
num of the month
```

9) **isSameDay() :** Returns true if the Date that called the method is the same as the specified date.

Example:

```
Date myDate = Date.today();
Date dueDate = Date.newInstance(2021, 8, 26);
Boolean dueNow = myDate.isSameDay(dueDate);//Will give true or false
System.debug('Are the dates same? ' + dueNow); // Are the dates same false
```

Example2:

- Find the date 20 years and 5 month after 1990-8-25
- Find the date 15 years and 9 months before 2010-9-29
- Then check if the first and second date is same.

```
Date firstDate = Date.newInstance(1990, 8, 25).addYears(20).addMonths(5);
System.debug(firstDate);

Date secondDate = Date.newInstance(2010, 9, 29).addYears(-15).addMonths(-9);
System.debug(secondDate);

Boolean isSame = firstDate.isSameDay(secondDate);
System.debug(isSame);//False
```

Example3: Get the last 2 digit of the year from today date.

```
Date todayDate = Date.today();
Integer yearOfDate = todayDate.Year();
```

Note: To get to get remainder from a division apex have a static method in math class called mod().

```
Ineteger lastTwoDigit = Math.mod(yearOfDate, 100);
System.debug(lastTwoDigit);
```

Example: Replace numbers of a year in a given date with *.

For Example: 06/30/2022 ➔ 06/30/****

```
Date myDate = Date.newInstance(2022, 06, 30);
Integer month= myDate.Month();
Integer day= myDate.Day ();
String newDate= ‘0’ + month + ‘/’ + day + ‘/****’;
System.debug(newDate); //This is very long way not recommended.
```

10) format() method: converts date variable to string data type. It removes the time part and it displays the date in local date format. USA: MM/DD/YYYY TR: DD/MM/YYYY

Example:

```
Date myDate = Date.newInstance(2022, 06, 30);
System.debug(myDate);// 2022-06-30 00:00:00

String stringDate = myDate.format();
System.debug(stringDate);// USA==>6/30/2022      Turkey==>30/06/2022

//Change the year to ****
Integer year = myDate.year();
String expectedOutput = stringDate.replace(String.valueOf(year),'****');
System.debug('0' + expectedOutput);
```

11) valueOf() : Returns a Date that contains the value of the specified String.

Example:

```
String year = '2008';
String month = '10';
String day = '5';
String hour = '12';
String minute = '20';
String second = '20';
String stringDate = year + '-' + month + '-' + day + ' ' + hour + ':' +
minute + ':' + second;
Date myDate = Date.valueOf(stringDate);
System.debug(myDate);// 2008-10-05 12:20:20
```

12) parse() : Constructs a Date from a String. The format of the String depends on the local date format.

Example:

```
Date myDate = Date.parse('12/27/2009');// ==> Just that format works
System.debug('The date is created from from String : ' + myDate);
```

13) isLeapYear() : Returns true if the specified year is a leap year.

Leap Year:

1)If a year is divisible by 100 and divisible by 400 it is called leap year.

For example; 2000 is, 1900 is not

2)If a year is not divisible by 100 and divisible by 4 it is called leap year.

For example; 2004 is, 2007 is not

```
Boolean isLeapYear = Date.isLeapYear(2004);
System.debug('is the year leap year? ' + isLeapYear); // is the year leaap year true
System.assert(Date.isLeapYear(2004));
```

9. **Datetime:** represents a specific point in time. For example, timestamps. Contains information about both date and time.

Example:

```
1.Way: To get current GMT Date time
DateTime dt = DateTime.now();
System.debug( 'Current date and time in GMT is ' + dt); // Current date and time in GMT is  2021-11-30 11:13:23

2.Way: To get current GMT Date time
Datetime dt = System.now();
System.debug( 'Current date and time in GMT is ' + dt); // Current date and time in GMT is  2021-11-30 11:13:23
```

Note: Datetime values are stored in the force.com database in GMT time. However, when querying this data and displaying on a Visualforce page using the output Field component, the value is calculated based on the current user's time zone setting on their profile.

Note: If you want to get date and time in a specific zone;

Example:

```
String dtEST = dt.format('yyyy-MM-dd HH:mm:ss', 'EST');
System.debug('Current time in EST: ' + dtEST); // Current Time in EST:2021-11-30 06:23:13
```

Note: newInstance() method in DateTime Class uses GMT Time

```
DateTime myDateTime = DateTime.newInstance(2022, 10, 12, 18, 13, 51);
System.debug(myDateTime);// 2016-08-12 01:13:51
```

- If you use 'MM' for months you will get the month number
- If you use 'MMM' for months you will get the first 3 characters of the month name
- If you use 'MMMM' for months you will get the full name of the month name
- If you use 'M' for months you will get the month number for less than like 1, 2, 3,..
- If you use 'HH' you will get the time in 24 hours system
- If you use 'hh' you will get the time in 12 hours system
- If you use 'a' at the end you will get AM - PM at the end
- Lowe//rcase 'm's are used for minutes because of that we use uppercase 'M's for months
- If you use 'yy' for the year you will get just the last 2 digits of the year

Example: Just print hour and minute in 24 hours system

```
String t = myDateTime.format('HH:mm');
System.debug(t);
```

Example: Print date and time in Japan

Note: now() method in DateTime Class gives Date and Time in GMT

```
DateTime d = DateTime.now();
System.debug(d);//2022-03-26 15:58:58

String j = d.format('MMMM/dd/yy HH:mm:ss', 'JST');
System.debug(j);//March/27/22 01:02:23

String m = d.format('MMMM/dd/yy HH:mm:ss', 'Europe/Moscow');
System.debug(m);//March/26/22 19:04:08

String i = d.format('MMMM/dd/yy HH:mm:ss', 'Europe/Istanbul');
System.debug(i);//March/26/22 19:04:08
```

Datetime Class Methods in Apex

1) **date()** : Returns the Date component of a Datetime in the local time zone of the context user.

```
DateTime myDateTime = DateTime.newInstance(2006, 3, 16, 12, 6, 13);
Date myDate = myDateTime.date();
Date expectedDate = Date.newInstance(2006, 3, 16);
System.debug('My date: ' + myDate); // My Date:2006-03-16 00:00:00
System.debug('Expected date: ' + expectedDate); // Expected Date: 2006-03-16
00:00:00
System.assertEquals(expectedDate, myDate);
```

2) **Time():**

```
DateTime dt = DateTime.now();

Time t = dt.time();

System.debug('Time is ' + t); ==> It displays PST time // Time is
04:15:36.195Z
```

3) format(): Converts the date to the local time zone and returns the converted date as a formatted string using the locale of the context user. If the time zone cannot be determined, GMT is used.

```
DateTime myDateTime = DateTime.newInstance(1993, 6, 6, 3, 5, 8);

String myDateTimeFormatted = myDateTime.format();

System.debug('My date time is ' + myDateTimeFormatted); // My Date Time is:
6/6/1993, 3:05 AM
```

4) **format (dateFormatString):** Converts the date to the local time zone and returns the converted date as a string using the supplied Java simple date format. If the time zone cannot be determined, GMT is used.

Example:

```
Datetime myDT = Datetime.newInstance(2021, 8, 27, 16, 20, 24);

System.debug('My date time is ' + myDT);// My date time is 2021-08-27
23:20:24

String myDate = myDT.format('h:mm a'); ==> H is for 24 hours system - a is
for AM/PM, if you remove 'a' it does not display AM/PM.

System.debug('My date time is ' + myDate); // My date time is 4:20 PM
```

5) **format (dateFormatString, timezone):** Converts the date to the specified time zone and returns the converted date as a string using the supplied Java simple date format. If the supplied time zone is not in the correct format, GMT is used.

```
Datetime GMTDate = Datetime.newInstanceGmt(2021,6,1,16,1,5);
```

```
System.debug('GMT date time is ' + GMTDate); // GMT date time is: 2021-06-01
16:01:05
String strConvertedDate = GMTDate.format('MM/dd/yyyy HH:mm:ss a',
'America/New_York');
System.debug('New York date time is ' + strConvertedDate); // New York date
time is: 06/01/2021 16:01:05 PM
```

6) **isSameDay(dateToCompare):** Returns true if the Datetime that called the method is the same as the specified Datetime in the local time zone of the context user. Returns the minute component of a Datetime in the GMT time zone.

```
datetime myDate = datetime.now();
datetime dueDate = datetime.newInstance(2021, 8, 27); //==> Make it current
data
boolean dueNow = myDate.isSameDay(dueDate);
System.debug('Are the dates same? ' + dueNow); // Are the dates same? False
```

7) **valueOf()**

```
String year = '2021', month = '10', day = '5', hour = '16', minute = '20',
second = '15';
String stringDate= year+'-'+month+'-'+day+''+hour+':'+minute+ ':'+second;
Datetime myDate = Datetime.valueOf(stringDate);
System.debug('My Date is:' +  myDate); // My Date is: 2021-10-05 23:20:15
```

10. **Time:** represents a particular time. Can only be used with system methods (very limited use cases). Get current time in any time zone.

Example:

```
DateTime dt = DateTime.now();
System.debug('Current time in GMT: ' + dt); // Current time in GMT: 2021-08-
26 18:14:17
```

Time tFromDt = dt.time(); //That will conert GMT to user local time

System.debug(tFromDt);//19:01:51.318Z

Time t = time.newInstance(12, 30, 60, 40); //This another way to create time variables.

System.debug(t);//12:31:00.040Z

```
String dtEST = dt.format('yyyy-MM-dd HH:mm:ss', 'EST');
System.debug('Current time in EST: ' + dtEST);// Current time in EST: 2021-08-26 13:14:17
String hours = dtEST.substring(11,13);
String minutes = dtEST.substring(14,16);
String seconds = dtEST.substring(17,19);
System.debug(hours+':'+minutes+':'+seconds);//13:14:17
```

Note: If you want you can get user's time zone and the get user's time

```
TimeZone tz = UserInfo.getTimeZone();
System.debug('Display name: ' + tz.getDisplayName()); // Display name: (GMT-07:00) Pacific Daylight Time (America/Los_Angeles)
Note: If you want to get a specific time zone
Timezone tz = Timezone.getTimeZone('America/New_York');
System. Debug ('New York time zone is ' + tz.getDisplayName()); // New York time zone is (GMT-04:00) Eastern Daylight Time (America/New_York)
```

Arithmetic, Concatenation, and Unary Operators

In addition to the simple assignment operator (=), programming languages have to provide for the ability to perform arithmetic operations and to modify data stored in variables. These actions are made possible, in part, through the arithmetic, concatenation, and unary operators.

A) Basic Arithmetic and Concatenation:

- (+)The additive operator. Used to perform addition with numerical values. Also used to concatenate (merge) text strings together.

Example:

```
String a = 'Apex';
String b = 'is easy';
System.debug(a + ' ' + b); // Apex is easy ➔ (Put space between a and b)
```

- (-) The subtraction operator. Used to perform subtraction with numerical values.
- (*) The multiplication operator. Used to multiply numerical values.
- (/) The division operator. Used to divide numerical values.

- (%) The remainder operator. Also called modulo. Performs division, but the returned value is the remainder of the operation only. Therefore5%2 = 1.

Operators	Description	Syntax
=	= operator (Assignment Operator) assigns the value of "y" to the value of "x".	x = y
+=	+= operator (Addition assignment operator) adds the value of "y" to the value of "x" and then it reassigns the new value to "x".	x += y
*=	*= operator (Multiplication assignment operator)multiplies the value of "y" with the value of "x" and then it reassigns the new value to "x".	x *= y
-=	-= operator (Subtraction assignment operator) substract the value of "y" with the value of "x", and then it reassigns the new value of "x".	x -= y
/=	/= operator (Division assignment operator) divides the value of "x" with the value of "y" and then reassigns the new value of "x".	x /= y
|=	|= operator(OR Assignment operator), If "x" (Boolean), and "y" (Boolean) both are false, then "x" remains false.	x |= y
&=	&= operator, if "x" (Boolean), and "y" (Boolean) both are true, then "x" remains true.	x &= y
&&	&& operator (AND logical operator) it shows "short-circuiting" behavior that means "y" is evaluated only if "x" is true.	x && y
||	|| OR logical operator.	x || y
==	== Operator, if the value of "x" equals to the value of "y", then the expression evaluates true.	x == y
===	=== operator, if "x" and "y" reference the same location in the memory, then the expression evaluates true.	x === y
++	++ Increment operator.	X ++
--	-- Decrement operator.	x --

B) Compound Assignments:

Compound assignments can be formed by combining the basic arithmetic operators with the assignment operator =. Compound assignments are shorthand for performing arithmetic with a variable if you are doing a simple operation. For example, if you are adding 5 to x (assume x = 1 already), you could do x += 5 and after this, x would be 6. This is equivalent to writing x = x + 5.

- += Compound addition assignment x += 5, same as x = x + 5
- -= Compound subtraction assignment x -= 5, same as x = x - 5
- /= Compound division assignment x /= 5, same as x = x /5
- *= Compound multiplication assignment x *= 5, same as x = x * 5

C) Unary Operations

Unary operations are ones where only one operand is used, basically a single input. For example, taking the number 2 and making it negative. You do so with the unary minus operator - (commonly known as a minus sign). So -2 is the unary operation of making 2 a negative number. In programming,

you also have a unary operator to turn booleans into their opposites, called the not operator!. !true is false and !false is true.

- Assignment Operator "="

```
Integer x = 12;
System.debug(x); // 12
```

- - Unary negation operator. Used to negate a value.
- Increment operator. "+=" or "*=" or "++" Used to increment a variable's value in a single statement. For example, x++; increments the value of x by one.

```
Integer x = 2;
x += 8;  x *= 3;       x++;
System.debug(x); // 30
```

- Decrement operator "-=" or "/=" or "--". Used to decrement a variable's value in a single statement. For example, x--; decrements the value of x by one.

```
Integer x = 15;
x -= 5;
x /= 2;
x--; System.debug(x); // 4
```

- ! (not) operator. Also known as the logical complement. Inverts the value of a boolean. For example, !true is equivalent to false, and !false is equivalent to true.

D) Logical Operators

Logical operators allow the developer to check whether or not certain criteria evaluate to true or false, and to then make decisions about the application's flow based on the result. They are similar to mathematical arithmetic operators.

- **OR Operator "||"** checks to see if at least one of the left or right side is true

Note: b3 |= b4 means b3 = b3 || b4

```
boolean b1 = true, b2 = true,  b3 = false,  b4 = false;
System.debug(b1||b2); // true || true ➔true          System.debug(b2||b3);
// true || false ➔true
System.debug(b3||b2); // false || true ➔ true         System.debug(b3||b4);
// false || false ➔ false
```

- **AND Operator "&&"** If both sides are true then result is true otherwise the result is false

Note: b3 &= b4 means b3 = b3 && b4

```
boolean b1 = true,  b2 = true,  b3 = false,   b4 = false;
System.debug(b1 && b2); // true && true ➔ true
System.debug(b2 && b3); // true && false ➔ false
System.debug(b3 && b2); // false && true ➔ false
System.debug(b3 && b4); // false && false ➔ false
```

- Equality Operator(==): If the value of right equals the value of left, the expression evaluates to true. Otherwise the expression evaluates to false.

```
String s1 = 'Apex', s2 = 'Apex',  s3 = 'APEX', s4 = '',  String s5 = null;
System.debug(s1==s3); //true String comparison using == is case-insensitive
System.debug(s4==s5);//false
```

- Exact equality operator (===): If x and y reference the exact same location in memory the expression evaluates to true. Otherwise the expression evaluates to false. This is like "==" in Java We need to learn SOQL to give example

Example:

```
Integer x = 12;
Integer y;
x = y;
//Since its not saved in memory it cannot compare their location in memory.
System.debug(x===y);//Exact equality operator only allowed for reference
types: Integer
```

Example:

```
Contact a = new Contact(FirstName='Test', LastName='Contact');
insert a;
Contact b = a;
System.debug(a===b);//true
```

- **Inequality Operator(!=):** If the value of right does not equal the value of left, the expression evaluates to true. Otherwise the expression evaluates to false.

```
String s1 = 'Apex', s2 = 'Apex', s3 = 'APEX', s4 = '', s5 = null;
System.debug(s1!=s2);//false
System.debug(s1!=s3);//false ➔String comparison using != is case-insensitive
System.debug(s4!=s5);//true
```

- **Exact Inequality Operator(!==):**If x and y do not reference the exact same location in memory, the expression evaluates to true Otherwise the expression evaluates to false.
- **Comparison Operators:**
  - < (less than) checks if the left value is less than the right value

```
5<6 evaluates to true
6<5 evaluates to false
```

  - <= (less than or equal to) checks if the left side is less than or equal to the right side

```
5<=5 evaluates to true
5<=6 evaluates to true
6<=5 evaluates to false
```

  - > (greater than) checks if the left side is greater than the right side •

6>5 evaluates to true 5>6 evaluates to false

  - >= (greater than or equal to) checks if the left side is greater than or equal to right side

```
6>=6 evaluates to true
6>=5 evaluates to true
```

```
5>=6 evaluates to false
```

- **Remainder Operator:** If math.mod(number, divisor)==0 then number is divisible by divisor

Example:

```
Integer i3 = 25;
Integer i4 = 10;
Integer remainder = Math.mod(i3, i4);
System.debug('Remainder:' + remainder);//Remainder:5
```

Flow Control:

Flow control is a very important aspect of programming. It is what allows the developer to specify the order in which the application should flow, or in other words, what order things happen in. In any given application, it is rare for events to always happen in the same order; there are typically any number of conditions that could cause one thing to happen instead of another, or maybe there is an extra step to take, or a step that should be skipped. Flow control can be exacted on the application logic in a number of ways.

Short Circuit Evalation:

When using the AND and OR operators (&& and ||) there is a feature called short circuit evaluation. Short circuit evaluation allows developers to optimize their logic as well as prevent some errors during execution. For example, if an evaluation statement is a || banda is true then there is no need for the application to evaluate since we already know the expression will evaluate to true. Similarly, for the expression a && b, if a is false, there is no way for the expression to ever be true, so we can short circuit and move on. This becomes extremely useful if your expression contains division for example. If your expression is a/b == 5, but b=0,then you would get a fatal error (causing your program to break) because division by 0 is not allowed. You could prevent this problem using short circuit evaluation by first testing to ensure that b is not equal to 0: (b != 0 && a/b==5).

Sequential:

The most straightforward way to affect flow is sequential control, and this simply means that the application executes its programmed logic one line at a time, top to bottom. This can be seen in the code snippet (short piece of code, possibly taken out of context).

Example:

```
String firstName = 'Mike';

String lastName = 'Wicherski;
```

```
String fullName = firstName + ' ' + lastName;
```

Selection:

Selection controls determine which sequence, or branch, of code should be executed. Selection controls are also known as “if/else” branches, as they mimic the thought process that goes into them. “If A is true, then do X, else (otherwise), do Y).”

If/else Statement

If/else statements are self-explanatory. If this, then that, otherwise (else) that. They are a way for developers to essentially implement flow charts into the application logic. They can be expanded with as many branches as necessary by adding in an else if statement. So the structure is really if, else if, else. **The final else is actually not necessary.** It is a way to specify an action if none of the other branch criteria are met, but if it is omitted, the application simply continues sequentially.

1. If Statement:

```
If (Condition){ //if the condition is true run the code in {}.

Your codes goes here;

}// if the condition is false move to next line.
```

If it cold, I will stay home.

If I study Apex hard, I will find job very soon.

If (Boolean_Condition) {if the condition is true run this code} if the condition is false proceed to the next line.

Example1: Check if given number is even or odd.

```
Integer num = 12;
if(Math.mod(num, 2)==0){
    System.debug(num + ' is even integer');
}
if(Math.mod(num, 2)!=0){
    System.debug(num + ' is odd integer');
} // 25 is odd integer
```

Example2: If the given string is small print 'S', If the given string is medium print 'M', If the given string is Large print 'L',

```
String str = 'Small';
if(str=='Small'){
    System.debug('The size of the product is '+ 'S');
}
if(str=='Meduim'){
    System.debug('The size of the product is '+ 'M');
}
```

```
if(str==’Large’){
    System.debug(‘The size of the product is ‘+ 'L');
}
```

Example3:

If the given number is negative print 'Negative',

If the given number is zero print 'Neutre',

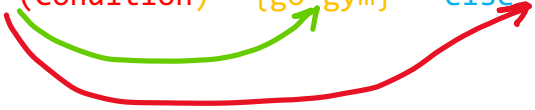
If the given number is positive print 'Positive'

```
Integer n = 12;
if(n<0){
   System.debug('Negative');
}
if(n==0){
    System.debug('Neutre');
}
if(n>0){
   System.debug('Positive');
}
```

2. **If/else Statement:** If you have 2 options, do not use multiple if, use if-else.
   if-else makes your code more logical and faster.

If it cold, I will go gym; else I will go out for running.

```
If (Boolean_Condition){
If the condition is true run this code;
}else{
If the condition is false run this code;
}
If (Condition)   {go gym}   else {go out to run}
```



Example1: Check if given number is odd or even.

```
Integer num = 11;
Integer remainder = Math.mod(num, 2);
if(remainder==0){
    System.debug('Even');
}else{
    System.debug('Odd');
}
```

Example2: If the age is between 18 and 65 then output will be “You should work”, else output will be “No need to work.

```
Integer age = 30;
if(age>18 && age<65){
    System.debug('You Should work');
}else{
    System.debug('No need to work');
} //You Should work
```

3. If()- else if () Statement: if you have more than 2 condition to test use that.

```
If (Boolean_Condition1){
//do this if the condition1 is true, then exit if/else if
//if condition1 is false test the next condition
}If Else(Boolean_Condition2){
//do this if the condition2 is true, then exit if/else if
//if condition2 is false test the next condition
}If Else (Boolean_Condition3){
//do this if the condition3 is true, then exit if/else if
//if condition3 is false test the next condition
}Else{
//do this if the none of the condition above met, then exit if/else if
}
```

Example1: for the name of the day, print the number of day.

Sunday==>1 Monday==>2

```
String d = 'WEDNESDAY';
if(d.equalsIgnoreCase('Sunday')){
    System.debug(1);
}else if(d.equalsIgnoreCase('Monday')){
    System.debug(2);
}else if(d.equalsIgnoreCase('Tuesday')){
    System.debug(3);
}else if(d.equalsIgnoreCase('Wednesday')){
    System.debug(4);
}else if(d.equalsIgnoreCase('Thursday')){
    System.debug(5);
}else if(d.equalsIgnoreCase('Friday')){
    System.debug(6);
}else if(d.equalsIgnoreCase('Saturday')){
    System.debug(7);
}else{
    System.debug('Input is incorrect please put a day name...');
}
```

Example2: Type code to check if a given year is Leap Year or not If the year is divisible by 100 then it must be divisible by 400 .If the year is not divisible by 100 then it must be divisible by 4

```
Integer year = 1997;
if(year<0){
    System.debug('Negative year value is not valid');
}else if(Math.mod(year, 100)==0 && Math.mod(year, 400)==0){
    System.debug(year + ' is leap year');
}else if(Math.mod(year, 100)!=0 && Math.mod(year, 4)==0){
    System.debug(year + ' is leap year');
}else{
    System.debug(year + ' is not leap year');
}// 1997 is not leap year.
```

Example3: According to the given rules convert number grades to letter grades

```
        0-49==>FF    50-65==>DD   70-79==>CC    80-89==>B    90-100==>A
```

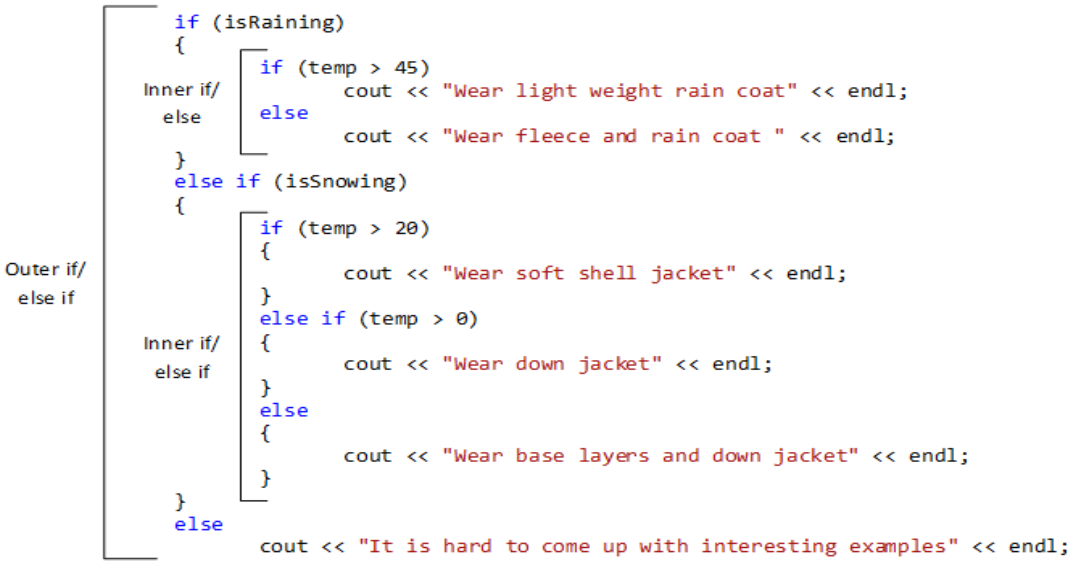
```
Integer grade = 10;
if(grade<0){
    System.debug('Negative values are not acceptable');
}else if(grade<50){
    System.debug('F');
}else if(grade<70){
    System.debug('D');
}else if(grade<80){
    System.debug('C');
}else if(grade<90){
    System.debug('B');
}else if(grade<101){
    System.debug('A');
}else{
    System.debug('Values greater than 100 are not acceptable');
}
```

Example 10: Print the month names for the given month number

```
Integer m = 2;
if(m==1){
  System.debug('January');
}else if(m==2){
   System.debug('February');
```

```
}else if(m==3){
   System.debug('March');
}else if(m==4){
   System.debug('April');
}else if(m==5){
   System.debug('May');
}else if(m==6){
   System.debug('June');
}else if(m==7){
   System.debug('July');
}else if(m==8){
   System.debug('August');
}else if(m==9){
   System.debug('September');
}else if(m==10){
   System.debug('October');
}else if(m==11){
   System.debug('November');
}else if(m==12){
   System.debug('December');
}else{
   System.debug('Invalid month number');
}
```

4. Nested if Statement:




```
if (isRaining)
{
        if (temp > 45)
                cout << "Wear light weight rain coat" << endl;
        else
                cout << "Wear fleece and rain coat " << endl;
}
else if (isSnowing)
{
        if (temp > 20)
        {
                cout << "Wear soft shell jacket" << endl;
        }
        else if (temp > 0)
        {
                cout << "Wear down jacket" << endl;
        }
        else
        {
                cout << "Wear base layers and down jacket" << endl;
        }
}
else
        cout << "It is hard to come up with interesting examples" << endl;
```

Example1 *:* Type code by using nested if statement

If a number is even then check if it is divisible by 3 or not.
If it is divisible by 3 the output will be “Perfect even number” otherwise, the output will be “Good even number.”
If the number is odd then check if it is divisible by 3 or not. If it is divisible by 3 the output will be “Perfect odd number” otherwise, the output will be “Good odd number.”

```
Integer m = 123;
if(Math.mod(m, 2)==0){
    if(Math.mod(m, 3)==0){
        System.debug(m + ' is perfect even number');
    }else{
        System.debug(m + ' is good even number');
    }
}else{
    if(Math.mod(m, 3)==0){
      System.debug(m + ' is perfect odd number');
    }else{
        System.debug(m + ' is good odd number');
    }
}//123 is perfect odd number
```

Example2:
If the person is male
i)If he is less than 16, print 'Should go to school'
ii)If he is less than 66, print 'Should work'
iii)If he is greater than 65, print 'Should be retired'
If the person is female
i)If she is less than 18, print 'Should go to school'
ii)If she is less than 60, print 'Should work'
iii)If she is greater than 60, print 'Should be retired'

```
String gender = 'Female';
Integer age = 12;

if(gender.equalsIgnoreCase('Male')){

    if(age<0){
        System.debug('Enter valid age');
    }else if(age<18){
        System.debug('Should go to school');
    }else if(age<66){
        System.debug('Should work');
    }else{
        System.debug('Should be retired');
    }

} else if(gender.equalsIgnoreCase('Female')){
```

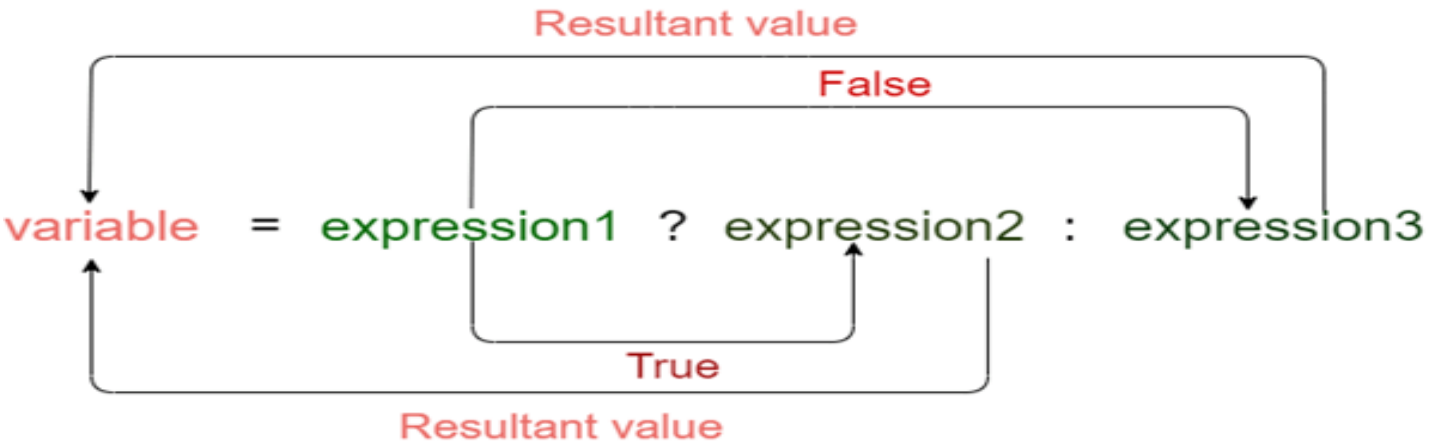
```
    if(age<0){
        System.debug('Enter valid age');
    }else if(age<18){
        System.debug('Should go to school');
    }else if(age<60){
        System.debug('Should work');
    }else{
        System.debug('Should be retired');
    }

} else {
    System.debug('Your gender was not defined in that application');
}
```

5. Ternary Operator:

Ternary statements (operators) allow the developer to write a single if/else branch statement on a single line. This is considered shorthand, and while it may make development easier and allows for writing less code, it also lowers the readability aspect. This should be taken into consideration in larger applications and its use should be limited.

A ternary statement has the following
syntax:

```
(if condition )?(if_true):(if_false);
```



Example1:

```
Boolean isFive = (5==5)?True:False;
```

Example2:

```
Integer num = 12
(num>5)?(2*num): (3*num);
Or
Integer int = (num>5)?(2*num): (3*num);
```

Example3= The code to check to be even or odd

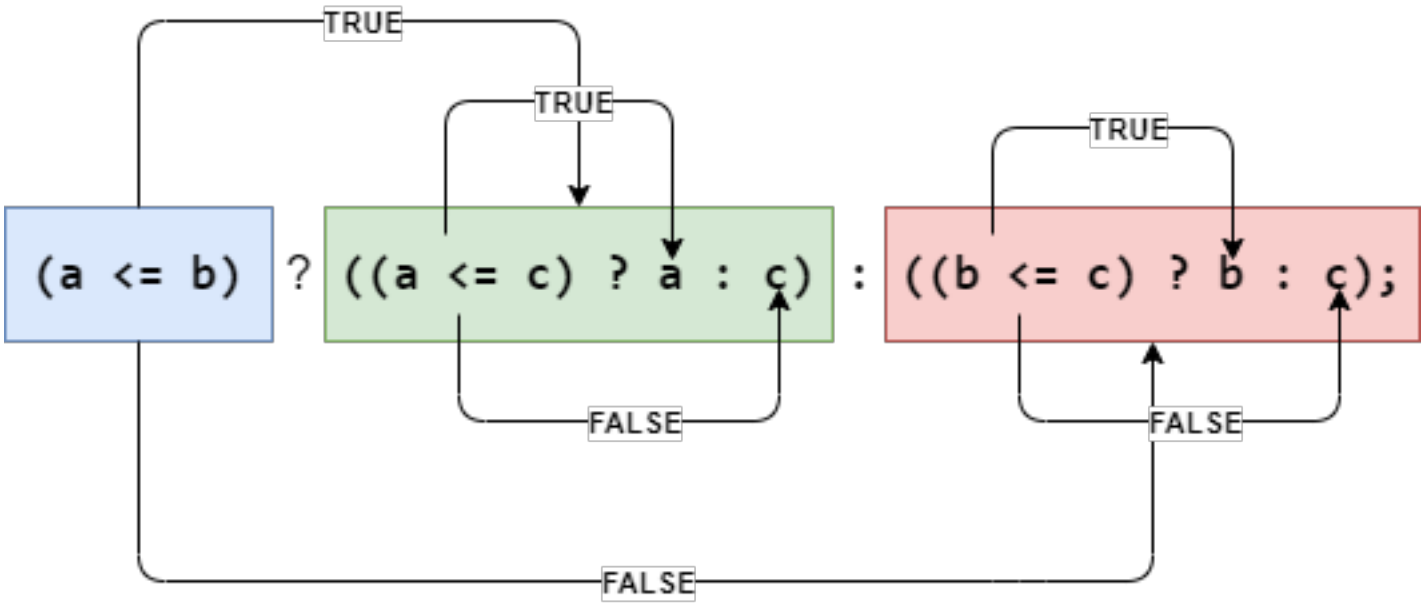
```
Integer num = 13;
```

```
String result = Math.mod(num, 2)==0 ? num + ' is even integer' : num + ' is odd
integer';
System.debug(result);
```

Example 4: Write a program to print the minimum of 2 integers on the console. Use ternary...

```
Integer x = 13;
Integer y = 13;
Integer result = x<y ? x : y;
System.debug(result);
```

6. Nested Ternary Operator:



Example1: Type Apex code by using nested ternary.
Write a program to check if a year is leap year or not. If the year is divisible by 100 then it must be divisible by 400.If a year is not divisible by 100 then it must be divisible by 4.

```
Integer year = 2000;
String result = Math.mod(year, 100)==0 ? (Math.mod(year, 400)==0 ? 'Leap year' :
'Not leap year') : (Math.mod(year, 4)==0 ? 'Leap year' : 'Not leap year');
System.debug(result); 2000 is leap year.
```

Example2: Type code to check the password with ternary operator
If it has more than 8 characters, initial should be 'i'
If it does not have more than 8 characters initial should be 'K'
Solve the task by using nested-ternary

```
String password = 'AAAAAAAAA';
String isValid = (password.length()>8) ? ( (p.startsWith('i')) ? ('Valid') :
('Invalid') ) : ( (password.startsWith('K')) ? ('Valid') : ('Invalid') );
System.debug(isValid);
```

7. Switch Statement: Apex switch statement expressions can be one of the following types.

• Integer • Long • sObject • String • Enum

Example1: For 'Male' print 'Boy', for 'Female' print 'Girl', for others print 'Undefined'

```
1.Way:
String gender = 'Male';
if(gender.equalsIgnoreCase('Male')){
    System.debug('Boy');
}else if(gender.equalsIgnoreCase('Female')){
    System.debug('Girl');
}else{
    System.debug('Undefined');
}

2.Way:
String gender = 'male';
switch on gender {
 when 'female'{
System.debug('This is a girl');
 }
when 'male'{
System.debug('This is a boy');
}
when else {
System.debug('Invalid input');
 }
} // This is a boy
```

Exampe2: Print the day name according to given number:

```
Integer dayNumberInWeek = 3;
switch on dayNumberInWeek {
when 1{
System.debug('Sunday');
 }
when 2
{ System.debug('Monday');
}
when 3{
System.debug('Tuesday');
 }
when 4{
System.debug('Wednesday');
}
when 5{
System.debug('Thursday');
```

```
 }
when 6{
System.debug('Friday');
}
when 7{
System.debug('Saturday');
}
when else {
 System.debug('Invalid input');
}
 }//Tuesday
```

Example3: Print 'Weekday' for weekdays, print 'Weekend day' for weekend days
Saturday==>'Weekend day' Tuesday==>'Weekday' etc.

```
String day = 'SATURDAY';
switch on day.toLowerCase(){

    when 'sunday', 'saturday'{
        System.debug('Weekend day');
    }
    when 'monday', 'tuesday', 'wednesday', 'thursday', 'friday' {
        System.debug('Weekday');
    }
    when else{
       System.debug('Invalid day name');
    }
}
```

Example 4: How to use Enum in switch statement

Enum: An enum is an abstract data type with values that each take on exactly one of a finite set of identifiers that you specify. Enums are typically used to define a set of possible values that don't otherwise have a numerical order. Typical examples include the suit of a card, or a particular season of the year.
To define an enum, use the enum keyword in your declaration and use curly braces to demarcate the list of possible values.

Example: the following code creates an enum called Season:

```
Public enum Seasons {Winter, Spring, Summer, Fall}
```

First Step:
Create an Enum by following the given steps
➔File ==> New ==> Apex Class ==> Give a name for the Enum ==> Click on OK
You will see a Class on the screen, change "class" keyword to "enum" then File ==> Save

Second Step:

```
Season s = Season.WINTER;
switch on s{
```

```
when WINTER {
System.debug('Snow Boarding');
}
when SPRING, SUMMER {
System.debug('Fishing');
}
when FALL {
System.debug('Trekking');
}
when else {
System.debug('None of the above'); } }
```

BASIS	SWITCH STATEMENT	IF-ELSE STATEMENT
Definition	It is a control statement that uses single enumerated value to determine which case or statement needs to be executed	It is control statement that evaluates relational expression to determine which block would be executed, i.e if or else
Expression	It uses a single integer or enumerated value as its expression	It uses a relational statement or combination of relational statements as its expression
How It Works	A switch statement checks for equality by matching the value of the expression with the cases	If-statement works differently by evaluating a relational expression to make a decision
Execution Sequence	Based on the value, a corresponding case would be selected and executed until a break statement is found. if no case is matched then default statement is executed	Based on condition evaluation, if true is evaluated then if-block would be executed, otherwise else-block
Speed	It runs faster than if-statement as it just checks for equality	It runs slower as it evaluates an expression first and then makes decision
Default Execution	Default statement would be executed if no match is found	Else statement would be executed, if the condition is false
Editing	If more choices needed to put in the code, then simply more cases can be appended within a switch statement	If more choices needed to put in, then it requires as much number of If-else statement to be declared as many choices are needed

Example5: Get the season names from Seasons Enum, and print the month names for every season

```
switch on Seasons.WINTER{
    when WINTER{
        System.debug('December, January, February');
    }
    when SPRING{
        System.debug('March, April, May');
    }
    when SUMMER{
        System.debug('June, July, August');
    }
    when AUTUMN{
        System.debug('September, October, November');
    }
}
```

How to Create Class and Functions (Method) in Apex

Class in Apex: An Apex class is a template or blueprint from which Apex objects are created. An object is instances of class. Classes consist of other classes, user-defined methods, variables, exception types, and static initialization code.

Open Developer Console➔ file➔new➔Apex Class➔Write Class name

To define a class, specify the following:

1. Access modifiers:
   You must use one of the access modifiers (such as public or global) in the declaration of a top-level class.
   You do not have to use an access modifier in the declaration of an inner class.
2. Optional definition modifiers (such as virtual, abstract, and so on)
3. Required: The keyword class followed by the name of the class
4. Optional extensions and/or implementations

Below is the sample structure for Apex class definition:

private | public | global ➔ Access Modifiers

[virtual | abstract | with sharing | without sharing]➔ Optional definition modifiers (sharing Modes)

class ClassName [implements InterfaceNameList] [extends ClassName] {//➔Class definition

// Classs Body

}

Following is a sample structure for Apex class definition

```
public class MySampleApexClass {        //Class definition and body
    public static Integer myValue = 0;   //Class Member variable
    public static String myString = ''; //Class Member variable

    public static Integer getCalculatedValue () {
    // Method definition and body
    // do some calculation
        myValue = myValue+10;
        return myValue;
    }
}
```

Access Modifiers

An access modifier restricts the access of a class, constructor, data member, and method in another class there are 4 different access modifiers in Apex.

Private: This access modifier is the default and means that the method or variable is accessible only within the Apex class in which it's defined. If you don't specify an access modifier, the method or variable is private.

Note1: An outer class cannot be private

Note2: To make a class member "private", we do not type any access modifier

Protected: If you make any class member "protected", it means the class members (variables, functions) can be used by child classes as well.

Note: Class itself cannot be protected.

Public: If you make any class or any class member "public", then it is accessible for the entire project.

Note: Class member means variables and functions inside the class.

Global: If you make any class or class member "global", it means the class or class member can be used from other project as well.

Sharing Modes: Let us now discuss the different modes of sharing.

1. **With Sharing:** This is a special feature of Apex Classes in Salesforce. When a class is specified with 'With Sharing' keyword then it has following implications: When the class will get executed, it will respect the User's access settings and profile permission. Suppose, User's action has triggered the record update for 30 records, but user has access to only 20 records and 10 records are not accessible. Then, if the class is performing the action to update the records, only 20 records will be updated to which the user has access and rest of 10 records will not be updated. This is also called as the User mode.
2. **Without Sharing:** Even if the User does not have access to 10 records out of 30, all the 30 records will be updated as the Class is running in the System mode, i.e., it has been defined with Without Sharing keyword. This is called the System Mode.
3. **Virtual**: If you use the 'virtual' keyword, then it indicates that this class can be extended, and overrides are allowed. If the methods need to be overridden, then the classes should be declared with the virtual keyword.
4. **Abstract**: If you declare the class as 'abstract', then it will only contain the signature of method and not the actual implementation.

Example of Parent and Child Classes:

```
public class myOuterClass {
   // Additional myOuterClass code here
   class myInnerClass {
     // myInnerClass code here
   }
 }
```

Class Members:

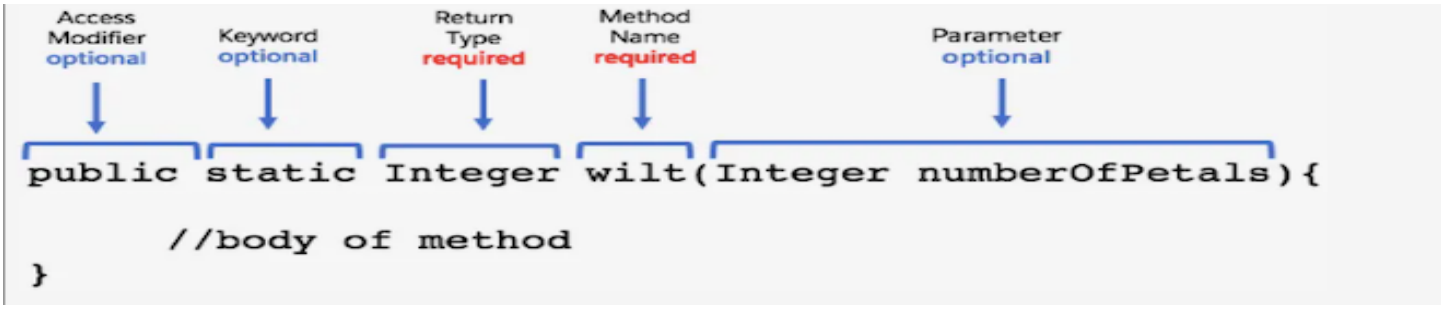
Class Variables: To declare a variable, specify the following:

1. Optional: Modifiers, such as public or final, as well as static.
2. Required: The data type of the variable, such as String or Boolean.
3. Required: The name of the variable.
4. Optional: The value of the variable.

Use the following syntax when defining a variable:

[public | private | protected | global] [final] [static] data_type variable_name [= value]

Class Methods (functions): Methods are defined within a class. A method describes the behaviors inherited by objects of that class. A class can have one or more methods. Return type is mandatory for



method and if method is not returning anything then you must mention void as the return type. Additionally, Body is also required for method.

To define a method, specify the following:

1. Optional: Modifiers, such as public or protected.
2. Required: The data type of the value returned by the method, such as String or Integer. Use **void** if the method doesn't return a value.
3. Required: A list of input parameters for the method, separated by commas, each preceded by its data type, and enclosed in parentheses (). If there are no parameters, use a set of empty parentheses. A method can only have 32 input parameters.
4. Required: The body of the method, enclosed in braces {}. All the code for the method, including any local variable declarations, is contained here.

Example

```
//Method definition and body
public static Integer getCalculatedValue () {

   //do some calculation
   myValue = myValue+10;
   return myValue;
}
```

This method has return type as Integer and takes no parameter.

A Method can have parameters as shown in the following example −

```
// Method definition and body, this method takes parameter price which will then be used
// in method.

public static Integer getCalculatedValueViaPrice (Decimal price) {
   // do some calculation
   myValue = myValue+price;
   return myValue;
}
```

How to create an object?

Apex01 obj1 = new Apex01( ) ;

Apex01 is class name or data type
obj1 is object name
Apex01( ) is constructor
new is keyword to create an object from scratch

Sample Class Example:

```
public class MyClass {
    Integer myInteger = 10;

    public void myMethod (Integer multiplier) {
       Integer multiplicationResult;
       multiplicationResult = multiplier*myInteger;
       System.debug('Multiplication is '+multiplicationResult);
    }
 }
```

// Object Creation: Creating an object of class

MyClass objClass = new MyClass();

```
// Calling Class method using Class instance
objClass.myMethod(100);
```

sObject creation

sObjects are the objects of Salesforce in which you store the data. For example, Account, Contact, etc., are custom objects. You can create object instances of these sObjects.

```
// Execute the below code in Developer console by simply pasting it
// Standard Object Initialization for Account sObject
Account objAccount = new Account(); // Object initialization
objAccount.Name = 'Testr Account'; // Assigning the value to field Name of Account
objAccount.Description = 'Test Account';
insert objAccount; // Creating record using DML
System.debug('Records Has been created '+objAccount);
// Custom sObject initialization and assignment of values to field
APEX_Customer_c objCustomer = new APEX_Customer_c ();
objCustomer.Name = 'ABC Customer';
objCustomer.APEX_Customer_Decscription_c = 'Test Description';
insert objCustomer;
System.debug('Records Has been created '+objCustomer);
```

Example 2: Example for creating standard and custom objects from class.

```
// Execute the below code in Developer console by simply pasting it
// Standard Object Initialization for Account sObject
Account objAccount = new Account(); // Object initialization
objAccount.Name = 'Testr Account'; // Assigning the value to field Name of Account
objAccount.Description = 'Test Account';
insert objAccount; // Creating record using DML
System.debug('Records Has been created '+objAccount);

// Custom sObject initialization and assignment of values to field
APEX_Customer_c objCustomer = new APEX_Customer_c ();
objCustomer.Name = 'ABC Customer';
objCustomer.APEX_Customer_Decscription_c = 'Test Description';
insert objCustomer;
System.debug('Records Has been created '+objCustomer);
```

What is constructor?

A constructor is a code that is invoked when an object is created from the class blueprint. It has the same name as the class name. We do not need to define the constructor for every class, as by default a no-argument constructor gets called. Constructors are useful for initialization of variables or when a process is to be done at the time of class initialization.

1. Constructor is inside the class
2. Constructor has the same name with the class
3. Constructor has no return type

Note 1: Constructor is not a method, because it has no return type.

Do not mention “constructor method”, mention just “constructor” while you talk about constructor

Note 2: When we create a class, Java creates a “default constructor” automatically for the class.

So, we can create objects by using every class without creating any constructor.

Note 3: Default constructors have no any parameters ==> *MyClass( ) { }*

Note 4: When we create a constructor by ourselves, default constructor is cancelled by Java

Use the following syntax when defining a constructor:

```
Public class MyClass {
    MyClass() {  }  // Constructor with no-argument
    MyClass(String str) { }// Constructor with one Argument:
    MyClass(Integer i, String str) { }// Constructor with 2 argument:
}
```

```
// Class definition and body
public class MySampleApexClass2 {
   public static Double myValue;   // Class Member variable
   public static String myString;  // Class Member variable

   public MySampleApexClass2 () {
      myValue = 100; //initialized variable when class is called
   }

   public static Double getCalculatedValue () { // Method definition and body
      // do some calculation
      myValue = myValue+10;
      return myValue;
   }

   public static Double getCalculatedValueViaPrice (Decimal price) {
      // Method definition and body
      // do some calculation
      myValue = myValue+price; // Final Price would be 100+100=200.00
      return myValue;
   }
}
```

Overloading Constructors

Constructors can be overloaded, i.e., a class can have more than one constructor defined with different parameters.

```
public class MySampleApexClass3 {  // Class definition and body
   public static Double myValue;   // Class Member variable
   public static String myString;  // Class Member variable

   public MySampleApexClass3 () {
      myValue = 100; // initialized variable when class is called
      System.debug('myValue variable with no Overaloading'+myValue);
   }

   public MySampleApexClass3 (Integer newPrice) { // Overloaded constructor
      myValue = newPrice; // initialized variable when class is called
      System.debug('myValue variable with Overaloading'+myValue);
   }

      public static Double getCalculatedValue () { // Method definition and body
      // do some calculation
      myValue = myValue+10;
      return myValue;
   }

   public static Double getCalculatedValueViaPrice (Decimal price) {
      // Method definition and body
      // do some calculation
      myValue = myValue+price;
      return myValue;
   }
}
```

```
// Developer Console Code
MySampleApexClass3 objClass = new MySampleApexClass3();
Double FinalPrice = MySampleApexClass3.getCalculatedValueViaPrice(100);
System.debug('FinalPrice: '+FinalPrice);
```

Purpose of Static Keyword in Apex:

Static variables will be instantiated only once when class is loaded, and this phenomenon can be used to avoid the trigger recursion. Static variable value will be same within the same execution context and any class, trigger or code which is executing can refer to it and prevent the recursion

- The static word can be used to attach a Variable or Method to a Class.
- The Variable or Method that are marked static belongs to the Class rather than to any particular object
- To call a static variable or method, no need to create an object
- To call a non-static variable or method, you have to create an object

	within non-static method of same class	within static method of same class	within non-static method of other class	within static method of other class
Non-Static Method or Variable	Access directly	Access with object reference	Access with object reference	Access with object reference
Static Method or Variable	Access directly	Access directly	Access with object reference	Access with object reference

Tutorial4us.com

Example1:

➔**Create an object from myClassClass to access getRandomChar() function**

```
public class myClass {

    public Static Integer counterStatic = 0; //Class member variable
    public Integer counterNonStatic = 0; //class member variable

    Public myClass (){
        counterStatic =counterStatic+1;
        counterNonStatic = counterNonStatic+1;
    }

    //Create a function which return random character from a given string
    public String getRandomChar (String name){
        Integer lastIndex = name.length()-1;
        Integer randomIndex = Integer.valueOf(math.random()*lastIndex);
        //random() method will give a random number between 0-1
        //if we multiply random number with lastIndex we will asure
        //that not getting a index higher that our string have
        //then if save our result to integer container we will remove decimal part
        return name.substring(randomIndex, randomIndex+1);
    }
}
```

```
myClass obj1 = new myClass (); // myClass() is called "Construuctor", it is used to create object from a class
```

➔By using obj1 you can access to the getRandomChar() function (try that without **static** keyword in front of method)

```
String randomChar = obj1.getRandomChar('Noah');
System.debug(randomChar); //you will get random character from Noah
```

➔Let us try to access class members without creating object

```
String randomChar = myClass.getRandomChar('KEMAL');
System.debug(randomChar); //Non static method cannot be called just by using class name
```

➔Create objects from Apex001 and check the counters values

```
myClassobj1 = new myClass ();
System.debug(myClass.counterStatic);
System.debug(obj1.counterNonStatic);
myClass obj2 = new myClass ();
System.debug(myClass.counterStatic);
System.debug(obj2.counterNonStatic);
myClass obj3 = new myClass ();
System.debug(myClass.counterStatic); //Static variable counts all objects ==> 3
System.debug(obj3.counterNonStatic);//Non static variable counts just the last object => 1
```

Example2: The usage of static keyword

```
public class createId {

    //Type code to create id for every students
    //id format is year+Initials+age+counter
    public String name = '';
    public Integer age = 0;
    public String id = '';
    public Static Integer counter =1000;

    Public createId (String name, Integer age){
        this.name = name;
        this.age=age;
        Integer currentYear = system.today().year();
        String initialFirstName = name.substring(0,1);
        Integer indexofSpace = name.indexOf(' ');
        String initialOfLastName = name.substring(indexofSpace+1, indexOfSpace+2);
        This.id = currentYear+initialFirstName+initialOfLastName+age+counter;
        counter =counter+1;
    }

}

//Developer Console Code to Test Above Class:
createId std1 = new createId('Noah Ozay', 30);
System.debug(std1.id);//2022NO301001
createId std2 = new createId('Kemal Sunal', 56);
System.debug(std2.id);//2022KS561002
createId std3 = new createId('Victoria Chen', 45);
System.debug(std3.id);//2022VC451003
createId std4 = new createId('Razi Kumar', 70);
System.debug(std4.id);//2022RK701004
```

Constants: As in any other programming language, Constants are the variables which do not change their value once declared or assigned a value. In Apex, Constants are used when we want to define variables which should have constant value throughout the program execution. Apex constants are declared with the keyword 'final'.

Example:

```
Integer age=31;
final String name='Mustafa';
System.debug('My age now is ' +age);
System.debug('My name now is '+name);
```

```
age=32;
name='Mark';
System.debug('My age next year is going to be ' +age);
System.debug('My name next year is going to be '+name); //
System.FinalException: Final variable has already been initialized
age=33;
System.debug('My age the year after that is going to be ' +age);
```

Example 3: How to use Functions in switch statement

First Step: Create a class and create function to get random character from alphabet

```
public class TestClass1{
    public static String getRandomCharacter(){
    String s = 'ABCDEFGHIJKLMNOPRSTUVYZ';
    Integer maxIndex = s.length() - 1;
    //"Math.random() * maxIndex" ==> returns random value from 0 to the
value of maxIndex
    Integer randomIndex = Integer.valueof((Math.random() * maxIndex));
    return s.substring(randomIndex, randomIndex+1);
    }
}
```

Second Step: Call the function by using class name then store the return value in a String container

String c = TestClass1.getRandomCharacter();
System.debug(c);

Third Step: Use the function in switch statement

```
switch on TestClass1.getRandomLetter(){
    when 'A', 'a' {
    System.debug('First character');
    }when 'B', 'b' {
    System.debug('Second character');
    }when 'C', 'c' {
    System.debug('Third character');
    }when 'D', 'd' {
    System.debug('Second character');
    }when else {
    System.debug('Other characters');
    }
}
```

LOOPS

If you need to do same steps again and again use loops. These types of procedural loops are supported:

- while (Boolean_condition) {statement; then increment/decrement}
- do {statement; then increment/decrement } while (Boolean_condition);
- for (initialization; Boolean_exit_condition; increment) {statement;}
- for (variable : array_or_set) {statement;}
- for (variable : [inline_soql_query]) {statement;}

1) **While- loop:** Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

```
while (Condition){
Code-Block;
Increment or decrement;
}
```

Example1: Lets writing code to outputs the numbers from 1 to 10 into the debug console.

```
Integer count = 1;
while (count < 11) {
    System.debug('Value of count is ' + count);
    count++;
}// Here as you can see from debug console output has been written 10 times
one under the other
```

Step2 lets improve that same Code:

```
Integer count = 1;
String numbers= '';
while (count < 11) {
```

```
    system.debug(numbers=numbers +','+count);
    count++;
}//As you can see it has written numbers and add another number each time
one under the other
```

Step2 lets improve that same Code and remove comma before 1 (,1) and write all number once side by side

```
Integer count = 1;
String numbers= '';
while (count < 11) {
    if(count==1){
        numbers=''+count;
    }else{
        numbers=numbers +','+count;
    }
    count++;
}
System.debug('Numbers from 1 to 10 : ' + numbers);
```

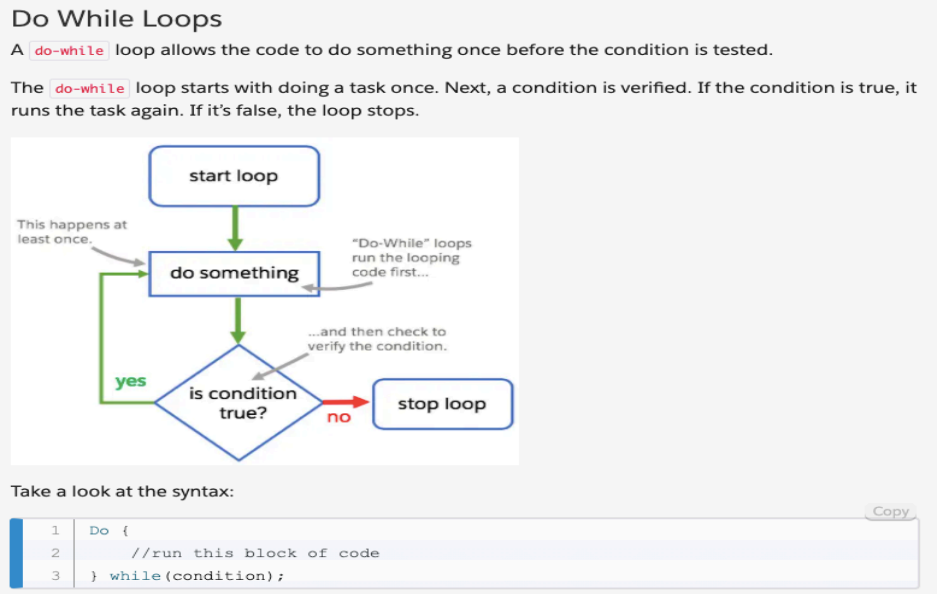
Example2: Type Apex code by using while loop, It should find the sum of the digits of given number.
856 ==> 8+5+6 = 14

```
Integer num = 856;
Integer sum = 0;
while(num!=0){
    sum = sum + Math.mod(num, 10);
    num = num / 10;
}
System.debug(sum);
```

2) **Do-While Loops:** Unlike the for and the while loops which test the loop condition at the top of the loop, the do...while loop checks its condition at the bottom of the loop. Because of that under every condition "do-while-loop" will be executed at least once.

Syntax:

```
do {
    Code-Block;
    Increment or decrement;
    } while (condition);
```



Example1: Lets do previous example by using do-while loop.

```
Integer count = 1;
do {
    System.debug('Value of count is ' + count);
    count++;
} while (count < 11);
```

Example2: lets improve same code to write numbers side by side on the console.

```
Integer count = 1;
String numbers='';
do {
    if(count==1){
        numbers = ''+count;
    } else{
        numbers=numbers + ','+count;
    }
    count++;
} while (count < 11);
system. Debug(numbers);
```

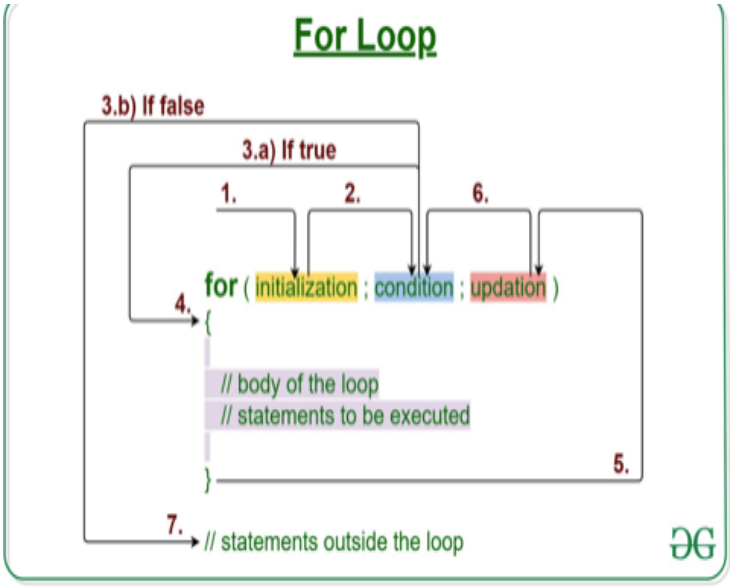
3) **Traditional For- Loop:** A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times. Consider a business case wherein, we are required to process or update the 100 records in one go. This is where the Loop syntax helps and makes work easier

Syntax:

```
for (init_stmt; exit_condition; increment_stmt) {
    code_block
}
```

When executing this type of for loop, the Apex runtime engine performs the following steps, in order:

1. Execute the init_stmt component of the loop. Note that multiple variables can be declared and/or initialized in this statement.
2. Perform the exit_condition check.
3. a) If true, go to the code_block.
   b) If false, the loop exits.
4. Execute the code_block
5. Execute the increment_stmt statement.
6. Return to Step 2.
7. Exit the loop run the code after loop statements.



Example: let's write down numbers from 1 to 10 on debug console with for- loops:

```
for (Integer i = 0; i < 10; i++) {
    System.debug(i+1);
}
```

Example: let's improve our code step by step:

```
String numbers= '';
for (Integer i = 1; i < 11; i++) {
    System.debug(numbers=numbers +','+ i);
}
```

Example: Lets improve code as below:

```
String numbers= '';
for (Integer i = 1, j = 0; i < 11; i++) {
    if (i==1){
        numbers=''+i;
    }else{
        numbers=numbers+','+i;
            }
}

System.debug(numbers);
```

Example2: Write a program to print odd numbers from 35 to 75 on the console by using for-loop. (35 included, 75 not included)

```
for(Integer i=35; i<75; i++){
    if(Math.mod(i,2)!=0){
```

```
            System.debug(i);
        }
}
```

Example3: Type code to print all unique characters on the console from a given string.
For example Java ➔ Jv

```
String  str = 'Java';

For( Integer i=0; i<str.length(); i++) {

      If(str.indexOf(str.substring(i, i+1) ==
str.lastIndexOf(str.substring(i , i+1)){
            System.debug(str.substring(i, i+1));
      }
}
```

Way2: countMatches() ;This methods counts number of same character in a string

```
String str2 ='Ankara';
system.debug(str2.countMatches('a')); // 2
String str ='Java';
String result='';
for (Integer i =0;   i<str.length() ;   i++)  {

      string chrc = str.substring(i,i+1);

    if ( str.countMatches(chrc)== 1){
       result=result + chrc;
    }
}
```

Example4: Find all Prime numbers between 1 and 100.

```
for(integer i=1;i<=100;i++){
        integer count=0;
        for(integer j=i; j>=1;j--){
            if(math.mod(i,j)==0 ){
                count++;
            }
        }
        if( count==1 ||count==2){
          system.debug(i);
        }
}
```

4) **For- Each- Loop:** This is the enhanced-loop. for-each-loop can be used with arrays or collections(lists, sets, queues).

The list or set iteration for loop iterates over all the elements in a list or set. Its syntax is:

Syntax:

```
for (variable : list_or_set) {
    code_block
}
```

where **variable** must be of the same primitive or sObject type as list_or_set.

When executing this type of for loop, the Apex runtime engine assigns variable to each element in list_or_set, and runs the code_block for each value.

For example, the following code outputs the numbers 1 - 10 to the debug log:

```
Integer[] myInts = new Integer[]{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
for (Integer i : myInts) {
    System.debug(i);
}
```

Example1:

```
String[] arr = new String[]{'A', 'K', 'L', 'E', 'X'};
System.debug(arr);//(A, K, L, E, X)
for(String w : arr){
    System.debug(w + '!');
}
```

Example2: Find the vowels in a String by using arrays and for-each loop

```
String s = 'Learn Apex, earn money';
String[] arr = s.split('');
System.debug(arr);//(L, e, a, r, n,  , A, p, e, x, ,  , e, a, r, n,  , m, o,
n, e, y)
for(String w : arr){
    if(w.equalsIgnoreCase('a') || w.equalsIgnoreCase('e') ||
w.equalsIgnoreCase('i') ||
       w.equalsIgnoreCase('o') || w.equalsIgnoreCase('u')){
        System.debug(w);
```

```
    }
}
```

Type of Loop control structures

1. Break

The break keyword is used to end the loop when any condition is met. Normally we write the Break keyword inside an "If" condition and it gets executed if the condition is true and breaks the loop. Let's see an example.

Syntax: Write following syntax in Loop body.

If (condition) {

Break;

}

Example:

```
for(i=0; i<10; i++){
    if(i==5){
        break; //after it find 5  loop statement will be terminated
    }
    System.debug(i);
}
```

2. Continue

When it comes to Continue Keyword, It just makes the current iteration end and continues with the following iterations. That means the Continue statement executes the current ongoing loop breaks but then continues again with its following iterations unlike in Break Statement. Let's see an example to understate the difference between the both.

```
for(i=0; i<10; i++){
    if(i==5){
        break; //when 5 is found it will skip to next iteration.
    }
    System.debug(i);
}
```

Iterating Collections

Collections can consist of lists, sets, or maps. Modifying a collection's elements while iterating through that collection is not supported and causes an error. Do not directly add or remove elements while iterating through the collection that includes them.

Adding Elements During Iteration

To add elements while iterating a list, set or map, keep the new elements in a temporary list, set, or map and add them to the original after you finish iterating the collection.

Removing Elements During Iteration

➔To remove elements while iterating **a list,** create a new list, then copy the elements you wish to keep. Alternatively, add the elements you wish to remove to a temporary list and remove them after you finish iterating the collection.

➔ To remove elements while iterating a **map or set**, keep the keys you wish to remove in a temporary list, then remove them after you finish iterating the collection.

Practice1:

```
Integer x = 9;
Integer y= 2;
Integer sum = 0;
While (x>y){
      x--;
      y= y+3;
      sum = sum+x+y;
}
System.debug(sum); //30
```

Practice2: Type all perfect even integers from 130 to 10. (Perfect even numbers are the ones which divisible by 2 and 3. For example 6 and 12 are perfect even numbers.

```
for (Integer i=130 ; i>9; i--){
      if (math.mod(i,2)==0 & math.mod(i,3)==0){
            System.debug(i);
      }
}
```

Practice3: Convert 'Apex to A*p*e*x*'

```
String str = 'Apex';
String result = '';
for(Integer i=0; i<str.length(); i++){
      result=result + str.substring(i,i+1) + '*';
}
System.debug(result);
```

Practice4: Type code to find the sum of integers from 12 to 200 ==> 12+13+14+15… (12 and 200 is included)

```
Integer sum = 0;
for(Integer i=12; i<201; i++){
    sum = sum + i;
```

```
}
System.debug(sum);
```

Practice5: Type code to find the multiplication of integers from 6 to 9 ==> 6x7x8x9

```
Integer result = 1;
for(Integer i=6; i<10; i++){
    result = result * i;
}
System.debug(result);
```

Practice6: Do loops in class and Method.

```
public class loopClass{

    public static String reverseString(String str) {
        String reverseStr = '';
        for (integer i = str.length()-1; i >=0; i--) {
             reverseStr = reverseStr + str.substring(i, i+1);
        }
    return reverseStr;
    }
}
System.debug(loopClass.reverseString('Ahmet')); //temhA
```

Composite Data Types

Composite data types, also known as compound data types, are data types which may be included natively in a programming language along with built-in support or are types which can be defined and constructed by the developer. Composite data types are typically assembled from primitive data types and/or other composite types and creating our own composite type will be covered in the Classes section of this chapter. Collections, or aggregate data types, are also considered to be composite data types.

Collections

Collections are composite data types which allow the developer to aggregate, or collect, multiple other types into a single variable. The collection types available in Apex are List, Set, and Map.

1. **Arrays:** Arrays in Apex are basically the same as Lists in Apex. There is no logical distinction between the Arrays and Lists as their internal data structure and methods are also same.

What is the Array?

1) Array is a structure to store multiple data with the same data type
2) When you create an Array you have to declare
   i) Data Type of the elements
   ii) The number of the elements you want to store

➔**How to create an Array:** When we create an array we must declare the capacity. Int example it is 3

1.Way:Dynamic Declaration

```
String[] studentNames = new String[5];
System.debug(studentNames);//(null, null, null, null, null)
```

Note1: Default values for array elements will be "null" for every data type

Note2: Arrays use memory less and can be access faster because of that it is being preferable

Note3: Arrays are fixed in length. After declaring the length you cannot store elements more than the length

➔**How to add elements into an array**

```
studentNames[0] = 'Noah Ozay';
System.debug(studentNames);//(Noah Ozay, null, null, null, null)
studentNames[2] = 'Ayse Gul';
System.debug(studentNames);//(Noah Ozay, null, Ayse Gul, null, null)
studentNames[1] = 'Jack Jones’;
System.debug(studentNames);//(Noah Ozay, Jack Jones, Ayse Kan, null, null)
studentNames[5] = 'Shouyi Wang';
System.debug(studentNames); //it did not add that to into array since it has nothing on 5 index.
```

2.Way:Static Declaration: This is the shorter way to create array and add value.

```
String[] studentNames = new String[]{'Noah Ozay', 'Ayse Gul', 'Jack Jones'};
System.debug(studentNames);
```

➔ **How to get a specific value from an array**

```
String std3 = studentNames[2];
System.debug(std3);//Ayse Gul
```

➔ **How to add sObjects into an array**

```
Account a1 = new Account(Name='Checking', Phone='1234567890');
Account a2 = new Account(Name='Saving', Phone='2345678901');
Account[] acounts = new Account[]{a1, a2};
System.debug(acounts);//(Account:{Name=Checking, Phone=1234567890},
Account:{Name=Saving, Phone=2345678901
```

➔**How to add sObjects into an array**

```
Account a1 = new Account(Name='Checking', Phone='1234567890');
Account a2 = new Account(Name='Saving', Phone='2345678901');
```

Practice1: Print the difference between maximum and minimum values of the elements in an Array

```
Integer[] stdGrades= new Integer[]{76, 50, 80, 69, 80, 90};

➔Find the maximum value
Integer max = stdGrades [0];
for(Integer i = 1; i<grades.size(); i++){
    max = Math.max(max, stdGrades [i]);
}
System.debug(max);

➔Find the minimum value
Integer min = stdGrades [0];
for(Integer i = 1; i< stdGrades.size(); i++){
    min = Math.min(min, grades[i]);
}
System.debug(min);

System.debug(max - min);//40
```

Practice2: Print the number of characters of every element in a String Array

{'Noah', 'Brad Pitt' 'Mary', 'Tom Hanks', } ==> Noah= 4 Brad Pitt= 10 etc.

```
String[] names = new String[]{'Noah', 'Erhan', 'Tom Hanks', 'Brad Pitt', 'Jack Jones'};

//1.Way: for-loop
for(Integer i=0; i<names.size(); i++){
      System.debug(names[i] + '=' + names[i].length());
}
```

```
//2.Way: for-each-loop: It can be used just with Arrays, Lists, Sets
for(String w : names){
    System.debug(w + '=' + w.length());

}
```

Practice3: Print the first and last characters of every element in a String Array
{'Noah', 'Erhan', 'Tom Hanks', 'Brad Pitt'} ==> Nh En Ts Bt JJ

```
String[] names = new String[]{'Noah', 'Erhan', 'Tom Hanks', 'Brad Pitt',
'Jack Jones'};

for(String w : names){
    String firstChar = w.substring(0, 1);

    Integer lastIdx = w.length()-1;
    String lastChar = w.substring(lastIdx, lastIdx+1);

    System.debug(firstChar + lastChar);
}
```

Practice5: Find the multiplication of all elements in an Integer Array

```
Integer[] numArrays = new Integer[]{2, 5, 3, 4, 7, 8, 9};
Integer m = 1;
for(Integer w : numbers){
    m = m*w;
}
System.debug(m);
```

```
Practice6: [5, 0, 2, 0, 3] put the zeros to the end in the given array
Integer[] arr1 = new Integer[]{5, 0, 2, 0, 3, 6, 8, 0, };
Integer[] arr2 = new Integer[arr1.size()];

Integer firstIdxArr2 = 0;
Integer lastIdxArr2 = Arr2.size()-1;
for(Integer w : arr1){
    if(w!=0){
       brr[firstIdxArr2] = w;
       lastIdxArr2++;
    }else{
       brr[lastIdxArr2] = w;
       lastIdxArr2--;
    }
}
```

```
System.debug(arr2);
```

2. **List:** List can contain any number of records of primitive, collections, sObjects, user defined and built in Apex type. This is one of the most important type of collection and also, it has some system methods which have been tailored specifically to use with List. List index always starts with 0.

➔**How to create a List**

1.Way:

```
List<string> ListOfCities = new List<string>();
System.debug('Value Of ListOfCities'+ListOfCities); //()
```

2.Way: Declaring the initial values of list is optional.

```
List<string> ListOfStates = new List<string> {'NY', 'LA', 'LV'};
System.debug('Value ListOfStates'+ListOfStates);
```

➔Lets create new list and apply some common method **like add(), size(), get(), clear(), and set()** that are available to the list.

```
List<String> myList = new List<String>();
System.debug(myList);//()
```

➔**How to add elements into a list…** We use method add(), add(index, value) to add something to the list.

```
myList.add('Noah');
myList.add('Sibel');
myList.add('Asiye');
myList.add('Erhan');
System.debug(myList);
```

➔**How to add an element into a specific index; add(index, value)**

```
myList.add(1, 'X');
System.debug(myList);//(Noah, X, Sibel, Asiye, Erhan)
```

➔**How to add multiple elements (add list in another list): addAll(list_to_add)**

```
List<String> myListToAdd = new List<String>();
myListToAdd.add('A');
myListToAdd.add('B');
myListToAdd.add('C');
myList.addAll(myListToAdd);
System.debug(myList);//(Noah, X, Sibel, Asiye, Erhan,A, B, C)
```

➔How to create a new list from an existing list (How to clone a list)

```
List<String> myListShorterCloned = new List<String>( myList);
System.debug(myListShorterCloned); //((Noah, X, Sibel, Asiye, Erhan,A, B, C)
```

➔How to create a new List from Set(Set is used to store unique elements)

```
Set<String> mySet = new Set <String>(); //this syntax of creatin a set
mySet.add('www');
mySet.add('xxx');
mySet.add('yyy');
System.debug(mySet);//{www, xxx, yyy}
```

Note: If you try to put repeated elements into a Set, it does not give error, it accepts repeated elements just once

```
List<String> myListFromSet = new List<String>(mySet); // Syntax of creating list from set
System.debug(myListFromSet);//{www, xxx, yyy}
```

➔**How to sort list elements in alphabetical order: Use sort() and sort(DESC)**

```
List<String> myListShorter = new List<String>{'Ab', 'Xy', 'Wz' ,'Cd', 'Ef'};
System.debug('Before sorting: ' + myListShorter);
myListShorter.sort();
System.debug('After sorting: ' + myListShorter);
```

Example2: Type code to find the difference between maximum and minimum values in an Integer List.

```
List<Integer> myList = new List<Integer>{12, 5, 23, 1, 2, 6, 8,30};
myList.sort();
System.debug(myList);
Integer minValue = myList.get(0);
Integer maxValue = myList.get(myList.size()-1);
System.debug(maxValue - minValue);
```

➔**How to check if list elements are in alphabetical order**

Logic:

1)Create a new list from the given list

2)Sort the new list

3)Check if the given list equals to the new list

```
List<String> myGivenList = new List<String>{'Ab', 'Xy', 'Wz' ,'Cd', 'Ef'};
myNewList.sort(DESC);
System.debug(myNewList);
```

1)Create a new list from the given list

```
List<String> myNewList = new List<String>(myGivenList);
```

2)Sort the new list

```
myNewList.sort(DESC);
System.debug(myNewList);
```

3)Check if the given list equals to the new list

```
if(myGivenList.equals(myNewList)){
    System.debug('List elements are in alphabetical order');
}else{
    System.debug('List elements are not in alphabetical order');
}
```

➔How to sort list elements in descending order

Logic:

1)Create a new list from the given list

2)Sort the list elements of the new list in ascending order

3)Create one more empty list

4)By using for loop transfer the elements of sorted new list into the empty list from the last to the first

5)Compare the given list with the list created in the 3rd step

```
List<Integer> myGivenList = new List<Integer>{13, 10};
```

1)Create a new list from the given list

```
List<Integer> myNewList = new List<Integer>(myGivenList);
```

2)Sort the list elements of the new list in ascending order

```
myNewList.sort();
```

3)Create one more empty list

```
List<Integer> myAnotherList = new List<Integer>();
```

4)By using for loop transfer the elements of sorted new list into the empty list from the last to the first

```
for(Integer i = myNewList.size()-1; i>=0; i--){
    myAnotherList.add(myNewList.get(i)); //add each element that we get from myNewList to do myAnotherList
}
System.debug(myAnotherList);
```

5)Compare the given list with the list created in the 3rd step

```
if(myGivenList.equals(myAnotherList)){
    System.debug('The list elements are in descending order');
}else{
     System.debug('The list elements are in not descending order');
}
```

➔How to check if all elements are unique in a List

Logic:

1)Create a Set by using the given list

2)Compare the sizes of the List and the Set.

If the sizes are same, then the list has unique elements otherwise it has repeated elements

```
List<Integer> myGivenList = new List<Integer>{13, 10, 15, 13, 9, 10};
```

```
System.debug(myGivenList);
1)Create a Set by using the given list
Set<Integer> mySet = new Set<Integer>(myGivenList);
System.debug(mySet);
2)Compare the sizes of the List and the Set.
if(myGivenList.size() == mySet.size()){
    System.debug('All elements are unique in the given list');
}else{
    System.debug('All elements are not unique in the given list');
}
```

➔**How to remove all elements from a list:** Use clear () method to remove everything from a list.

```
List<Integer> myGivenList = new List<Integer>{13, 10, 15, 13, 9, 10};
System.debug(myGivenList);
myGivenList.clear();  //clear() makes the list empty list
System.debug(myGivenList); //()
```

➔ **How to remove an element in a specific index:** use remove(index) method we can put index of the elemet that we want to remove.

Note: In Apex, there is just one remove() method which works with index

```
List<Integer> myGivenList = new List<Integer> {13, 10, 15, 13, 9, 10};
System.debug(myGivenList);
myGivenList.remove(2);
System.debug(myGivenList); //{13, 10,  13, 9, 10}
```

Example 2: Type code to delete elements whose lengths are greater than 5 in a String List

```
List<String> myList = new List<String>{'Noah', 'Jack', 'Mehmet', 'Jimmy'};
for(Integer i=0; i<myList.size(); i
    if(myList.get(i).length()>5){
        myList.remove(i);
        i--;
    }
}
System.debug(myList);
```

➔**How to find the index of a specific element:** we can use index Of(value) method to find the index of specified element.

```
List<Integer> myGivenList = new List<Integer>{13, 10, 15, 13, 9, 10};
Integer idx = myGivenList.indexOf(15);
System.debug(idx); // 2
```

➔**How to check if a specific element exists in a list:** for that we can use contains(element) method

```
List<Integer> myGivenList = new List<Integer>{13, 10, 15, 13, 9, 10};
Boolean isExist = myGivenList.contains(13);
System.debug(isExist);
```

➔**How to update an existing element**: for that we use set(index, new_value) method

```
List<Integer> myGivenList = new List<Integer>{13, 10, 15, 13, 9, 10};
myGivenList.set(0, 50);
System.debug(myGivenList);//(50, 10, 15, 13, 9, 10)
```

➔**How to use sObjects in a List**

```
List<account> AccountToDelete = new List<account> (); //This will be null
System.debug('Value AccountToDelete'+AccountToDelete);
```

➔**One more difference between for-loop and for-each loop**

Example: Add 31 to the list if 9 exists in the list

```
List<Integer> myGivenList = new List<Integer>{15, 13, 9, 10};
for(Integer w : myGivenList){
    if(w==9){
        myGivenList.add(31); //Cannot modify a collection while it is being
iterated.
    }   }
```

NOTE: If you want to modify a collection, you cannot use for-each loop, you have to use for-loop

```
List<Integer> myGivenList = new List<Integer>{15, 13, 9, 10};
for(Integer i=0; i<myGivenList.size(); i++){
   if(myGivenList.get(i)==9){
        myGivenList.add(31);
    }
}
```

3. **Sets:** A Set is a collection type which contains multiple number of unordered unique records. A Set cannot have duplicate records. Most of the method that can be used with list also can be used in sets.

Example: We will be defining the set of products which company is selling.

```
Set<string> ProductSet = new Set<string>{'Iphone', 'Mac', 'Imac'};
System.debug('Value of ProductSet'+ProductSet);
```

➔Another way to create a Set and add elements

```
Set<Integer> mySet = new Set<Integer>{8, 12, 45};
System.debug(mySet);//{8, 12, 45}
```

```
//We can create Set by using List
List<String> a = new List<String>{'A', 'B', 'C', 'A', 'N', 'B'};

Set<String> b = new Set<String>(a);
System.debug(b);//{ 'C', 'N' }
```

➔Type code to remove elements from a Set if the elements exists in a List

```
Set<String> mySet = new Set<String>();
mySet.add('A');
mySet.add('K');
mySet.add('L');
mySet.add('M');
mySet.add('C');
System.debug(mySet);
List<String> myList = new List<String>{'K', 'M', 'C'};  //create new list
mySet.removeAll(myList); //use removeAll() method to remove all element of list
from a set
System.debug(mySet);//{A, L}
```

➔Type code to keep just common elements between a Set and a List

```
Set<String> mySet = new Set<String>();
mySet.add('A');
mySet.add('K');
mySet.add('L');
mySet.add('M');
mySet.add('C');
mySet.add('D');
List<String> myList = new List<String>{'K', 'X', 'M', 'D'}; //Creating of a new
list
```

retainAll() Method: checks the common elements then removes the uncommon ones then keeps the common ones.

NOTE: retainAll() comes from Set Class, because of that it can be used just with a Set

```
mySet.retainAll(myList);
System.debug(mySet);//{'K', 'M'}
System.debug(myList);
```

Example: How to check if two Sets have common elements or not

```
Set<Integer> a = new Set<Integer>{8, 12, 45};
Set<Integer> b = new Set<Integer>{120, 8, 12, 100};
a.retainAll(b);
System.debug(a);
```

```
if(a.size()==0){
    System.debug('No common elements');
}else{
    System.debug('There are ' + a.size() + ' common elements');
}
```

Example 2: Type code to remove the elements whose lengths are greater than 5 from a set

```
Set<String> mySet = new Set<String>{'Noah', 'Nuh', 'Adam', 'Nergiz', 'Victoria'};
System.debug(mySet);
for(String w : mySet){
if(w.length()>5){
mySet.remove(w);
}
}
System.debug(mySet);
```

Example 3: Type code to remove elements from the set if the elements exist in the list

```
List<String> myList = new List<String>{'a', 'e', 'a', 'd', 'e'};
System.debug(myList);
Set<String> mySet = new Set<String>{'a', 'b', 'c', 'd', 'e'};
System.debug(mySet);
mySet.removeAll(myList);
System.debug(mySet);
```

Example 4: Type code to keep just the common elements between a set and a list

```
Set<integer> mySet = new Set<integer>{1, 2, 3};
System.debug(mySet); // {1, 2, 3}
List<integer> myList = new List<integer>{1, 3, 5, 6};
System.debug(myList); // (1, 3, 5, 6)
mySet.retainAll(myList); // In simple words, it compares a set with another
set/list and check for common elements. if there is any common element between
them, it keeps the common elements and removes the uncommon ones.
System.debug(mySet); // {1, 3}
System.debug(myList); // (1, 3, 5, 6)
```

```
Example 5: How to see different elements between 2 Sets
Set<Integer> a = new Set<Integer>{8, 12, 45};
Set<Integer> b = new Set<Integer>{120, 8, 12, 100};
a.removeAll(b);
System.debug(a);
```

4. **Maps:** It is a key value pair which contains the unique key for each value. Both key and value can be of any data type.

   1)Maps use key-value structure

   2)"Key"s are unique

   3)"Value"s can be repatedly

➔**How to create (declare) a map**

```
Map<Integer, String> myMap = new Map<Integer, String>();
System.debug(myMap);// { }
```

➔**How to put elements(entry) into a map:** Use **put()** method to add elements in maps

```
myMap.put(1, 'Nuh Acar');
System.debug(myMap);//{1=Nuh Acar}
myMap.put(2, 'Mike Jack');
System.debug(myMap);
```

➔**How to use sObjects in a Map**

```
List<Account> accs = [SELECT Id, Name FROM Account]; //use SOQL to specify
what information to get
Map<Id, Account> myMap = new Map<Id, Account>(accs); //use the list that we
created using SOQL to create  Map.
System.debug(myMap);
```

➔**Create your own account objects in a list and use them in a Map**

```
List<Account> accs = new List<Account> ();
accs.add(new Account(id='0015e00000EKsyVAAT', Name='First', Phone='123'));
accs.add(new Account(id='0015e00000ElO4YAAV', Name='Second', Phone='456'));
accs.add(new Account(id='0015e00000ElO4ZAAV', Name='Third', Phone='789'));
Map<Id, Account> myMap = new Map<Id, Account>(accs);
System.debug(myMap);
```

➔**How to get just keys from a Map**

```
Set<String> keys = myMap.keySet();
System.debug(keys);
```

➔**How to get just values from a Map**

```
List<String> values = myMap.values();
System.debug(values);
```

➔**Find the sum of the number of the characters in keys**

```
Map<String, Integer> myMap = new Map<String, Integer>();
myMap.put('Noah', 13);
```

```
myMap.put('Seren', 14);
myMap.put('Victoria', 25);
myMap.put('Shouyi', 22);
Set<String> keys = myMap.keySet();
System.debug(keys);
Integer sum = 0;
for(String w : keys){
   sum = sum +  w.length();
}
System.debug(sum);
```

➔**Check if there is any repeated element in values of Map**

```
Map<String, Integer> myMap = new Map<String, Integer>();
myMap.put('Noah', 13);
myMap.put('Seren', 14);
myMap.put('Victoria', 25);
myMap.put('Shouyi', 22);
List<Integer> values = myMap.values();
System.debug(values); //(13, 14, 13, 22)
Set<Integer> valueSet = new Set<Integer>(values);
System.debug(valueSet); //(13, 14, 22)
if(values.size()==valueSet.size()){
    System.debug('No repetation');
}else{
    System.debug('No uniqueness'); }
```

```
Practice1:
➔Get the first and the last characters of the countries (France ➔ Fe
Map<String, String> myMap = new Map<String, String>();
myMap.put('Erhan Ozduran', 'USA');
myMap.put('Sibel Ates', 'Germany');
myMap.put('Seren Acar', 'France');
System.debug(myMap);

List<String> values = myMap.values();
List<String> firstAndLastChars = new List<String>();

for(String w : values){
    Integer lastIndex = w.length()-1;
    String first = w.substring(0,1);
    String last = w.substring(lastIndex, lastIndex+1);
    firstAndLastChars.add(first+last);
```

```
}
System.debug(firstAndLastChars);

➔ Get the first characters of first and last names
Map<String, String> myMap = new Map<String, String>();
myMap.put('Erhan Ozduran', 'USA');
myMap.put('Sibel Ates', 'Germany');
myMap.put('Seren Acar', 'France');
System.debug(myMap);

Set<String> keys = myMap.keySet();
LIst<String> initials = new List<String>();

for(String w : keys){
    String initialFirst = w.substring(0,1);
    String initialLast = w.split(' ')[1].substring(0,1);
    initials.add(initialFirst + initialLast);
}
System.debug(initials);

Practice2:
Type code to count the number of alphabetical characters in a String
➔Noah==> N=1, o=1, a=2, h=1

String str = 'Apex is easy so far';
String justAlphabets = str.replaceAll('[^A-Za-z]', '');
System.debug(justAlphabets);
String[] alphabets = justAlphabets.split('');
System.debug(alphabets);
Map<String, Integer> numOfOccurrence = new Map<String, Integer>();
for(String w : alphabets){
    Integer n = numOfOccurrence.get(w);
    if(n==null){
       numOfOccurrence.put(w, 1);
    }else{
       numOfOccurrence.put(w, n+1);
    }
}
System.debug(numOfOccurrence);

Practice3:
Type code to count the number of words in a String
'Apex is easy and Apex is fun, I liked it'==> Apex=2, is=2, easy=1…
```

➔p{Punct}: regex represents all punctuation marks

```
String str = 'Apex is easy so far';
String updatedstr = s.replaceAll('\\p{Punct}', '');
String[] words = updatedstr.split(' ');
System.debug(words);
Map<String, Integer> numOfOccurrence = new Map<String, Integer>();
for(String w : words){
    Integer n = numOfOccurrence.get(w);
    if(n == null){
        numOfOccurrence.put(w, 1);
    }else{
        numOfOccurrence.put(w, n+1);
    }
}
System.debug(numOfOccurrence);
```

Practice4:
Type code to check if a List has repeated elements or not by using maps

```
List<Integer> myList = new List<Integer>{12, 21, 12, 13, 12, 21, 35};
Map<Integer, Integer> myMap = new Map<Integer, Integer>();
for(Integer w : myList){
    Integer numOfOccurence = myMap.get(w);
    if(numOfOccurence==null){
        myMap.put(w, 1);
    }else{
        myMap.put(w, numOfOccurence+1);
    }
}

System.debug(myMap);//{12=3, 13=1, 21=2, 35=1}
```

Principles of Object-Oriented Programming Language

- Encapsulation
- Inheritance
- Polymorphism
- Abstraction

1) Encapsulation: Encapsulation is "Hiding Data"

➔How can you hide data? ➔ I use "private" access modifier for the data

➔After hiding data, how can you use it from other classes?

I use "getter methods" to read data, "setter methods" to update data

Example: Lets create class that called student with two construction:

```
public class Student { //Creation Student Class
    public String name; // we use public modifier so variable can be accessed from
within our org.
    public Integer age;
    public String id;
    public Student(){ // constructor without parameters
        this.name = 'Noah';
        this.age = 30;
        this.id = 'N30';
    }
    public Student(String name, Integer age, String id){ //Constructor2 with
parameters
        this.name = name;
        this.age = age;
        this.id = id;
    }
}
```

Note ➔**This keyword**: You can use the this keyword in dot notation, without parenthesis, to represent the current instance of the class in which it appears. Use this form of the this keyword to access instance variables and methods.

➔ lets create instance of student class and call variables that we created inside class from anonymous window;

```
Student std1 = new Student(); //creating class instance using constructor without
parameters.
System.debug(std1.name + ', '+ std1.age+', '+std1.id); //Noah, 30, N30
Student std2 = new Student('Seren', 25, 'S25'); //creating class instance using
constructor with parameters.
System.debug(std2.name + ' '+ std2.age+' '+std2.id); //Fatma, 25, F25
```

Note: As you can see, we can call variables and set new values to them with just using instance of class.

Example2: Lets convert variables in Student class to private.

```
public class Student { //Creation Student Class
    private String name; // we hide our data 'private'
    private Integer age;
    private String id;
    public Student(){ // constructor without parameters
        this.name = 'Noah';
        this.age = 30;
        this.id = 'N30';
    }
    public Student(String name, Integer age, String id){
        this.name = name; //We called apex in my class there is name
variable update that one with new name.
        this.age = age;
        this.id = id;
    }
}
```

➔lets create instance of student class and call variables that we created inside class from anonymous window;

```
Student std1 = new Student(); //creating class instance using constructor without
parameters.
System.debug(std1.name + ', '+ std1.age+', '+std1.id); // Variable is not visible:
Student.name
```

➔We need to create get methods to get data and set method to update data that are private in class. Lets create get() and Set() method in Student class;

```
public class Student { //Creation Student Class
    private String name; // we hide our data 'private'
    private Integer age;
    private String id;
    public Student(){ // constructor without parameters
        this.name = 'Noah';
        this.age = 30;
        this.id = 'N30';
    }
    public Student(String name, Integer age, String id){
        this.name = name;
```

```
        this.age = age;
        this.id = id;
    }
//To create getter
public String getName(){ //1)Use public as access modifier
    return this.name;   //2)Make the return type of the method same with the
data type of the variable
}
public Integer getAge(){
    return this.age;    //3)Use  return this.<variable name>; inside the
method
}
public String getId(){
    return this.id;
}

//To create setter
public void setName(String name){ //1)Use public as access modifier
    this.name = name;        //2)Return type will be "void" everytime since
we are updating data not creating new data.
}                   //3)Put parameter inside the method parenthesis
public void setAge(Integer age){
    this.age = age;
}
public void setId(String id){
    this.id = id;
}
}
```

➔ Now we can call variables with get() method and update variables with set() methods. Let's do it in from anonymous window;

```
Student std1 = new Student();
System.debug(std1.getName()); //Noah
System.debug(std1.getAge()); //30
System.debug(std1.getId());   //N30
Note: If we do not write set() method, we cannot update the value of std1,
with the help of setter method we can change the value of std1.
std1.setName('Seren');
System.debug(std1.getName()); //Seren
```

```
std1.setAge(25);
System.debug(std1.getAge()); //25
std1.setId('S25');
System.debug(std1.getId()); //S25
```

Note:➔ If you do not create getters nobody can read the data from outside the class

➔If you do not create setters nobody can update your data

➔If you do not create any setter in a Class after making all variables private, it means no data in the class can be updated. That kind of classes are called **"immutable classes"**

2) Inheritance: Benefits of Inheritance

- It stops repetation
- It makes update and maintenance easy
- It makes our code more readable
- It makes our codes run faster
- It makes our codes more reusable

Example: Lets create a parent class called animal and two child class named lions and birds,

Step1: if you want to your classes inherit from a specific class you must use keyword **virtual** (use virtual keyword in parent class)

Step1: You must use **extends** keyword following parent class name (childClassName Extend prentClassName) which means that that child class is extending from parent.

➔Parent Class

```
public virtual class allAnimals{  //If you want other classes inherit from
Animal class use 'virtual' keywords.
    public void drink (){
        system.debug('Anilmals should drink water...');
    }
    public void eat (){
        system.debug('Anilmals should eat...');
    }
    public static void breathe(){
        system.debug('Animals should breathe...');
    }
}
```

➔Child1 Class

```
public class Lions extends AllAnimals { // 'Lionss extend AllAnimals'  Lions
is child [sub_class] and AllAnimals is parent
```

```
    public void Roar(){
        System.debug('Lion Roars..');
         }
}
```

➔ **Child2 Class**

```
public class Birds extends AllAnimals { //in 'cats extends Animals' cats is
child and Animals is parent [super_class].
    public void sing(){
        System.debug('Birds sing..');
    }
}
```

➔Let's use anonymous window to test classes;

```
Lions l1 = new Lions(); //create instance of lionsnew class called d1
l1.roar();  // Lions Roars… ➔ called the roar() method in lions class
l1.eat(); //Animals should eat…➔Called the eat() method in AllAnimals (parent)
class by using instance of lions class.
l1.drink(); //Animals should drink water…➔ Called the eat() method in AllAnimals
(parent) class by using instance of lions class.
```

Note: breathe () method is a static method in a parent class lets called like we did above with instance of child class.

```
l1.breathe(); //Static method cannot be referenced from a non static context:
```

Important: As you can see it give an error. To call static method parent class, do not use objects to call static methods. You can use parent or child class names to call static methods.

➔To call static methods you can use "parent class name" or "child class name" both work.

```
AllAnimals.breathe();  // Animals breathe...
Lions.breathe();  // Animals breathe...
```

Note 1: Apex does not support multiple inheritance. Multiple inheritance means multiple parents for a child class.

Note 2: Apex supports multi-level inheritance. Multi level inheritance means child -> parent -> grand parent →

Note 3: Apex supports hierarchical inheritance. Hierarchical inheritance means a parent can have multiple child classes.

```
Birds.breathe(); // Animals breathe...
AllAnimals.breathe();  // Animals breathe...
Birds b1 = new Birds();  //create object of birds class
```

```
b1.sing(); // Birds sing… calling bird() method of birds class.
b1.eat(); //Animals should eat… Calling eat() method of prent class by object of child class
b1.drink(); //Animals should drink water.. Calling drink() method of parent class by object of child class
```

3) Polymorphism: Overloading + Overriding

Example: Lets update the birds class that we used before

```
public class birds extends AllAnimals { //in 'birds extends Animals' birds is child and Animals is parent
    public void sing(){
        System.debug('birds sing..');
    }
    Private static Integer add(Integer a, Integer b){ // we create method called add() with two parameters
    return a+b;
    }
     Private static Integer add(Integer a, Integer b, Integer c){ //we create another method with same name add() with 3 parameters. This situation called overloading.
    return a+b+c;
    }
    public static void methodCall(){ //private method can be overloading we use two private add() method above in that class
        System.debug(add(3,5)+add(3,5,2));
    }
}
```

➔**Overloading:** Creating methods whose names are same, parameters are different

System.debug(Birds.add(3, 5));//8 ➔this code used add() method with two parameters.

*System.debug(Birds.add(3, 5, 2));//*10 ➔this code used add() method with three parameters.

➔private methods can be overloaded because overloading is done in a single class

➔static methods and non-static methods can be overloaded

```
Cats.methodCall();//18
```

➔ **Overriding:** If you change the body of a method after inheriting, it is called "Method Overriding".

Overriding

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }
}
class Hound extends Dog{
    public void sniff(){
        System.out.println("sniff ");
    }

    public void bark(){
        System.out.println("bowl");
    }
}
```

Same Method Name, Same parameter

Overloading

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }

    //overloading method
    public void bark(int num){
        for(int i=0; i<num; i++)
            System.out.println("woof ");
    }
}
```

Same Method Name, Different Parameter

Why we need overriding?

When we ask is Lions eat? The answear will yes and since we do not have eat() in lions class it will go animal class and will print 'Animals eat'. That is not good answer we want to give more specific answer we should says 'lions eat only fresh meat'. To do that we will change the body of method for each sub-classes.

How to Override a method?

1)Use "**virtual**" keyword for the method which you want to override in parent class

2)Type the method without using "virtual" keyword in child class

3)Change the body however you need

4)Put "**Override**" keyword at the top of the method in child class

Note 1: Static methods cannot be overridden because they are common for all objects

Note 2: private methods cannot be overridden because you cannot see them from the child class

Note 3: protected methods can be overridden from child classes and from inner classes

Example:

Step1: To override a method in a parent class we need to use **virtual** keyword. Lets override the eat() method in a animals class.

Eat() method from parent (Animals) Class:

```
public virtual void eat (){ //to override that method in child class we use virtual key words
    system.debug('Anilmals eat...');
}
```

Step2: copy that method in a child class without virtual keyword and put **override** keyword on top of that method.

Step3: change the body of the method to what you want for each child class

Eat() method from lions class:

```
Override //we override eat() method from parent class
public void eat (){
        system.debug('Lions eat only fresh meat...');  //we changed the body
of method to 'Lions eat only fresh meat...' instead of ‘Animal eat’
}
```

Eat() method from birds class:

```
Override //we override eat() method from parent class
public void eat (){
        system.debug('birds eat grains and grass...');  //we changed the
body of method to 'birds eat grains and grass...'instead of ‘Animal eat’
}
```

➔ **let’s call that method from anonymous window:**

```
lions l2 = new lions();
l2.eat(); // 'Lions eat only fresh meat...'\ as you can see instead of print
animals eat we make it more specific.
birds b2= new birds();
b2.eat(); //'birds eat grains and grass...'\ as you can see instead of print
animals eat we make it more specific.
```

4) **Abstraction:** If you create a method which has no body it is called "abstract method" ➔"abstract methods" can be created just in "abstract classes"

➔How to create "abstract methods"

1) Be inside an abstract class

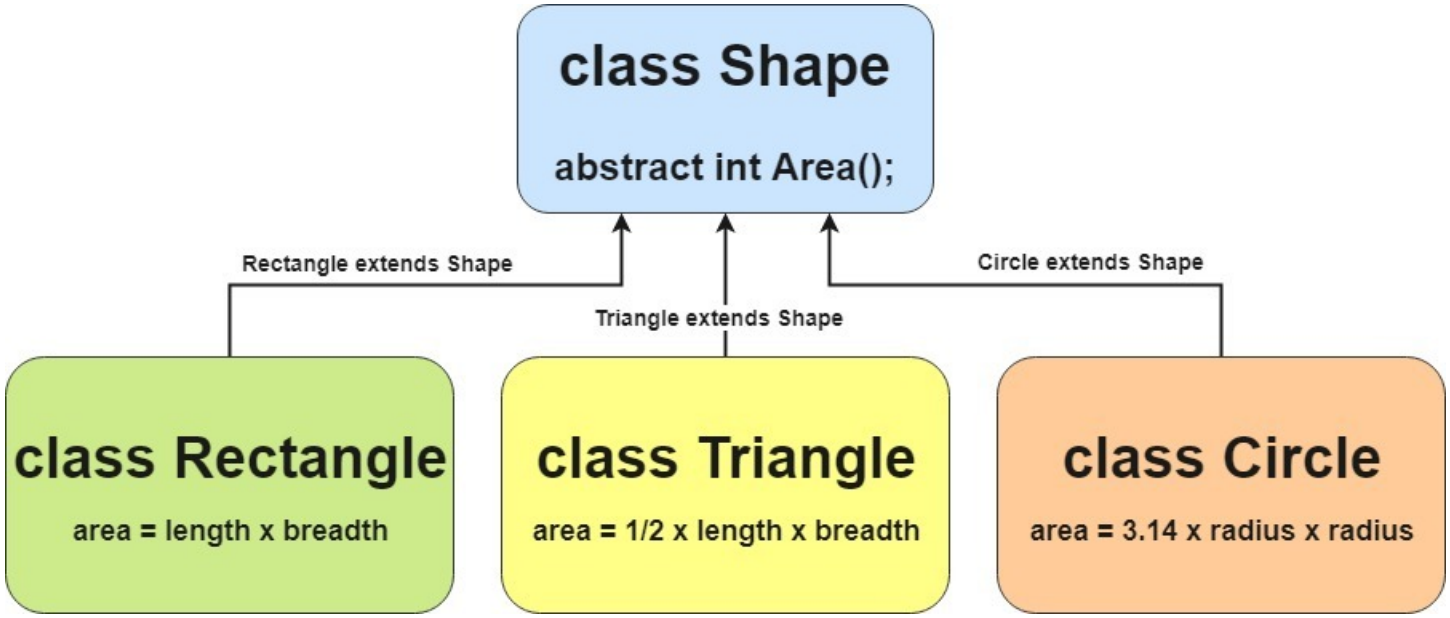
2) Type code tp create method except the body

3) Use "abstract" keyword between the access modifier and the return type

➔**Can I put "concrete method" inside an "abstract class"?**

Yes, "abstract class" can have both "concrete method" and "abstract method"

Note: Do not use "abstract" keyword and "method body" together in a method. If you use you will get error



Example:

```
public abstract class Shapes { //to create abstract class we write abstract keyword after acc. modifier
    public abstract void calculateArea(); // we did not put {} since we dont write body in abstract method
    public void width(){ //abstract classes can have both concreate and abstract methods.
        system.debug('The width of each shape is different');
    }
}
```

➔ lets create child class from Shapes class called circle

```
public class Circle extends Shapes{ // Circle extend Mammals class which mean it child of Shapes class
     Override
    public void calculateArea(Decimal radius){ //we must implement calculateArea () method from parent class in child class since it is abstract
        System.debug('The area of give circle is ' + '(3.14*r*r)');
} }
```

➔Why do we need abstract methods?

1)Sometimes we do not use body of the method which is in parent class for any child class. So, the body is being unusable code, no need to type unusable code

2)Sometimes we want to make some functionalities mandatory for all child classes. To make it we make the method abstract in parent class.

➔What are the differences between "virtual" and "abstract" classes?

1)You can create objects from "virtual" classes but you cannot create objects from "abstract" classes

Note: Abstract classes have constructors but constructors are not used to create objects

➔ let's see difference by construct object of class from anonymous window:

Animals a1 = new Animals();

a1.eat(); // Animals eat… ➔as you can see we were able to construct object of virtual class

Mammals m1 = new Mammals(); // Abstract classes cannot be constructed

2)When you create parent-child relationship by using "abstract class", all abstract methods must be overridden by concrete child classes.

When you create parent-child relationship by using "virtual class", there is no any mandatory overriding because "virtual class" cannot have "abstract methods"

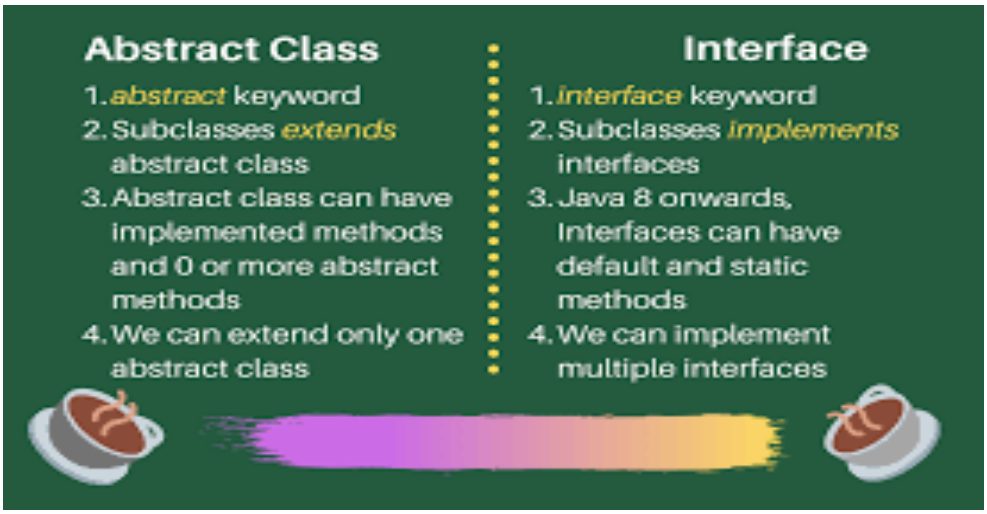
Interface:

➔1) We need interfaces to be able to create multiple parents for child classes

➔2) All methods in an interface are "abstract methods" because of that all methods in an interface must be overridden by concrete child classes

➔3) All methods in an interface are public as default

➔4) All methods in an interface are static as default because interfaces do not have constructor so you cannot create object by using interfaces. If you cannot create object you cannot access to non-static methods. Therefore, all methods in an interface are static.



➔5) Interfaces are not classes. They cannot store any variables and they cannot be used to call methods

Example: Lets create two interface and another class that implements them.

Step1: to create implemets just change Class keyword with Implemets

```
public Interface AirConditions {. //First Interfaca
    //Even we did not type public, static, and abstract it is abstract
public and static by default.
    void cool(); //even if you write abstract or public apex will complain
since it already public, abstract, and static by default.
}
public Interface Breaks { //Second Interface
     void secureBreak();
}
```

Step2: We need to put Implements keyword after class name and write interfaces name that we want to implement

```
public class HondaCivic Implements AirConditions, Breaks { //we implement
AirCondition and Breaks classes
   public void cool(){ //no need to write override keyword since it is
interface by default apex know it.
        System.debug('Cool well...');
    }
   public void secureBreak(){
        System.debug('Stopped Securely...');
    }  }
```

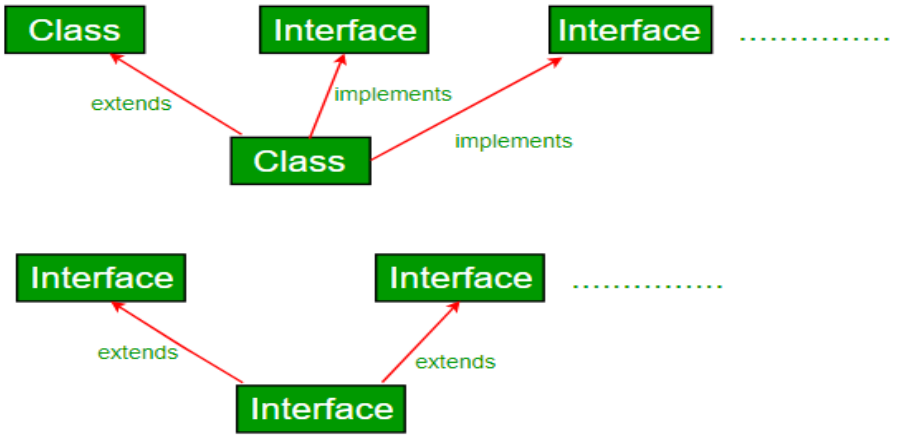
➔Lets call cool() method from HondaCivic class from anonymous window

```
HondaCivic hc1 = new HondaCivic();
hc1.cool();  // Cool well...
```

Note1: A class can inherited (extend) only one class that is why we need interface,

Note1: A Class can implement as many as interface needed (multiple inheritance).

Note3: A interface can extend other interfaces.



DQL (Data Query Language) Queries

- By queries, we can fetch data from Salesforce itself to the Apex side.
- Queries are essential if we are to read and modify data that already exists in Salesforce.
- We have two types of queries: SOQL and SOSL. Each are useful in different scenarios.

SOQL (Salesforce Object Query Language)

This is Salesforce Object Query Language designed to work with SFDC Database. It can search a record on a given criterion only in single sObject.

➔ Developer console>File>Open Resource> Object you want to open > Select the field values you want to retrieve> Query [it creates the syntax] > Execute

➔**Syntax is like that:** ***SELECT*** *field* ***FROM*** *object(API name)*

Note 1: Every SOQL Query should have SELECT and FROM

Note 2: We use "FIELD NAME" instead of "FIELD LABEL" in SOQL Queries

Note 3: We use API of the sObject

Example: Retrieve name, company, rating and lead status from the lead object.

SELECT Name, Company, Rating__c, Status FROM Lead

WHERE key word in SOQL: In order to filter out unwanted records we use the **WHERE** key word after your object, to describe more specifically which records should return.

Example: Fetch the leads that are still open and not contacted.

SELECT *Name, Company, Rating__c, Status* ***FROM*** *Lead* ***WHERE*** *Status='Open - Not Contacted'*

➔**1) AND OR:** We can also filter out our query with multiple conditions using AND OR between them.

Example:

SELECT *Name, Company, Rating__c, Status* ***FROM*** *Lead* ***WHERE*** *Status = 'Closed - Converted'* ***AND*** *(Rating__c = 'Warm'* ***OR*** *Rating__c = 'Cold')*

SELECT *Name, Company, Rating__c, Status, LeadSource* ***FROM*** *Lead WHERE Status = 'Closed - Converted'* ***OR*** *LeadSource='Web'*

➔**2) IN:** If we have multiple possible values for the same field such as the status is hot or cold or warm, we can filter it with using the **IN** keyword after the **WHERE** keyword and field name.

Example:

SELECT Name, Company, Rating__c, Status, LeadSource FROM Lead WHERE Status IN ('Open - Not Contacted', 'Working - Contacted') OR LeadSource='Web'

➔**3) LIKE:** If we have multiple values starting with the same word or include the same word such as 'Closed – Converted' and 'Closed – Not Converted'; we can use the **LIKE** keyword. Like keyword matches partial text string with the support of **wildcards.**

'%' wildcards: Means zero or more characters before or after the word it is attached to.

Example:

SELECT *Name, Company, Rating__c, Status, LeadSource* ***FROM*** *Lead* ***WHERE*** *Status* ***LIKE*** *'Closed****%****'*

'_' wildcard: matches partial text string which can have any random single character before or after the character it is attached to anywhere in the string.

Example:

SELECT Name, Company, Rating__c, Status, LeadSource FROM Lead WHERE Name LIKE '_a%'

Note: The main difference is the **IN** operator is used when we know exactly what match we are looking for, **LIKE** operator is used we are not sure what exactly, but we give an estimate such as it should include a value starting with these characters etc.

SOQL Query Questionsc

1) Select Name and Phone number for all records

SELECT Name, Phone FROM Account

2) Select Name, Phone number, and NumberOfEmployees for the records whose number of employees is greater than 500

SELECT Name, Phone, NumberOfEmployees FROM Account WHERE NumberOfEmployees > 500

3) Select Name, Phone number, NumberOfEmployees for the records whose number of employees is greater than 500 OR Rating is cold

SELECT Name, Phone, NumberOfEmployees, Rating FROM Account WHERE NumberOfEmployees>500 OR Rating='Cold'

4) Select Name, Phone number, NumberOfEmployees for the records whose number of employees is greater than 500 AND Rating is cold

SELECT Name, Phone, NumberOfEmployees, Rating FROM Account WHERE NumberOfEmployees>500 AND Rating='Cold'

5) Select Name, NumberOfEmployees, Rating for the records whose number of employees is greater than 500 OR less than 300 AND Rating is Cold

SELECT Name, Phone, NumberOfEmployees, Rating FROM Account WHERE (NumberOfEmployees>500 OR NumberOfEmployees<300) AND Rating = 'Cold'

6) Select Name, Rating for the records Rating is Cold OR Warm

1.Way: SELECT Name, Rating FROM Account WHERE Rating = 'Cold' OR Rating = 'Warm'

2.Way: SELECT Name, Rating FROM Account WHERE Rating IN ('Cold', 'Warm')

7) Select Name, Rating for the records Rating is not Cold, not Warm

SELECT Name, Rating FROM Account WHERE Rating NOT IN ('Cold', 'Warm')

8) Select Names which start with 'U'

SELECT Name FROM Account WHERE Name LIKE 'U%'

9) Select Names whose second character is 'i'

SELECT Name FROM Account WHERE Name LIKE '_i%'

10) Select Names which have 'x' in any position

SELECT Name FROM Account WHERE Name LIKE '%x%'

11) Select Names whose 3rd character is 'o' it has 6 characters in total

SELECT Name FROM Account WHERE Name LIKE '__o___'

12) Select Names whose last 3 characters are 'ort'

SELECT Name FROM Account WHERE Name LIKE '%ort'

Sorting Records that We Fetched by SOQL

We can also sort the records we retrieved in an ascending or descending order by putting **ORDER BY** after your object and using a field name and **ASC** or **DESC** keywords. We generally use to filter date, number or text fields(alphabetically). By default, it sorts in ascending order so we don't have to use **ASC** but if we want to sort it descending from highest value to the lowest we need to use **DESC** key word.

Exercises:

SELECT Name, Company, Rating__c, Status, LeadSource FROM Lead ORDER BY Name

➔by default it is in an ascending order (starting from A, goes to Z).

SELECT Name, Company, Rating__c, Status, LeadSource FROM Lead ORDER BY Name ASC

SELECT Name, Company, Rating__c, Status, LeadSource FROM Lead ORDER BY Name DESC

SELECT Name, Company, Rating__c, Status, LeadSource, CreatedDate FROM Lead ORDER BY CreatedDate DESC

NULLS FIRST & NULLS LAST keywords:

We can also use NULLS FIRST or NULLS LAST keywords to put the null values to the beginning or end of our list.

Example:

SELECT Name, Company, Rating__c, Status, LeadSource, AnnualRevenue FROM Lead ORDER BY AnnualRevenue NULLS LAST

LIMIT AND OFFSET Keywords

➔You can limit the number of records you retrieve by adding the **LIMIT** keyword after the object.

➔ OFFSET is used to point on a particular row. If you say OFSET 5, it will skip the first five values and retrieve the rest.

SELECT Id, Name, Industry, Number_of_Contacts__c FROM Account LIMIT 10 (this will give you only the first 10 accounts)

SELECT Name, Company, Rating__c, Status, LeadSource, AnnualRevenue FROM Lead ORDER BY AnnualRevenue NULLS LAST LIMIT 20

SELECT Name, Company, Rating__c, Status, LeadSource, AnnualRevenue FROM Lead ORDER BY AnnualRevenue DESC LIMIT 5

SELECT Name, Company, Rating__c, Status, LeadSource, AnnualRevenue FROM Lead ORDER BY Name DESC OFFSET 5

SELECT Name, Company, Rating__c, Status, LeadSource, AnnualRevenue FROM Lead ORDER BY Name DESC LIMIT 10 OFFSET 5 (Skip first 4 values, give me the 10 records after that.)

Exercises:

13) Select Names, Phone Numbers and sort the records in alphabetical order by using names

SELECT Name, Phone FROM Account ORDER BY Name

14) Select Names, Phone Numbers and sort the records in descending order by using names

SELECT Name, Phone FROM Account ORDER BY Name DESC

15) Select Names which are starting with 'U' containing 'a' and sort the records in descending order by using names

SELECT Name FROM Account WHERE Name LIKE 'U%a%' ORDER BY Name DESC

16) Select Names which are not starting with 'U' and sort the records in descending order by using names

SELECT Name FROM Account WHERE NOT Name LIKE 'U%' ORDER BY Name DESC

17) Select 5th Name after sorting in alphabetical order by using names

SELECT Name FROM Account ORDER BY Name LIMIT 1 OFFSET 4

Fields(ALL), Fields(Custom) and Fields(Standard) (This is new update released on 2021 Spring)

FIELDS(ALL) - This fetches all the fields of an object at one go. This is similar to * operator in SQL query.

FIEDLS(STANDARD) - This can be used to fetch all the standard fields of an object at one go.

FIELDS(CUSTOM) - This is use to fetch all the custom fields alone on an object.

Example:

SELECT FIELDS(ALL) FROM Account LIMIT 200

SELECT Name, Id, FIELDS(CUSTOM) FROM Account LIMIT 25

SELECT Subscription__c, FIELDS(STANDARD) FROM Account

Aggregate Functions (MIN(), MAX(), AVG(), SUM(), COUNT() etc.)

SOQL does have aggregate function as we have in SQL. Aggregate functions allow us to roll up and summarize the data.

AggregateResult []: Note that any query that includes an aggregate function returns its results in an array of AggregateResult objects. AggregateResult is a readonly sObject and is only used for query results. It is useful when we need to generate the Report on Large data.

Exercises Continue…

18) Select minimum number of employees from Account

1.Way: SELECT NumberOfEmployees FROM Account ORDER BY NumberOfEmployees NULLS LAST LIMIT 1

2.Way: SELECT MIN(NumberOfEmployees) FROM Account

19) Select maximum number of employees from Account

1.Way: SELECT NumberOfEmployees FROM Account ORDER BY NumberOfEmployees DESC NULLS LAST LIMIT 1

2.Way: SELECT MAX(NumberOfEmployees) FROM Account

20) Find the average number of employees

SELECT AVG(NumberOfEmployees) FROM Account

21) Find the sum of the number of employees in Account

SELECT SUM(NumberOfEmployees) FROM Account

Group By Keyword: GROUP BY clause is used in SOQL query to group set of records by the values specified in the field. We can perform aggregate functions using GROUP BY clause.

22) Find the total number of employees per Rating

SELECT SUM(NumberOfEmployees), Rating FROM Account GROUP BY Rating

23) Find the average annual revenue per Industry

SELECT AVG(AnnualRevenue), Industry FROM Account GROUP BY Industry

24) Select the name and number of employees of the Account whose number of employees is the highest

SELECT Name, NumberOfEmployees FROM Account ORDER BY NumberOfEmployees DESC NULLS LAST LIMIT 1

25) Select the name and number of employees of the Account whose number of employees is the second highest

SELECT Name, NumberOfEmployees FROM Account ORDER BY NumberOfEmployees DESC NULLS LAST LIMIT 1 OFFSET 1

26) Select the name and number of employees of the Account whose number of employees is the third lowest

SELECT Name, NumberOfEmployees FROM Account ORDER BY NumberOfEmployees NULLS LAST LIMIT 1 OFFSET 2

27) Find the number of accounts whose number of employees is less than 2000

SELECT COUNT(NumberOfEmployees) FROM Account WHERE NumberOfEmployees < 2000

28) Find the number of accounts per Industry

SELECT count(name), Industry FROM Account GROUP BY Industry

Note: ➔WHERE and HAVING both are for filtering. But HAVING can be used after GROUP BY, WHERE can be used without GROUP BY

➔HAVING can be used with aggregate functions and operators

29) Find the number of accounts per Industry. The number of accounts should be greater than 1

SELECT count(name), Industry FROM Account GROUP BY Industry HAVING count(Name)>1

Working with Date and DateTime in SOQL

Date format = YYYY-MM-DD

DateTime format = SOQL support 3 datetime formats including time zones

YYYY-MM-DDThh:ss+hh:mm

YYYY-MM-DDThh:ss-hh:mm

YYYY-MM-DDThh:mm:ssZ

➔You can use **Date Literal** as DateTime in SOQL TODAY, YESTERDAY, TOMORROW, LAST_WEEK, NEXT_MONTH, THIS_QUARTER [Fiscal year], NEXT_YEAR etc.

LAST_90_DAYS, NEXT_90_DAYS, LAST_N_DAYS:n, NEXT_N_MONTHS:n, LAST_N_WEEKS:n, NEXT_N_YEAR:n

Example:

Step1: Sorts the leads by their last modified date in a descending order starting from the most recently updated one.

SELECT Name, Company, Rating__c, Status, LastModifiedDate FROM Lead ORDER BY LastModifiedDate DESC

Step2: If you want to see only the account that are updated in the last 60 days, you can use the following syntax:

SELECT Name, Company, Rating__c, Status, LastModifiedDate FROM Lead **WHERE** LastModifiedDate =**LAST_N_DAYS:60**

Step3: Show the lead that updated today only.

SELECT Name, Company, Rating__c, Status, LastModifiedDate FROM Lead WHERE LastModifiedDate > YESTERDAY

Exercises Continue…

30) Find the accounts created after 2020-10-08 in Account

SELECT name, CreatedDate FROM Account WHERE CreatedDate > 2021-06-08T23:59:59Z

31) Find the accounts created YESTERDAY

SELECT name, CreatedDate FROM Account WHERE CreatedDate = YESTERDAY

Practice:

Step 1: Write a SOQL query to retrieve all Contacts from a Salesforce Org. Retrieved results should show Contact Name, Title, Phone and Email.

SELECT Name, Title, Phone, Email FROM Contact

Step 2: Modify SOQL query to retrieve only those contacts with title ‘VP, Technology’.

SELECT Name, Title, Phone, Email FROM Contact WHERE Title='VP, Technology'

Step 3: Modify SOQL, and add another field called ‘Department’ in the results. This field should be the 2nd field in the result.

SELECT Name, Department, Title, Phone, Email FROM Contact WHERE Title='VP, Technology'

Step 4: Modify SOQL, retrieve all results satisfying step 2 condition and has department value as ‘Finance’

SELECT Name, Department, Title, Phone, Email FROM Contact WHERE Title='VP, Technology' AND Department='Finance'

Step 5: Modify SOQL, include all SVP and VP in your search results. Make sure your result still meets step 4 condition.

SELECT Name, Department, Title, Phone, Email FROM Contact WHERE Title LIKE '%VP%' AND Department='Finance'

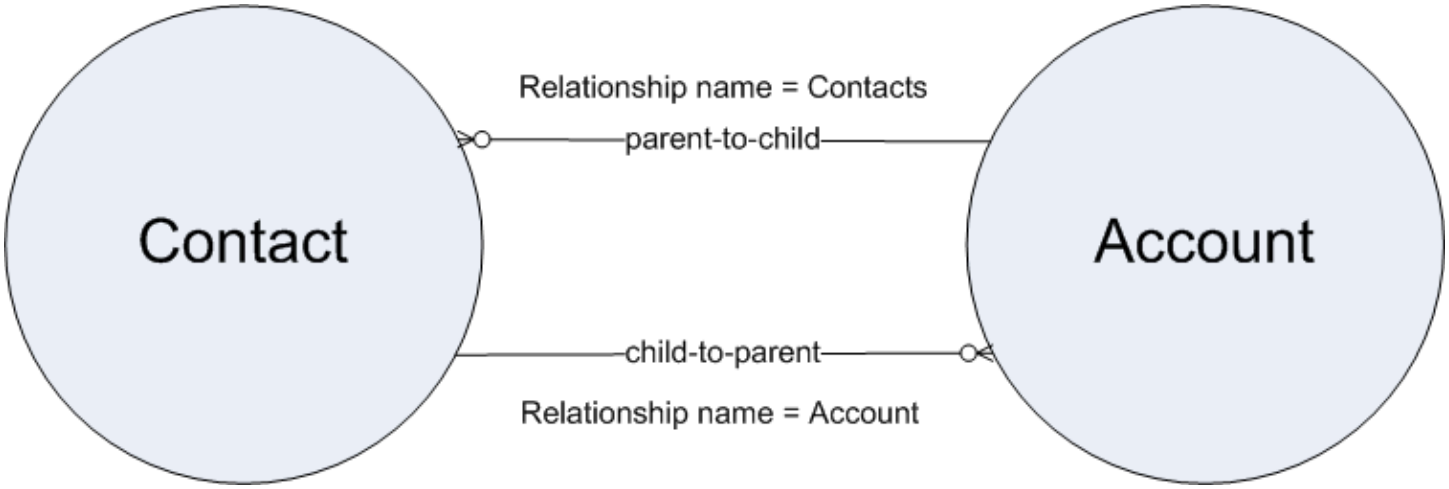
Step 6: Sort results by name in descending order.

SELECT Name, Department, Title, Phone, Email FROM Contact WHERE Title LIKE '%VP%' AND Department='Finance' ORDER BY Name DESC

Step 7: Limit search result to only 3.

SELECT Name, Department, Title, Phone, Email FROM Contact WHERE Title LIKE '%VP%' AND Department='Finance' ORDER BY Name DESC LIMIT 3

Relationship SOQL Queries



Parent object has the data, and the child object will have the lookup field which will refer those data. Hence, whichever object you create a lookup field will be the child object and the data which it refers to is considered as the parent object.

For instance, you can query fields from the Account object. And as the Contact object is child of Account, you can query the fields of contacts of these accounts too. This will be parent to child relationship query. Or if you are querying about the Contact object, you can query about the Account object related to these contacts. This will be child to parent relationship query.

1. Parent to Child SOQL

The main object for the query is the parent object and you fetch related child records.

Example: SELECT Name, (SELECT Name, Phone FROM **Contacts**) FROM Account

➔ Contacts here is the name of relationships.

Note: To get the relationship name, go to the child object (on object manager), go to the relationship field, click on it and you will see “Child Relationship Name”.

Exercise Continue…

32) Find the contact first names and last names for every account. (Relationship name is Contacts)

How

33) Find the contact first names and last names for every account if the first name of the contact is starting with A. (Relationship name is Contacts)

SELECT Account.Name, (SELECT Contact.FirstName, Contact.LastName FROM Account.Contacts WHERE Contact.FirstName LIKE 'A%') FROM Account

34) Find the contact first names and last names, and opportunity name and opportunity amount for every account if the first name of the contact is starting with A. (Relationship name is Contacts and Opportunities)

SELECT Account.Name,

(SELECT Contact.FirstName, Contact.LastName FROM Account.Contacts WHERE Contact.FirstName LIKE 'A%') ,

(SELECT Opportunity.name, Opportunity.amount FROM Account.Opportunities)

FROM Account

Note1: If you work with "**Custom Objects**" put "**_ _ c**" to the end of the Object Name and put "**_ _ r**" to the end of the custom object relationship name

Parent to Child Relationship Limitations

Note2: Subquery can be typed just in **1 level**; you cannot type subquery inside a subquery

Note3: In a single query, you can use up to **20 child** objects.

2. Child to Parent SOQL

The main object for the query is the child object and you fetch related parent details. Child records can only have one parent object. So you will be able to reach only one parent record.

Exercises Continue…

35) Find the contact name, contact phone, and account name, account website for every contact. (Field name is Account - Account is parent)

SELECT Name, Phone, Account.Name, Account.Website FROM Contact

36) Find the contact name, contact phone, and account name, account website, and account owner id for every contact. (Field name is Account - Account is parent) - (Field name is Owner - Owner is parent of Account, it means Owner is grand parent of Contact)

SELECT Name, Phone, Account.Name, Account.Website, Account.OwnerId FROM Contact

Child to Parent Relationship Limitations

Note: In each specified relationship, no more than 5 levels can be specified in a child-to-parentrelationship.

For example, Contact.Account.Owner.FirstName (3 levels)

Note: In a single query, you can use up to 55 different related objects

SOQL Relationship Challenge

➔ Account is a parent object of Contact; Contact is a parent object of Case.

Step1: Retrieve Name Department and Title of all Contacts.

SELECT Name, Department, Title FROM Contact.

Step2: Retrieve all Cases [CaseNumber, Subject] raised by the contact. (Relationship name is Cases)

SELECT Name, Department, Title, (SELECT CaseNumber, Subject FROM Cases) FROM Contact

Step3: Get parent Account's name, Rating, for each contact. (Relationship name is Cases) - (Field name is Account)

SELECT Name, Department, Title, (SELECT CaseNumber, Subject FROM Cases), Account.Name, Account.Rating FROM Contact

Step4: Make sure Account fields are the initial columns in result.

SELECT Account.Name, Account.Rating, Name, Department, Title, (SELECT CaseNumber, Subject FROM Cases) FROM Contact

Step5: Retrieve only those records where Account Rating is Hot.

SELECT Account.Name, Account.Rating, Name, Department, Title, (SELECT CaseNumber, Subject FROM Cases) FROM Contact WHERE Account.Rating = 'Hot'

Step6: Sort result by Contact Name.

SELECT Account.Name, Account.Rating, Name, Department, Title, (SELECT CaseNumber, Subject FROM Cases) FROM Contact WHERE Account.Rating = 'Hot' ORDER BY Name

Step7: Only retrieve open cases (use IsClosed checkbok field value)

SELECT Account.Name, Account.Rating, Name, Department, Title,
(SELECT CaseNumber, Subject FROM Cases WHERE IsClosed=false) FROM Contact
WHERE Account.Rating = 'Hot' ORDER BY Name

Step8: Add one more filter condition, Contact Department must be equals to 'Technology'

SELECT Account.Name, Account.Rating, Name, Department, Title,
(SELECT CaseNumber, Subject FROM Cases WHERE IsClosed=false)
FROM Contact WHERE Account.Rating = 'Hot' AND Department = 'Technology' ORDER BY Name

SOQL in Apex

Example 1 and 2:

```
public void f1(){
   //To store all accounts in a List
   List<Account> accounts = [ SELECT Name, Phone FROM Account ];
   //To see all accounts in a single line
   System.debug(accounts);
   //To see account details one by one in different lines by using for-each loop
   for(Account w : accounts){
      System.debug( 'Account name: ' + w.name );
   }
}

public void f1(){
   //To store all accounts in a Map
   Map<Id, Account> accountsMap = new Map<Id, Account>( [ SELECT Name, Phone FROM Account ] );
   //To see account details one by one in different lines by using for-each loop
   for( Account w : accountsMap.values( ) ){
      System.debug( 'Account name: ' + w.name );
   }
}
```

3) How to work with parent and child objects on Apex

```
public void f1(){
  List< Contact > contacts =[ SELECT Account.Name, Account.Rating, Name, Department, Title, (SELECT CaseNumber, Subject
FROM Cases ) FROM Contact ] ;
  //Take data from main object
   for( Contact w : contacts ){
      System.debug( 'Contact Name:' + w.name + ' Contact Department:' + w.Department + ' Contact Title:' + w.Title );
   }
   //Take data from parent object
   for( Contact w : contacts ){
      System.debug( 'Account Name:' + w.Account.Name + ' Account Rating:' + w.Account.Rating );
   }
   //Take data from child object
   for( Contact w : contacts ){
      for( Case t : w.Cases ){
         System.debug( 'Case Number:' + t.CaseNumber + ' Case Subject:' + t.Subject );
       }
   }
}
```

4) How to work with aggregate functions on Apex

1.Way:

```
public void f1(){
   AggregateResult[ ] groupedResults = [SELECT MIN( NumberOfEmployees ), MAX( NumberOfEmployees
   ), AVG( NumberOfEmployees ) FROM Account];
   System.debug( 'Min number of employees: ' + groupedResults[0].get('expr0') );
   System.debug( 'Max number of employees: ' + groupedResults[0].get('expr1') );
   System.debug( 'Average number of employees: ' + groupedResults[0].get('expr2') );
}
```

Then open Anonymous Window, create an object from Apex Class and by using object call the function f1

2.Way: By using Aliases:

```
public void f1(){
   AggregateResult[ ] groupedResults = [SELECT MIN( NumberOfEmployees ) min, MAX(
   NumberOfEmployees ) max, AVG( NumberOfEmployees ) avg FROM Account];
   System.debug( 'Min number of employees: ' + groupedResults[0].get('min') );
   System.debug( 'Max number of employees: ' + groupedResults[0].get('max') );
   System.debug( 'Average number of employees: ' + groupedResults[0].get('avg') );
}
```

3.Way: By using Aliases and multiple data

```
public void f1(){
   AggregateResult[ ] groupedResults = [SELECT Industry, AVG(NumberOfEmployees) avg FROM Account
   GROUP BY Industry];
   for (AggregateResult w : groupedResults) {
   System.debug('Industry: ' + w.get('Industry') + ' - Average number of employees: ' + w.get('avg'));
   }
}
```

5) Binding Variables in SOQL in Apex

```
List<String> accountNames = new List<String> {'GenePoint', 'Burlington Textiles Corp of America', 'sForce',
'Dickenson plc'};
//“ :accountNames ” is called Binding Variables
//Binding Variables can be used after WHERE with IN, NOT IN, =, !=
List<Account> accounts = [SELECT Id, Name, Rating FROM Account WHERE Name IN :accountNames];
List<Account> accounts = [SELECT Id, Name, Rating FROM Account WHERE Name NOT IN
:accountNames];
List<Account> accounts = [SELECT Id, Name, Rating FROM Account WHERE Name = :accountNames];
List<Account> accounts = [SELECT Id, Name, Rating FROM Account WHERE Name != :accountNames];
for(Account w : accounts){
System.debug('Account name: ' + w.Name + ' Account rating: ' + w.Rating);
}
//Binding Variables can be used with LIMIT, OFFSET
Integer i = 2;
List<Account> accounts = [SELECT Id, Name, Rating FROM Account LIMIT :i ];
List<Account> accounts = [SELECT Id, Name, Rating FROM Account OFFSET :i ];
List<Account> accounts = [SELECT Id, Name, Rating FROM Account LIMIT :i OFFSET :i];
for(Account w : accounts){
System.debug('Account name: ' + w.Name + ' Account rating: ' + w.Rating);
}
```

SOSL (Salesforce Object Search Language)

➔Performs text searches in multiple records.

➔Use SOSL to search fields across multiple standard and custom object records in Salesforce.

➔It cannot search for numbers. It cannot search for records that meet a certain condition.

Whether you use SOQL or SOSL depends on whether you know which objects or fields you want to search, plus other considerations.

Use SOQL when you know which objects the data resides in, and you want to:

- Retrieve data from a single object or from multiple objects that are related to one another.
- Count the number of records that meet specified criteria.
- Sort results as part of the query.
- Retrieve data from number, date, or checkbox fields.

Use SOSL when you don’t know which object or field the data resides in, and you want to:

- Retrieve data for a specific term that you know exists within a field. Because SOSL can tokenize multiple terms within a field and build a search index from this, SOSL searches are faster and can return more relevant results.
- Retrieve multiple objects and fields efficiently where the objects might or might not be related to one another.
- Retrieve data for a particular division in an organization using the divisions feature.

- Retrieve data that's in Chinese, Japanese, Korean, or Thai. Morphological tokenization for CJKT terms helps ensure accurate results.

➔A SOSL query begins with the required FIND clause. After the required FIND clause, you can add one or more optional clauses in the following or

```
FIND {SearchQuery}
[ IN SearchGroup ]
[ RETURNING FieldSpec [[ toLabel(fields)] [convertCurrency(Amount)] [FORMAT()]] ]
[ WITH DivisionFilter ]
[ WITH DATA CATEGORY DataCategorySpec ]
[ WITH SNIPPET[(target_length=n)] ]
[ WITH NETWORK NetworkIdSpec ]
[ WITH PricebookId ]
[ WITH METADATA ]
[ LIMIT n ]
[ UPDATE [TRACKING], [VIEWSTAT] ]
```

In SearchGroups: If unspecified, the default is ALL FIELDS. You can specify the list of objects to search in the RETURNING FieldSpec clause. Optional. Scope of fields to search. One of the following values:

- ALL FIELDS
- NAME FIELDS
- EMAIL FIELDS
- PHONE FIELDS
- SIDEBAR FIELDS

Exercises:

Step1: Look for the name Joe Smith anywhere in the system, in a case-insensitive search. Return the IDs of the records where Joe Smith is found.

FIND {Joe Smith}

Step2: Look for the name Joe Smith in the name field of a lead, return the ID field of the records.

FIND {Joe Smith} IN Name Fields RETURNING lead

Step3: Look for the name Joe Smith in the name field of a lead and return the name and phone number.

FIND {Joe Smith} IN Name Fields RETURNING lead(name, phone)

Step4: Look for the name Joe Smith in the name field of a lead. Return the name and phone number of any matching record that was also created in the current fiscal quarter.

FIND {Joe Smith} IN Name Fields RETURNING lead (name, phone Where createddate = THIS_FISCAL_QUARTER)

Step5: Look for the name Joe Smith or Joe Smythe in the name field of a lead or contact and return the name and phone number. If a people record is called Joe Smith or Joe Smythe, that record isn't returned.

```
FIND {"Joe Smith" OR "Joe Smythe"}
IN Name Fields
RETURNING lead(name, phone), contact(name, phone)
```

Note: We can also use wildcard in searchquery. Asterisks (*) match zero or more characters at the middle or end of your search term. Question marks (?) match only one character in the middle or end of your search term

DML (Data Manipulation Language) Statements

DML are the actions which are performed in order to perform **insert, update, delete, upsert (insert/update)**, (undelete) restoring records, **merging** records, or converting leads operation.
DML is one of the most important part in Apex as almost every business case involves the changes and modifications to database.

DML Statements:

Insert Operation: Insert operation is used to create new records in Database. You can create records of any Standard or Custom object using the Insert DML statement.

Update Operation: Update operation is to perform updates on existing records.

Upsert Operation: Upsert Operation is used to perform an update operation and if the records to be updated are not present in database, then create new records as well.

Delete Operation: You can perform the delete operation using the Delete DML.

Undelete Operation: You can undelete the record which has been deleted and is present in Recycle bin. All the relationships which the deleted record has, will also be restored.

Database Methods

All operations which you can perform using DML statements can be performed using Database methods as well. Database methods are the system methods which you can use to perform DML operations. Database methods provide more flexibility as compared to DML Statements.

DML Statements	Database Methods
Partial Update is not allowed. For example, if you have 20 records in list, then either all the records will be updated or none.	Partial update is allowed. You can specify the Parameter in Database method as true or false, true to allow the partial update and false for not allowing the same.
You cannot get the list of success and failed records.	You can get the list of success and failed records as we have seen in the example.
Example – insert listName	**Example** – Database.insert(listName, False), where false indicate that partial update is not allowed.

Insert Example:

1) Let's insert data one-by-one:

```
//Create a new Account Object, assign account name which is mandatory, Phone is optional
Account acc = new Account( Name='Account to be Inserted', Phone='1234567890');
//Adding more fields can be done like the following as well
acc.Rating='Hot';
//To insert the "acc" record into Account Object;
Insert acc; // ==> Database.insert(acc); also works
```

Note 1: To understand if it is created type DML into text box on Anonymous Window next to the Filter Checkbox

Note 2: To see newly created record on the console type **"SELECT Id, Name FROM Account"**

2) Let's insert multiple data into org;

```
//Create a new Account Object, assign account name which is mandatory, Phone is optional
Account acc = new Account( Name='Account to be Inserted', Phone='1234567890');
//Adding more fields can be done like the following as well
acc.Rating='Hot';
//To insert the "acc" record into Account Object;
Insert acc; // ==> Database.insert(acc); also works
//Create a list to store multiple account
List<Account> accList = new List<Account>( );
//Create multiple objects (account)
Account acc1 = new Account( Name='myFisrtAccount', Phone='1234567890');
Account acc2 = new Account( Name='mySecondAccount', Phone='2345678901');
Account acc3 = new Account( Name='myThirdAccount', Phone='3456789012');
//Add objects into accList
accList.add(acc1);
accList.add(acc2);
accList.add(acc3);
//To insert the "accList" records into Account Object;
insert accList; // ==> Database.insert(accList); also works
```

Note 1: The difference between **insert accList**; and **Database.insert(accList);**

i**) insert accList**; Partial success is not allowed. It works in "All or None"

ii**) Database.insert(accList, true);** and Database.insert(accList); Partial success is not allowed.

iii) **Database.insert(accList, false)**; Where false indicate that Partial success is allowed.

Update Example: Please read following notes it's very important for update, delete, undelete DML operation.

Note 1: Without having Id you cannot do "update", "delete", and "undelete". Because of that, you have to find the Id.

Note 2: To find the Id type the following Query and assign it to a List because, maybe you will have multiple results from the query. If you use just object to assign, when the query returned multiple records, it fails.

```
//Step1:Creation of data to update;
Account acc = new Account( Name='myAccount', Phone='123456789');
//Step2:Insert acc into database
Database.insert(acc);
//Step3: Important: Fetch the acc together with Id and put data into list
List<Account> accounts = [SELECT Id FROM Account WHERE Name='myAccount' AND
Phone='123456789'];
//Use for each loop to update all data
for(Account w : accounts){
w.Name = 'myUpdated Account';
}
update accounts; // ==> Database.update(accounts); also works
```

Delete Example:

```
//Insert data to delete
Account acc = new Account( Name='myAccountToDelete', Phone='1234');
Database.insert(acc);
//Note 3:  Fetch the acc together with Id. if you dont fetch with Id it does not delete
List<Account> accounts = [SELECT Id FROM Account WHERE Name LIKE 'myAccountTo%'];
delete accounts; // ==> Database.delete(accounts); also works
}
```

Undelete Example:

Note 1: To undelete, you should find the deleted records first from Recycle Bin

Note 2: Queries realted with the RecycleBin cannot be executed in "Query Editor" if you write there it will give error.

Note 3: Queries realted with the RecycleBin can be executed just in "Anonymus Window" or in "Apex Class"

```
//fetch deleted data by following querry from recycle bin
List<Account> accounts = [SELECT Id FROM Account WHERE isDeleted = true ALL ROWS];
undelete accounts; // ==> Database.undelete(accounts); also works
```

Example: This example is very important to understand DML operations.

```
//Step1: Insert==> Lets create account records and Insert them to the Database
List <Account> accToInsert = new List <Account> ();
for (Integer i=0; i<10; i++){
   //Create account records for each iteration
   Account obj = new Account( Name = 'Test'+i, Phone='999'+i);
   //Add records to the list
   accToInsert.add(obj);
}
//If you want to see these accounts created or not on the Console  use following code
for(Account w:accToInsert){
   System.debug(w);
}
//Insert accToInsert into database;
Insert accToInsert;
//Step2: Lets update some of record that we created above
//Don't forget to fetch records with ID
List <Account> accToUpdate = [SELECT Id FROM Account WHERE Name LIKE 'Test%'];
//Update the Name and Type  of Fisrt 5 Account
For (Integer i = 0; i<5; i++){
   accToUpdate[i].Name='Updated Account'+i;
   accToUpdate[i].Type= 'Education';
}
//Update the accToUpdate List to the Database
Database.Update(accToUpdate);

//Step3: Delete ==> Lets delete Account satarting with 'Test'
//If you dont fetch with id it does not delete
List <Account> accToDelete = [SELECT id FROM Account WHERE Name LIKE 'Test%'];
//Delete Records in accToDelete list
Delete accToDelete; //or Database.delete(accToDelete);

//Step4: Let's restore Accounts from recycle bin that we deleted in previous Step;
//fetch deleted Account by following Querry from recyle-bin
List<Account> accToResotere = [SELECT id FROM Account WHERE isDeleted=true and Name LIKE 'Test%' ALL Rows];
//Restore accToResotere list into Database
undelete accToResotere; //or database.undelete(accToResotere);
//Step4: Upsert ==>
List<Account> AccToUpsert = [SELECT id FROM Account WHERE Name LIKE 'Updated%' Limit 3];
for (Account w: AccToUpsert){
   w.Industry = 'Banking';
   w.Name= 'Upsert Account';
}
Account obj01 = new Account (Name='New Account', Phone='111', Industry='Shipping');
AccToUpsert.add(obj01);
Account obj02 = new Account (Name='New Account2', Phone='222', Industry='Shipping');
AccToUpsert.add(obj02);
//To see final list
for (Account w: AccToUpsert){
   System.debug(w);
}//Upsert List; We have account that already in database also we add new accounts the list
Upsert AccToUpsert;
```

Exceptions Statements in Apex:

Apex uses exceptions to note errors and other events that disrupt the normal flow of code execution. **throw** statements can be used to generate exceptions, while **try**, **catch**, and **finally** can be used to gracefully recover from an exception.

Throw Statements: A throw statement allows you to signal that an error has occurred.

Example:

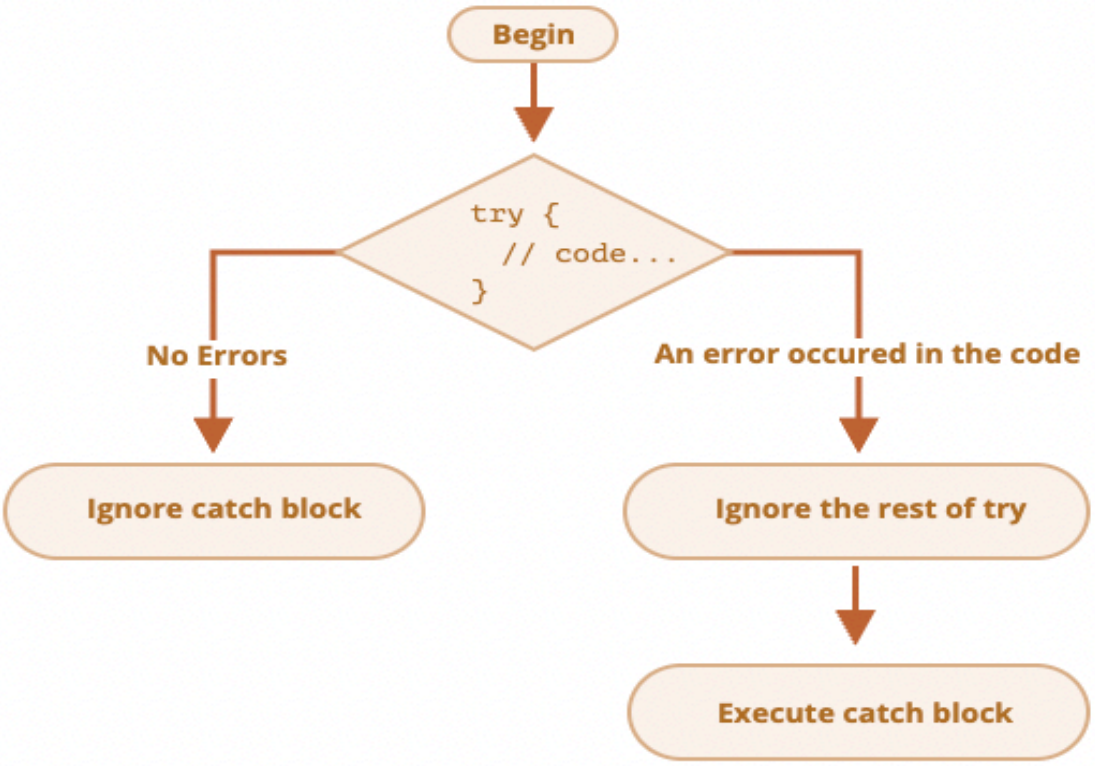
```
throw exceptionObject;
```

Try-Catch-Finally Statements:

The try, catch, and finally statements can be used to gracefully recover from a thrown exception.

Syntax:

```
try {
    // The try statement identifies a block of code in which an exception
can occur
   code_block
  } catch (exceptionType variableName) {
    // The catch statement identifies a block of code that can handle a
particular type of exception.
    // Initial catch block.
    // At least the catch block or the finally block must be present.
   code_block
  } catch (Exception e) {
    // Optional additional catch statement for other exception types.
    // Note that the general exception type, 'Exception',
    // must be the last catch block when it is used.
   code_block
  } finally {
    // The finally statement identifies a block of code that is guaranteed
to execute and allows you to clean up your code.
    // At least the catch block or the finally block must be present.
   code_block
}
```



Built-In

Exceptions and Common Methods:

1) **MatException:** We get MatException if we do any mathematical mistake in our code.

Example:

➔Here since we don't have try-catch code bock our code give exception and stopped working.

```
Integer num1 = 12;
Integer num2 = 0;
Integer result = num1 / num2;
System.debug('The result is ' + result);//System.MathException: Divide by 0
```

How to Handle Exception:

➔Positive scenario:Here no Exceptioan occoured and my code worked.

```
try{
    Integer num1 = 12;
    Integer num2 = 3;
    Integer result = num1 / num2;
```

```
    System.debug('The result is ' + result); //The result is 4

}catch(MathException me){
    System.debug('A mathematical issue occured ' + me.getMessage());
}
System.debug('As you see my app was not blocked'); //As you see my app was
not blocked
```

➔Negative scenario: **Here even Exception occured the code worked since we used try, catch**

```
try{
    Integer num1 = 12;
    Integer num2 = 0;
    Integer result = num1 / num2;
    System.debug('The result is ' + result);
}catch(MathException me){
    System.debug('A mathematical issue occured ' + me.getMessage()); //A
mathematical issue occured Divide by 0is 4
}
System.debug('As you see my app was not blocked'); //As you see my app was
not blocked
```

2) ListException: If you do any mistake in list rules Apex will throw ListException

Example:

```
List<Integer> list1 = new List<Integer>();
list1.add(11);
list1.add(12);
list1.add(13);
System.debug(list1);// (11, 12, 13)
System.debug(list1[0]);//11
System.debug(list1[1]);//12
System.debug(list1[2]);//13
```

➔**Here we will get listException error and our application will stop working**

```
System.debug(list1[5]);//System.ListException: List index out of bounds: 5
```

➔Negative scenario: **Even when listException occurred my whole code run till end.**

```
Integer idx1 = 0;
```

```
Integer idx2 = 1;
Integer idx3 = 5;
try{
    List<Integer> list1 = new List<Integer>();
    list1.add(11);
    list1.add(12);
    list1.add(13);
    System.debug(list1);// (11, 12, 13)
    System.debug(list1[idx1]);//11
    System.debug(list1[idx2]);//12
    System.debug(list1[idx3]); //Throws ListException because I used non-
existing index
}catch(ListException le){//If you do not want specify the exception name you
can use "Exception" instead of "ListException"
    System.debug('You break a list rule ' + le.getMessage());  //You break a
list rule List index out of bounds: 5
}
System.debug('As you see my app was not blocked'); //As you see my app was
not blocked
```

➔**Positive scenario:** **No listException catched so catch-blocked do not run.**

```
Integer idx1 = 0;
Integer idx2 = 1;
Integer idx3 = 2;
try{
    List<Integer> list1 = new List<Integer>();
    list1.add(11);
    list1.add(12);
    list1.add(13);
    System.debug(list1);// (11, 12, 13)
    System.debug(list1[idx1]);//11
    System.debug(list1[idx2]);//12
    System.debug(list1[idx3]);//13
}catch(ListException le){
    System.debug('You break a list rule ' + le.getMessage());
}
System.debug('As you see my app was not blocked'); //As you see my app was
not blocked
```

Multiple Exception in try-block:

Way1: We can use general **Exception** in catch-block to catch any type of exception.

```
Integer idx1 = 0;
Integer idx2 = 1;
Integer idx3 = 5;
try{
//In the following code as you can see we multiple exceptions in try-block
    Integer num1 = 12;
    Integer num2 = 3;
    Integer result = num1 / num2;
    System.debug('The result is ' + result);
    List<Integer> list1 = new List<Integer>();
    list1.add(11);
    list1.add(12);
    list1.add(13);
    System.debug(list1);// (11, 12, 13)
    System.debug(list1[idx1]);//11
    System.debug(list1[idx2]);//12
    System.debug(list1[idx3]);// here we have listException
}catch(Exception me){ //Here we handle alltype of exception by using general
Exception method.
    System.debug('An issue occured ' + me.getMessage());//that will give
message of first exception.
}
System.debug('As you see my app was not blocked');
```

Way2: We can as many as catch-block for each type of exceptions to catch them.

```
Integer idx1 = 0;
Integer idx2 = 1;
Integer idx3 = 5;
try{
    Integer num1 = 12;
    Integer num2 = 3;
    Integer result = num1 / num2;
    System.debug('The result is ' + result);
    List<Integer> list1 = new List<Integer>();
    list1.add(11);
    list1.add(12);
    list1.add(13);
```

```
    System.debug(list1);// (11, 12, 13)
    System.debug(list1[idx1]);//11
    System.debug(list1[idx2]);//12
    System.debug(list1[idx3]);//here we have listException.
}catch(MathException me){//first block of catch will get mathException
    System.debug('A mathemtical issue occured ' + me.getMessage());
}catch(ListException le){//second block of catch will get any listException
    System.debug('A list issue occured ' + le.getMessage());
}finally{//If you need any code to execute under every condition put it into
the finally block
    System.debug('End the connection with DB'); //End the connection with DB
}
System.debug('As you see my app was not blocked');//As you see my app was
not blocked
```

Important Notes:

➔Try-block must have at least one catch-block or a finally block.

➔try-block can be used with " ***only 1 catch" or "only 1 finally" or " 1 catch + 1 finally" or "multiply catch " or "Multiply catch + 1 finally"***

➔Don't forget try-block cannot be used alone.

3) NullPointerException: If you use null String with some String Class functions you will get NullPointerException

Example: In the following code we have NullPointerException

```
String s;
Integer numOfChars = s.length();
system.debug('The number of characters in the String is ' +
numOfChars);//System.NullPointerException: Attempt to de-reference a null
object
```

➔How to handle exception:

```
try{
    String s;
    Integer numOfChars = s.length();
    system.debug('The number of characters in the String is ' + numOfChars);
}catch(NullPointerException npe){
```

```
    System.debug('Do not use null String with String functions ' +
npe.getMessage());
    //Do not use null String with String functions Attempt to de-reference a
null object
}
```

4) QueryException: If you assign a query which returns records different from 1 it throws QueryException
Example:

```
List<Account> list1 = [SELECT name FROM Account WHERE name LIKE 'xyz%'];
System.debug(list1.size());//0==> since there is no any account with name
start with xyz list is empty.

Account acc1 = [SELECT name FROM Account WHERE name LIKE 'xyz%']; //Fetching
exactly one record
System.debug(acc1); //System.QueryException: List has no rows for assignment
to SObject
```

➔How to handle the exception:

```
try{
  Account acc1 = [SELECT name FROM Account WHERE name LIKE 'xyz%'];
  System.debug(acc1);
}catch(QueryException qe){
  System.debug('I was expecting exactly 1 record from the query but it
returned different ' + qe.getMessage());
  //I was expecting exactly 1 record from the query but it returned
different List has no rows for assignment to SObject
}
```

5) SObjectException: If you try to access un-fetched data from an SObject it will throw SObjectException
Example: Lets fetch an accounts with only their names from data base. Then when will try to call its phone it will thrown a sObjectException.

```
List<Account> list2 = [SELECT name FROM Account];
System.debug(list2[0].phone);//System.SObjectException: SObject row was
retrieved via SOQL without querying the requested field: Account.Phone
```

➔how to handle exception:

```
try{
  List<Account> list2 = [SELECT name FROM Account];
  System.debug(list2[0].phone);
```

```
}catch(SObjectException soe){
  System.debug('Do not try to get un-fetched data from an SObject ' +
soe.getMessage());
  //Do not try to get un-fetched data from an SObject SObject row was
retrieved via SOQL without querying the requested field: Account.Phone
}
```

5. Create Custom Exceptions

You can't throw built-in Apex exceptions. You can only catch them. But with custom exceptions, you can throw and catch them in your methods. Custom exceptions enable you to specify detailed error messages and have more custom error handling in your catch blocks.

Example:

Step1: Creation of Exception Class:

```
public class myException extends Exception{//To create your custom exception
class, extend the built-in Exception class and make sure your class name
ends with the word Exception,
    //1) Exception Classes can have member variable
    //2) can have methods
    //3) can have constructor.
    public static void checkUserAge(Integer age){//creation of method.
        if(age<0){
            throw new myException('Age cannot be negative');
        }
        System.debug('The age is valid and the age is ' + age);
    }
}
```

Step2: Call it from anonymous window:

```
try {
    myException.checkUserAge(-5);
}catch(myException ane){
    System.debug('Do not use negative values for ages ' + ane.getMessage());
    //Do not use negative values for ages Age cannot be negative
}
```

Common Exception Methods

You can use common exception methods to get more information about an exception, such as the exception error message or the stack trace. The previous example calls the **getMessage** method, which

returns the error message associated with the exception. There are other exception methods that are also available. Here are descriptions of some useful methods:

- **getCause:** Returns the cause of the exception as an exception object.
- **getLineNumber:** Returns the line number from where the exception was thrown.
- **getMessage:** Returns the error message that displays for the user.
- **getStackTraceString:** Returns the stack trace of a thrown exception as a string.
- **getTypeName:** Returns the type of exception, such as DmlException, ListException, MathException, and so on.

```
try {
    Account acc1 = [SELECT Name FROM Account LIMIT 1];
    // Causes an SObjectException because we didn't retrieve the phone field.
    String accPhone = acc1.phone;
} catch(Exception e) {
    System.debug('Exception type caught: ' + e.getTypeName());//Exception type
caught: System.SObjectException
    System.debug('Message: ' + e.getMessage());//SObject row was retrieved via SOQL without querying
the requested field: Account.Phone
    System.debug('Cause: ' + e.getCause());    // Cause: null
    System.debug('Line number: ' + e.getLineNumber()); //Line number: 5
    System.debug('Stack trace: ' + e.getStackTraceString());//AnonymousBlock: line 5, column 1  }
```

Attention Trailblazer!

To help you practice, we have created trailmixes which prepared by our expert instructors, on trailhead platform which is the best place to do practice for each topic. The context of each trailmix is usually same as the subject matter. You can reach the trailmix by following link: https://trailhead.salesforce.com/users/wisequarter/trailmixes/apex-fundamentals

Interview Questions:

In that section, you will find common interview questions related to the topic. The above text provides answers to those questions. Our expert instructors will also share and discuss the answers they prepared during the Interview Preparation class.

Please give extra attention to questions with a star beside them. They are always asked in every interviews.

1. **Who developed Apex programing Language?**
2. **What is APEX, and what is it used for?**
3. **Is Apex strongly typed language?**
4. **What are the different data types in APEX?**
5. **What is sObject in Apex?**
6. **What is Force.com Developer Console?**
7. **What is Enum in Apex?**
8. **What is ID type in Salesforce Apex?**
9. **How do you convert a string to an Integer type?** ★
10. **Can you explain some string methods which were used in your projects?** ★
11. **What does contains method do in Apex?**
12. **What are the different Collections in APEX?** ★★
13. **What is difference between set and list?**
14. **What is difference between static methods and non static methods?** ★★
15. **What are Access Modifiers?**
16. **What is the difference between APEX Class and Object?** ★★

17. What is Constructor in Apex Programming?
18. What are the best practices of Apex?
19. What does Object-oriented programming mean?
20. Explain Encapsulation?
21. Explain Inheritance?
22. Explain Polymorphism?
23. Explain Method Overloading in Apex?
24. Explain Method Overriding In Apex?
25. Explain Abstraction in Apex
26. What is different between interface and class?
27. Can we use Private access modifiers for the outer (main) class?
28. Why we need Setter and Getter method?
29. What is with sharing / without sharing class in APEX?
30. What is the mixed DML operation error?
31. How can you fix DML operation?
32. What is the difference between database.insert and insert?
33. What is the dynamic Apex?
34. Can you customize an Apex code in a production org?
35. What is SOQL?
36. What is the governor limit?
37. What is SOQL governor limits and maximum number of records that can be returned by a single query?
38. What is SOSL? When should it be preferred over SOQL?
39. What are some best practices to follow when using SOQL?
40. What are the two words that appear in every SOQL statement?
41. What is Exception handling in Apex?
42. How to handle exceptions in Apex?
43. What is difference between '==' and '===' ?
44. In Apex, when should you use == as a comparison operator and when should you use .equals()?
45. Type a code to calculate the factorial of a number.
46. Type a code to execute the multiplication table.

Reference:

- www.wisequarter.com
- https://keap.com/product/what-is-crm
- https://www.salesforce.com/crm/what-is-crm/
- https://www.zdnet.com/article/what-is-cloud-computing-everything-you-need-to-know-about-the-cloud/
  https://www.geeksforgeeks.org/cloud-deployment-models/
- https://www.bigcommerce.com/articles/ecommerce/saas-vs-paas-vs-iaas/#the-3-types-of-cloud-computing-service-models-explained
- https://www.forcetalks.com/salesforce-topic/how-many-types-of-portals-are-available-in-salesforce/
- https://shreysharma.com/portals-and-communities/