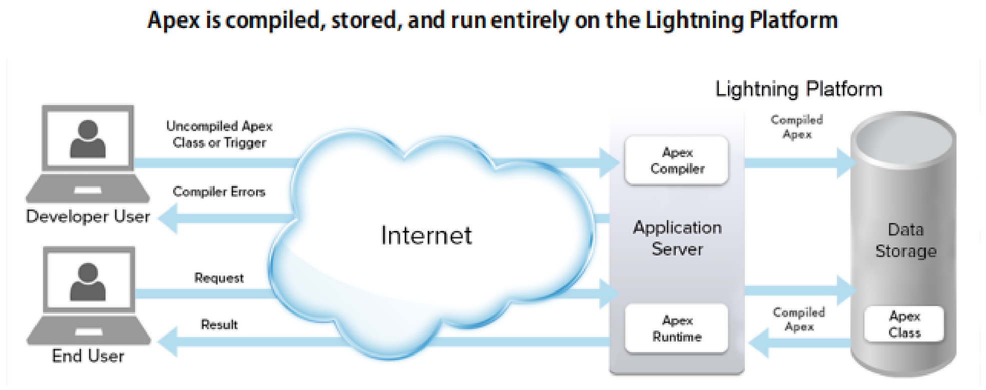
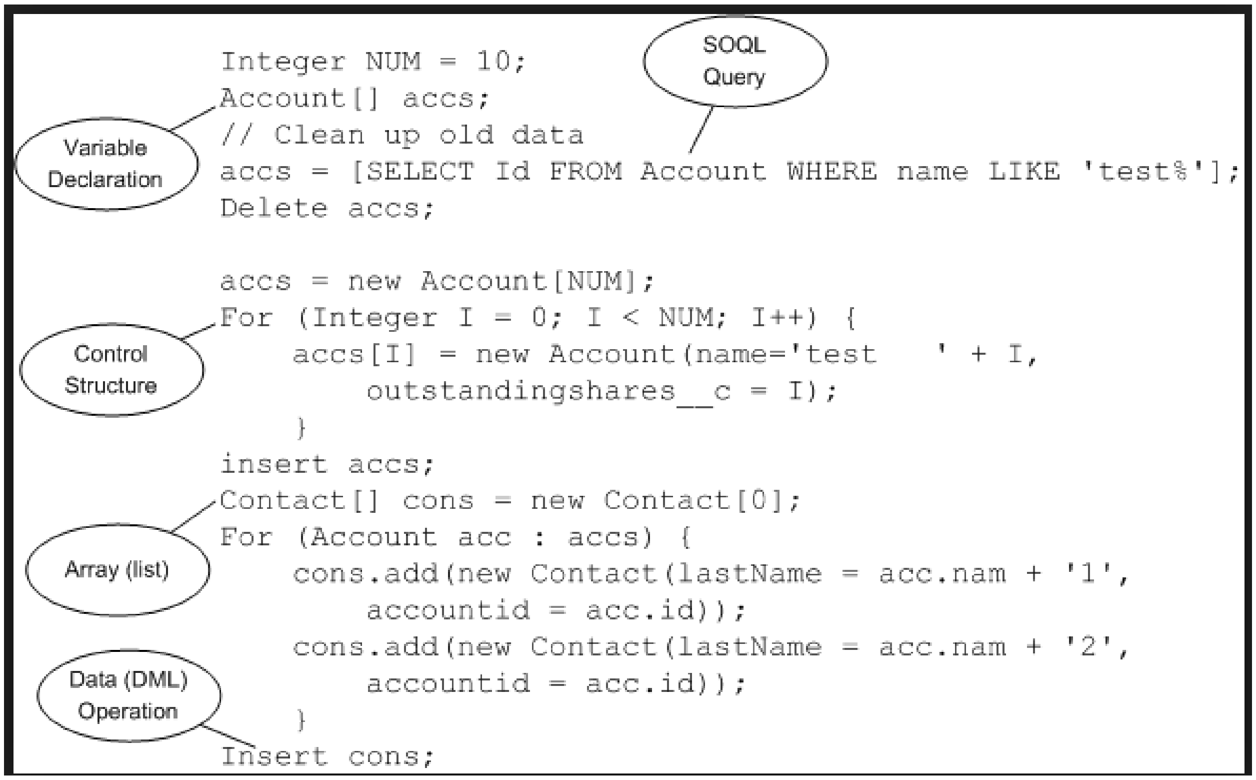
APEX - Understanding document

1.Basics of APEX



Apex is a programming language that uses Java-like syntax and acts like database stored procedures. Apex enables developers to add business logic to system events, such as button clicks, updates of related records, and Visualforce page



```
Integer NUM = 10;
Account[] accs;
// Clean up old data
accs = [SELECT Id FROM Account WHERE name LIKE 'test%'];
Delete accs;

accs = new Account[NUM];
For (Integer I = 0; I < NUM; I++) {
    accs[I] = new Account(name='test    ' + I,
        outstandingshares__c = I);
    }
insert accs;
Contact[] cons = new Contact[0];
For (Account acc : accs) {
    cons.add(new Contact(lastName = acc.nam + '1',
        accountid = acc.id));
    cons.add(new Contact(lastName = acc.nam + '2',
        accountid = acc.id));
    }
Insert cons;
```

Exception handling ,Test, Batch,Integration

1.1 Best Practices

- Variable names should not have Whitespace, Special characters (+,-,&..)
- Donot use reserved keywords as a part of name (String, Integer..)
- Always give meaningful names to variables and use CamelCase notation
- Camelcase notation - Starts with lowercase and capitalise each consequent word)
- String myString, mySecondInteger;
- String MyString, mysecondinteger;

1.2 APEX Variable declaration

String myName='Rahul';
Integer myInteger=5;
Decimal myDecimal=5.62;
Boolean myBoolean=True;

If we declare a variable and don't initialise it with a value , the value will be null.

1.2.1 APEX Constant

- A constant is a variable whose value doesnot change after initialisation is done
- Constants can be defined using **Final** Keyword

```
//Example
Final Decimal pi=3.14159265;
```

1.2.2 Strings

- Used to hold any character: letters, numbers..etc. • Declared with a single quote, not double quote!
- Use escape sequences to denote special characters, like \' for a single quote, \r for carriage return, \n for line feed…etc.

```
//Example
String myString = 'This is a String variable';
```

- Salesforce Fields Types:
  - All text and picklists fields
  - Auto-number, email, phone fields

- String Methods
    - Methods, like: .length(), .remove(substring), .isAlpha(), .reverse(), .toUpperCase()…etc.

```
//Example
String myStr = 'abcdef';
Integer result = myStr.length();
```

1.2.3 Integer

- Used to hold a number without decimal place (32-bit)
- Minimum value of -2,147,483,648
- Maximum value of 2,147,483,647

```
//Example
Integer myInteger = 365;
```

- Salesforce Fields Types:
    - Any Number field, but with 0 decimal place
- Integer Methods:
    - 3 methods: .format(), .valueOf(stringToInteger), .valueOf(fieldValue)

1.2.4 Decimal

- Used to hold a number with 1 or more decimal places (32-bit)

```
//Example
Decimal myDecimal = 2.75;
```

- Salesforce Fields Types:
    - Any Number field, but with 1 or more decimal places
    - Currency fields are automatically assigned the type Decimal
    -
- Decimal Methods:
    - Many methods, like: .setScale(scale), .toPlainString(), .precision()

1.2.5 Double

- Used to hold a number with 1 or more decimal places.(64-bit)

- Used when you need a wider range than Decimal

```
//Example
Double myDouble = 21474836470099.5;
```

- Salesforce Fields Types:
  - Any Number field, but with 1 or more decimal places, but with wider range than Decimal

- Double Methods:
  - Some methods: .format(), .intValue(), .valueOf(stringToInteger)

1.2.6 Date

- A value that indicates a particular day.
- Date values contain no information about time.
- You can add or subtract an Integer value from a Date value, returning a Date value

```
//Example
Date myDate = Date.newInstance(1986, 2, 17);
Date myDate = Date.today();
```

- Salesforce Fields Types:
  - Any Date field without time, like Close Date
- Date Methods:
  - Some methods, like: .month(), .addDays(additionalDays), dayOfYear()

1.2.7 DateTime

- A value that indicates a particular day + time, like a timestamp.
- Datetime values must always be created with a system static method.
- You can add or subtract an Integer or Double value from a Datetime value, returning a Datetime value.

```
//Example
Datetime myDT1 = Datetime.newInstance(2020, 1, 1, 12, 30, 2);
Datetime myDT2 = Datetime.now();
```

- Salesforce Fields Types:

    - Any Datetime field, like Created Date

- Datetime Methods:
    - Some methods: .format(), .hour(), .minute(), etc…

1.2.7 Time

- A value that indicates a particular time
- Time values must always be created with a system static method

```
//Example
Time myTime = Time.newInstance(18, 30, 2, 20);
```

- Salesforce Fields Types:
    - No field type related to time, only used in coding

- Time Methods:
    - Some methods: .addHours(additionalHours), .second(), .addMinutes(additionalMinutes)

1.2.8 Boolean

- Can only hold True or False (or null)

```
//Example
Boolean myBool1 = true;
Boolean myBool2 = false;
```

- Salesforce Fields Types:
    - Checkbox field

- Boolean Methods:
    - 2 methods: . valueOf(stringToBoolean), .valueOf(fieldValue)

1.2.9 ID

- 18-character Salesforce.com record identifier
- The Id Field Type is a base-62 encoded string

- Each character can be one of 62 possible values: 26 lower case letters, 26 UPPER case letters, 10 digits
- Declared with a single quote like String
- If you set ID to a 15-character value, Apex converts the value to its 18-character

```
//Example
Id myId='00300000003T2PGAA0';
Id myId='00300000003T2PG';
```

- Salesforce Fields Types:
  - ID field
  - Lookup and Master-Detail reference fields
- Id Methods:
  - Few methods: .getSObjectType(), .valueOf(toID)

1.2.10 Sobjects

- A Specific Salesforce Object
- Refers to any object that can be stored in the Salesforce.com platform database
- Represents a row of data and can only be declared in Apex using the SOAP API name of the object.
- Can be a Standard Object, or a Custom Object Record

```
//Example
Account a = new Account();
a=[SELECT ID FROM account limit 1];
Book__c book1 = new Book__c();
```

1.2.11 Objects

- Any data type that is supported in Apex:
  - Primitive data types (such as Integer),
  - sObject specific type (such as Account, Book__c…etc)
  - User-defined custom classes
  - The sObject generic type
- All Apex data types inherit from Object
- You can cast an object that represents a more specific data type to its underlying data type

```
//Example
Object obj = 10;
//Cast the object to an integer.
Integer i = (Integer)obj;
System.assertEquals(10, i);
```

2. Collections in APEX

Collection is a data structure which contains and processes a set of data. The data stored in the collection is encapsulated and the access to the data is only possible via predefined methods.

For example if your application saves data in an object of type People, you can store several People objects in a collection.

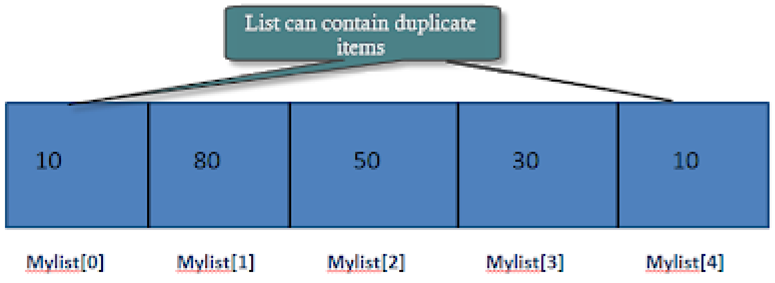
Each collection type is different, but there are four methods you can invoke on all of them:

1. clear: Removes all elements from the collection.

2. clone: Returns a copy of the collection.
3. isEmpty: Returns false if the collection has elements, true if empty.
4. size: Returns the number of elements in the collection as an Integer.

2.1 List

- Ordered/Indexed collection of non-Unique elements (ON)
- The index is an integer and it starts with 0
- List are equivalent to Arrays
- Can be of any data type:
    - Primitive types
    - sObjects
    - User-defined types
    - Built-in Apex types
- Can also contain any collection and can be nested within one another and become multidimensional.



Create a List:

```
List<Account> ls1 = new List<Account>(); //Empty
List<Integer> ls2 = new List<Integer>{1, 2, 3};
List<Integer> ls3 = new List<Integer>(5);

List<Contact> conList = new List<Contact> {
new Contact (FirstName='Joe', LastName='Doe'),
new Contact (FirstName='Vic', LastName='Doe') };


Contact josh = new Contact (FirstName='Joe', LastName='Doe');
Contact vic = new Contact (FirstName='Vic', LastName= 'Doe');
List<Contact> contacts = new List<Contact> { josh, vic};
```

2.1.1 List Methods

List<Integer> myList = new List<Integer>();

Method	Example

Add (element)
Add (index , element)
	myList.add(25);
	myList.add(1,50);

Get (index)	myList.get(2);
Set (index, listElement)	myList.set(3,60);
Clear()	myList.clear();
isEmpty ()	Boolean isEmpty = myList.isEmpty();

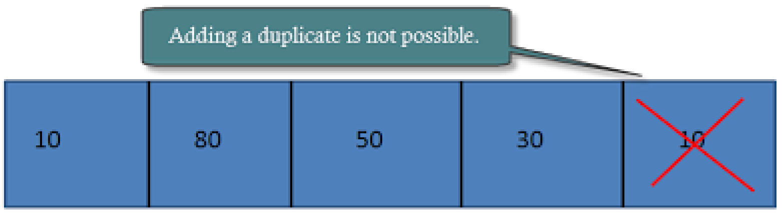
Size ()	Integer listSize = myList.size();
Remove(index)	myList.remove(3);

2.1.2 List Examples

```
//empty List initialized
List<String> strList = new List<String>();

//elements added to the set
strList.add ('element 0');
strList.add ('element 1');
strList.add ('element 2');

//duplicate element added to the list
strList.add (‘element 1');

//this will output size of list
Integer listSize = strList.size();
//the result will be 4

system.debug('@@size : ' + listSize);

//List with SOQL
List<Account> accs = [SELECT Id, Name FROM Account WHERE Name =’ACC’s];
```

2.2 Set

- Unordered Collection of Unique elements (UU)
- There is no index
- Sets are equivalent to Bags
- Can be of any data type:
  - Primitive types
  - sObjects
  - User-defined types
  - Built-in Apex types
- Can also contain any collection and can be nested within one another and become multidimensional.



Create a Set:

```
Set<Integer> s1 = new Set<Integer>();
Set<Integer> s2 = new Set<Integer>{1, 2, 3};

Set<Contact> conSet = new Set<Contact> {
new Contact (FirstName='Joe', LastName='Doe'),
new Contact (FirstName='Vic', LastName='Doe') };

Contact josh = new Contact (FirstName='Joe', LastName='Doe');
Contact vic = new Contact (FirstName='Vic', LastName= 'Doe');
Set<Contact> contacts = new Set<Contact> { josh, vic};
```

2.2.1 Set Methods

Set<Integer> myList= new Set<Integer>();

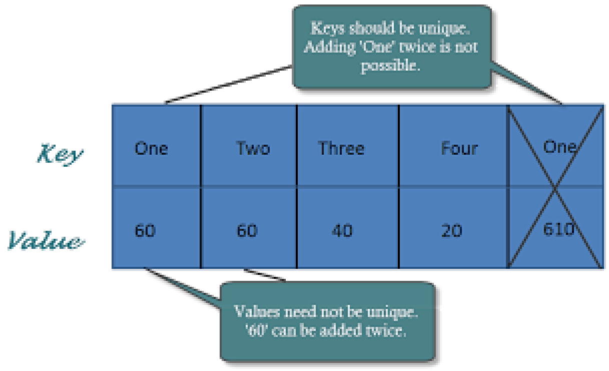
Method	Example
Add (setElement)	myList.add(25);
Remove(setElement)	myList.remove(25);
Clear()	myList.clear();
isEmpty ()	Boolean isEmpty = myList.isEmpty();
Size ()	Integer setSize = mySet.size();
contains(setElement)	Boolean contain = myList.contains(25);

2.2.1 Examples

```
//empty set initialized
Set<String> strSet = new Set<String>();

//elements added to the set
strSet.add('Paris');
strSet.add('London');
strSet.add('New York');

//duplicate element added to the set
strSet.add('Paris');

//this will output size of set
Integer setSize = strSet.size();
//the result will be 3
system.debug('@@size : ' + setSize);

system.debug(strSet.contains('New York'));

//remove value from list
strSet.remove('New York');
system.debug(strSet.contains('New York'));

//retain all method
Set<integer> mySet = new Set<integer>{1, 2, 3,4,5,6};
List<integer> myList = new List<integer>{1, 2,3,4};
Boolean result = mySet.retainAll(myList);
System.assertEquals(true, result);
```

2.3 Maps

- Collection of Unique Key-Value pairs, where each key is Unique and maps to a Non-unique value
- Keys are Unordered and Unique / equivalent to Sets (UU)
- Values are Ordered and NON-Unique / equivalent to Lists (ON)
- Can also contain any collection and can be nested within one another and become multidimensional.


Keys should be unique. Adding 'One' twice is not possible.
Key
Value
One
Two
Three
Four
One
60
60
40
20
610
Values need not be unique. '60' can be added twice.

```
//Create a Map:

Map<Integer,String> myMap = new Map<Integer,String>();
Map<Integer,String> myMap2 = new Map<Integer,String> {1 => 'one', 2 =>'two'};

Map<Integer,String> myMap = new Map<Integer,String>();
myMap.put(1, 'one');
myMap.put(2, 'two');
myMap.put(2, 'three');
```

2.3.1 Map Methods

Method	Example

Put (key, value)	Map<String, String> colorCodes = new Map<String, String>();
	
	colorCodes.put('Red', '#ff0000');
	colorCodes.put(‘Blue', '#FF0000');

Get (key)	String code = colorCodes.get('Blue');
Remove (key)	colorCodes.remove('Blue');
Clear()	colorCodes.clear();
isEmpty ()	Boolean isEmpty = colorCodes.isEmpty();
Size ()	Integer mapSize = colorCodes.size();
Keyset()	Returns a **set** that contains all of the keys in the map.
Values()	Returns a **list** that contains all the values in

	the map.

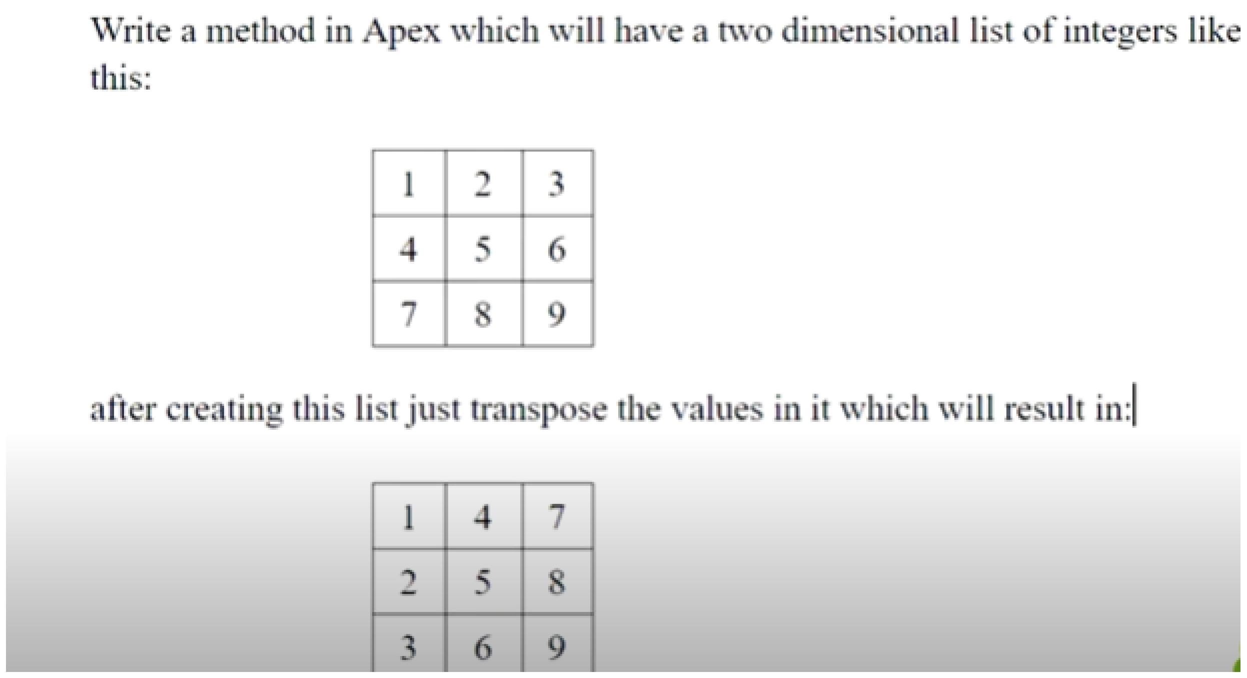
2.3.2 Map Examples

```
//Initialization of Map
Map<String, String> stringMap = new Map<String, String>();

//setting key value pairs in the Map
stringMap.put('Spielberg', 'Jurassic Park');
stringMap.put('Nolan', 'Dark Knight');
stringMap.put('Cameron', 'Avatar');

//accessing the value for key
system.debug(stringMap.get('Nolan'));

//this will check if the key exists in the map or not, will show false
system.debug(stringMap.containsKey('John'));
```

2.3.3 Preview of SOQL/MAP

```
Map<Id,Account> testmap = new Map<Id,Account>([Select Id,Name from Account limit 5]);
system.debug('The size of the map is'+testmap.size());
system.debug('The keys are'+testmap.keySet());
system.debug('The values are'+testmap.values());
```

2.4 Problem


Write a method in Apex which will have a two dimensional list of integers like this:

1	2	3
4	5	6
7	8	9

after creating this list just transpose the values in it which will result in:

1	4	7
2	5	8
3	6	9

```
public class twodlistandtranspose {
   public static void main() {
      List<List<Integer>> twodlist = new List<List<Integer>>();
      List<Integer> l1 = new List<Integer>{1,2,3};
      List<Integer> l2 = new List<Integer>{4,5,6};
      List<Integer> l3 = new List<Integer>{7,8,9};
      twodlist.add(l1);
      twodlist.add(l2);
      twodlist.add(l3);
      system.debug('Original list is: '+ twodlist);
      for(integer i=0; i<3; i++) {
         for(integer j=0; j<3; j++) {
            if(i < j) {
               integer temp = twodlist[i][j];
               twodlist[i][j] = twodlist[j][i];
               twodlist[j][i] = temp;
            }
         }
      }
      system.debug('List after transpose is: '+ twodlist);
   }
}
```

3. Control flow statements

- Cause Apex code to execute based on a certain condition

- Have a block of code execute repeatedly.

3.1 If / If else loop

```
//If Statement

Integer grade = 75;
if (grade >= 60) {
        System.debug('You have passed');
}

//If Else Statement
Integer grade = 75;
if (grade >= 60) {
        System.debug('You Pass');
}
else {
        System.debug('You Fail');
}

//If-ElseIf-Else statement

if (grade >= 90) {
        System.debug('Your grade is A');
}
else if (grade >= 80) {
        System.debug('Your grade is B');
}
else if (grade >= 70) {
        System.debug('Your grade is C');
}
else if (grade >= 60) {
        System.debug('Your grade is D');
}
else {
        System.debug('You Fail!'); }
```

3.2 While loop

- Executes a statement If the Boolean test returns True
- Execution will stop when the value of condition becomes False.
- The test takes place before each iteration.

```
**//Syntax**

while (Boolean_condition) {
          statement;
}

**//Example**

Integer count = 1;
while (count < 11) {
          System.debug('Count is now: ' +count);
          count++;
}
```

3.3 Do - While loop

- A do-while loop is similar to the while loop, except that a do-while loop is guaranteed to execute at least one time
- The do-while loop checks the condition at the bottom of the loop.

```
**//Syntax**

do {
          statement;
} while (boolean_condition);

**//Example**

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

3.4 Switch Statement

Apex provides a switch statement that tests whether an expression matches one of several values and branches accordingly.

```
//Syntax
```

```
switch on expression {
    when value1 {                    // when block 1
        // code block 1
    }
    when value2 {                    // when block 2
        // code block 2
    }
    when value3 {                    // when block 3
        // code block 3
    }
    when else {                // default block, optional
        // code block 4
    }
}
```

The when value can be a single value, multiple values, or sObject types. For example:

```
//Condition 1

when value1 {
}


//Condition 2

when value2, value3 {
}

//Condition 3

when TypeName VariableName {
}
```

```
//Example

switch on i {
   when 2, 3, 4 {
       System.debug('when block 2 and 3 and 4');
   }
   when 5, 6 {
       System.debug('when block 5 and 6');
   }
   when 7 {
       System.debug('when block 7');
   }
```

```
  when else {
     System.debug('default');
  }
}


//Switch Statement with Sobject

switch on sobject {
  when Account a {
     System.debug('account ' + a);
  }
  when Contact c {
     System.debug('contact ' + c);
  }
  when null {
     System.debug('null');
  }
  when else {
     System.debug('default');
  }
}
```

3.5 For loop

Apex supports three variations of the for loop

1. Traditional For loop
2. For each loop (List/Set)
3. SOQL For loop

3.5.1 For loop (traditional)

```
//Syntax

for (init_stmt ; exit_condition ; Increment_stmt) {
code_block;
}
//Example

for (Integer i = 0 ; i < 10 ; i++) {
          System.debug('Num is: ' +i);
}
```

3.5.2 For each loop (List_Set)

```
//Syntax

for (variable : list_or_set) {
          statement;
}
//Example

List<Integer> myIntList = new List<Integer> {1, 2, 3, 4, 5..500};
for (Integer i : myIntList) {
          System.debug(i);
}
// Returns 1 2 3 4 5

//For maps
for(Id id: mapname.values()){
//Code here
   }
```

3.5.3 SOQL For loop

The SOQL For loop iterates over all of the sObject records returned by a SOQL query.
//Intro about SOQL//

```
//Syntax

For (variable : [soql query] ) {
b.statements;
}

//Example1

for (Account a : [SELECT Id, Name FROM Account WHERE Name LIKE 'ACC%']) {
    a.Name = 'ACC Global ';
}
```

```
//Example 2

// Create a list of account records from a SOQL query
List<Account> accs = [SELECT Id, Name FROM Account WHERE Name LIKE 'ACC%'];

// Loop through the list and update the Name field
for(Account a : accs){
  a.Name = 'ACC Global';
}
```

//Why method1 of SOQL is preferred:

- In example 2,if accs list variable is holding 50,000(max) account record, if we say ,if each Record Is taking 2KB , the total Size will be 50,000 * 2 = 100,000 KB means approx 100 MB but allowed Limit is 6 MB (for synchronous Process) .Hence, a Variable accs is holding data of heap size 100 MB.So , Error will be Encountered.
- In example 1 Here also the Soql query will return 50,000 (max) record, but the variable a is holding one record at a time means 2 KB. So , Basically we are not hitting Heap Size governor Limit here.So, to avoid Hitting Heap Size Limit , Always use SOQL for Loops.

//Concepts on list of SOQL records:

- The single sObject format executes the for loop's <code_block> once per sObject record
  for (**Account** acc : [SELECT Id FROM Account WHERE Name = 'Acme']) {}

- The sObject list executes the for loop's <code_block> once per list of 200 sObjects.
  for (**Account[]** acc : [SELECT Id FROM Account WHERE Name = 'Acme']) {}

3.6 Break Statement

- It terminates the loop and goes out of it immediately when it appears
- Available for:
  - For loops
  - While loop
  - Do-While loop
- Break will not allow to continue iterating, even if the Boolean condition is still true

```
//Example

for(Integer i=0; i<100; i++) {
        if(i==10)
        break;
        system.debug('i value: ' + i);
}
```

```
//Will only show 0 to 9, not 10 and not anything
// after 10
```

3.7 Continue Statement

- It skips the current iteration and goes to the next.
- Available for:
  - For loops
  - While loop
  - Do-While loop
- Continue will immediately exit the current iteration and moves to the next as if the current iteration has ended

```
//Example

List<Integer> myIntList = new List<Integer> {1, 2, 3, 4, 5};
for(Integer x : myIntList) {
        if ( x == 3 ) {
                continue;
        }
        System.debug(x);
}
```

3.8 Summary

Apex supports the following five types of procedural loops:

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

All loops allow for loop control structures:

- break; exits the entire loop
- continue; skips to the next iteration of the loop