**Apex:**

Day 1:

* For Decimal var.setscale(2) will reduce decimal to 2 digits
* Modulus🡪Math.mod(number,number)

Day 4:

* Switch will not support decimal,Boolean
* Ternary operator:(Condition)?true:false
* Apex is not case sensitive to variables but case sensitive to values

Day 6:

* Oops
* Encapsulation🡪Grouping of function and data
* Abstraction 🡪Hiding complex information
* Inheritance🡪Eliminate redundant code
* Polymorphism🡪Different forms in base code

Day 7:

* Class Level Variable: Can be accessed throughout the class
* Function Level Variable: Can be accessed only inside the function
* Function overloading: more than function with same name but with different parameters

Day8:

* Constructor🡪Initialize member variable.

Day 9:

* Static block: Can be used before constructor, static block will execute only once in entire transaction context,Initialized only once,retain its last value.common memory
* Outside the class we can access public static variable using classname.static variables
* Static to non static functions is not possible even inside the class,,we have to create an object inside the class and access non static functions
* Non static functions to static functions there will be no restriction
* **This** keyword cannot be used for static variables

Day 10:Get,Set

* If we want to make the public variable not accessed in anonymous window that is **@read only** mode then use **get method inside the variable**
* To set the get variable inside the function then use **private set** so it variable can be set for any function inside the program
* Inheritance🡪
* Code reusability
* Helps to extend the functionality

Day11:

* Inheritance:
* Code reusability
* Capability to create new classes
* Helps to extend the functionality
* Override
  + Having different definition for super class(parent class) functions
  + If we use **protected** function in child from parent then we can use it in child class but cannot access in anonymous window
  + Cannot extend more than one class which is called multilevel inheritance
  + Parent class should have virtual keyword to extend it to child class
  + Function from parent class should have virtual keyword to override the same function in child class

Day 12:

* Abstract
* Object cannot be created
* It should be extended by other classes
* Keyword: **Abstract**
* All the classes extending the parent class should have the definition for the abstract method(i.e.,overriding)
* Use abstract method in parent class
* Abstract method cannot have a body
* Interface:
* Interface is a contract which classes will agree and follow
* Advantage:A class can implement more than one one interface
* Interface should not be a fat one
* Replace **class** with **Interface**
* **Keyword:Implements**

Day 13:

* Clean Code Base
  + Keep method short
  + Function should do one thing at a time(single responsibility)
  + Align code(Shift+Tab)
  + Store hard coded value at the top and use the same variable inside the class
  + Don’t repeat the same code again
  + Proper Naming Convention  
    classname –CustomerInfo  
    Method Name – calculateInterest
  + Don’t use comment all over the class

Day 14:

* Collections:
  + List: ordered collection, distinguished by indices
  + Set: Unordered collection, does not contain duplicates
  + Map: Collection of key value pairs

Day20:

* **Instanceof**: Used to check and provide the datatype result.
* Ex: variable **instanceof** String 🡺true
* When we use **return** inside a void function it moves out of the function
* If we use string == string then it is not case sensitive
* String.equals(string) then it is case sensitive
* String.equalsignorecase then it is not case sensitive
* **Dynamic template: Format**

String message=’course {0} Status:Course completed {1} out of {2}’;

List<object>data =new List<object>{‘Apex’,20,30};

**String.format(message,data);**

* **String.capitalize**🡪converts first letter to capital letter
* String.contains(‘string’)🡪case sensitive
* String.deletewhitespace()🡪Delete whitespaces in string
* String.normalizespace()🡪retain correct space
* String.indexof(‘string’)🡪checks index of string in variable
* String.lastindexof(‘string’)🡪 checks last index of string in variable
* String.isblank(variable)
* Variable.split(‘ ’)🡪list
* String.join(variable,#)

Day 21:

* Date and time
* Two ways:

GMT or UTC

Timezone based(offset with ref toGMT)

* In SF Timezone settings
  + Company Setting
  + User Settings
* If DST(Daylight saving time) – SF will take care of rotating clock
* How SF store date and time
  + SF will store date and time in GMT format
  + When displaying they show it in user timezone in UI
  + Timezone variable=userinfo.getTimezone();🡪user time zone
  + Tz.getId()
  + Tz.getDisplayName()
* Todays date
  + Date.today()(2023/11/3 00:00:00)
  + System.today()
  + Both are user specific
* Date based on user specific
  + Date.today().format provide date based on user specific(e.g.,1/11/2023)🡪locale specific
* Construct Date
  + Date td=Date.newInstance(YYYY,MM,DD);
  + Td.day()🡺Provide day of the date
* Date arithmetic
  + Date.today()+7
  + Today.addMonths(2)
  + Similarly add Years also we can use
  + Date difference🡪 today.daysbetween(enddate)

Day 22:

* Convert Date to string
  + String.valueof(system.today())
* Convert string to date
  + Date.parse(string)🡪based on **locale** date format we have to give string input
  + Date.valueof(string)🡪based on iso format eg;yyyy-mm-dd
* Get start of the week
  + Date.today().tostartofweek();//locale specific US-Sun,UK-Mon
* DateTime:
  + DateTime.now() //GMT
  + System.now //GMT
* DateTime based on user specific
  + DateTime.now().format() //convert into user time zone🡪string format
* Time:
  + DateTime.now().time() //User time zone
  + DateTime.now().format(‘hh:mm’) //User time zone
* Construct Date and time
  + DateTime dt= DateTime.newInstance(2024,2,28,15,0,0) //GMT
  + Dt.format() //User time zone
* DateTime Arithmetic
  + Add 3 Days 2 Hours 30 Minutes
  + DateTime.now().addDays(3).addHours(2).addMinutes(30)
* Date Difference
  + getTime 🡪 datetimeVariable.getTime() // milliseconds from unix time stamp(1900,1,7 - Sunday)
  + 1 sec = 1000 milliseconds
  + Long seconds=(toDate.getTime()-fromDate.getTime())/1000 //Seconds
* How to find week day number for a particular date
  + Date dt=Date.today();
  + No of days::Date.newInstance(1900,1,7).daysbetween(dt)
  + Math.mod(Date.newInstance(1900,1,7).daysbetween(dt),7) //Sun – 0,Mon – 1, and so on…

Day 23:

* SOQL:Structured object query language
* SOQL is used to query objects from SF Org
* DML:Data Manipulating Language

Day24:

* Singlerecord.get(‘Industry’)
* Maximum records of 50000 will be fetched by SOQL and accommodate in a list
* Exception:Too many SOQL 101 🡪100 SOQL in 1 transaction

Day 26:

* List<Account> accList=[SOQL];
* Map<Id,Account> accMap=new Map<Id,Account>([SOQL]);

Day27:

* Parent to child:
  + Standard- standard:Select Name,(Select Id,Name from Contacts) From Account
  + Standard - Custom:Select Name,( Select Id,Name from Departments\_\_r) From Account
  + Custom-custom: Select Name,( Select Id,Name from Departments\_\_r) From Employee\_c

Day28:

* Child to Parent:
* Standard-Standard:
  + Select LastName,Account.Name From Contact
  + Select LastName,Account.ownerId From Contact
  + Select LastName,Account.owner.Name From Contact
  + Select LastName,Account.owner.Manager.Name From Contact
* Custom-Standard:
  + Select Name,Account\_\_r.Name From Contact

Day 29:

* Semi Join:
  + It’s a subquery on another object in an IN Clause

[Select Name,AccountId,Account.Rating From Contact  
WHERE **AccountID IN(Select Id from Account Where Rating=’Hot’**)]

* Anti Join:

[Select Name,AccountId,Account.Rating From Contact  
WHERE **AccountID NOT IN(Select Id from Account Where Rating=’Hot’**)]

* Aggregete Function:
  + Count:

Integer countofContacts=[Select **count()** from Contact]; //When not assigning a field inside count() provide integer

List<AggrgateResult> countofContacts=[Select **count(Id)** from Contact]; //When assigning a field inside count() provide list<AggrgateResult>

List<AggrgateResult> countofContacts=[Select **count(Id) ratingCount** from Contact]; //Alliasing

System.debug(countofContacts[0].get(‘ratingCount’))

* Having:Used to filter aggregated result

Day 30:

* AggregateResult cannot be applied in child relationship query
* AggregateResult query cannot have child relationship query in the selection list
* Fields Function:
  + Fields(All)-Must use limit keyword🡺Select Fields(All) from Account Limit 200
  + Field(Standard)- Don’t need of limit keyword 🡪 Select Fields(Standard) from Account
  + Field(Custom)- Must use limit keyword 🡪 Select Fields(Custom) from Account Limit 200
* Apex will not support field level security since it runs from system context mode but FIELDS keyword will support field level security
* Mixed Fields:
  + Select Name,Phone,FIELDS(Custom) from Account Limit 200
  + Select SLA\_\_c,FIELDS(Standard) from Account Limit 200
* Dynamic SOQL:
  + Creation of SOQl Query at run time with APEX Code
* Escape Sequence:\
* We can use variable inside the query string while using it in database.query

Day31:

* Dynamic SOQl supprts Aggregate Query
* Sobject :
  + SobjectList.getsobjecttype()
  + Record.get(‘name’) or typecast it
* SOSL:Salesforce object search language
  + Use SOSL to search fields in multiple objects irrespective of the relationship
  + Provide word match not exact match
  + Return type:List<List<sobject>>

Day32:

* Day\_Only:
  + Use Day\_ONLY when u need to get a records based on particular date

List<AggregateResult> accList=[Select Name,DAY\_ONLY(createdDate)

From Account

Where DAY\_ONLY(createdDate)=:recordCreatedDate

Group By Name,DAY\_ONLY(createdDate)];

Day33:

* Convert time Zone
* DML Operations:
  + Insert
  + Update
  + Upsert
  + Delete
  + Undelete
  + Merge
* Gov Limit:150 DML in a single transaction

Day 34:

* When a code breaks while creating child record then the parent record also breaks during a single transaction
* Use it in try catch block to make a transaction to commit

Day 35:

* When updating the record we can update the non queried field in SOQL
* **ALL ROWS**🡪
  + Will fetch all records both **live and deleted record**
  + Use **ALL ROWS with keyword isDeleted=true** in a query to get only the deleted record
* **MERGE🡪**
  + Merge upto 3 records of SF sOBJECT
  + Only leads,Account and Contacts
  + Merger MasterRecord, List<Other records to be merged?>
  + When merging the records the child records from other record will be reparented to the master record but the fields will not be overwritten master record will retain its field value
* **Partial Insert:Database(AllorNothing=False)🡪**
  + Database.insert(accList,False)
  + While using in batch its more wise to use Database.insert(accList,False)
* **Database.SaveResult🡪**
  + Provide **success and failure result in database.insert**
  + Database.SaveResult[] sr =Database.insert(accList,False)

Day 36:

* To get error from **Database.SaveResult**
* When validation rule fires we cannot get **getfields(=NULL) in geterrors**
* Transaction control using **savepoint**:
  + When u want to roll back in a transaction if there is any error in the transaction then set the savepoint before the code starts for the transaction
  + **System.savepoint sp=Database.setsavepoint();**
  + Use **Database.rollback(sp);** at end of the transaction
* **Apex Security:**
  + Object,Field level security🡪Profile,Permission Set
  + Record Level-Sharing settings
  + Apex will run in system context mode(will not respect security rules)
  + Anonymous window respect sharing rule
  + **Using SOQL: WITH SECURITY\_ENFORCED🡪**will provide only fields which are accessible

Day37:

* When we use **WITH SECURITY\_ENFORCED** there may be chances for the code to break due to field/object level security …Use **Schema** class to overcome
* Schema has the all metadata function in the schema class
* For Field
  + Ex:…,, **Schema.SObjecttype.Account.Fields.Phone.isAccessible()**
  + **isCreatable,isUpdatable,isDeletable**
* For Object
  + Ex:…,, Schema.SObjecttype.Account.isAccessible
* **With USER\_MODE**🡪SOQL will run in user context
* **With SYSTEM\_MODE**🡪SOQl will run in system mode
* **Security.StripInaccessible**🡪This method will discard the fields(Inaccessible) from the result
  + sObjectAccessDecision d=Security.stripInaccessible(Accesstype.READABLE,accList)
  + d.getRecords
* Record level security:
  + With Sharing,Without Sharing
  + Default🡪**without sharing**(Does not respect record level security)
  + **With sharing**(respect record level security)
* **Class Level Security:**
  + When the logged in user doesn’t have the access to apex class then the class cannot be used
  + Setup🡪apex class🡪every class has security 🡪click🡪Add profiles to enable the permission for class
  + Setup🡪VisualForce Pages🡪 security🡪 Add profiles
  + When the controller class with sharing calls the without sharing helper then the helper runs based on with sharing since it is called from a with sharing mode

Day38:

* **Exception Handling:**
  + **Run time error**🡪
    - Unexpected event that occurs when code is executing
    - Code will stop executing there itself
    - Any DML Operation performed will be rolled back
  + **Handling:**
    - Exception that can be handled
    - Exception that cannot be caught(Limit Exception)
* **Most Common Exception:**
  + DML Exception
  + Query Exception
  + Nullpointer Exception
  + Indexoutofbounds
* **Exception e**
  + e.getMessage()
  + e.getLineNumber()
* **Try Block** can handle multiple catch(exceptions)
* **Finally** blockwill execute irrespective of the exception
* **Custom Exception:**
  + Create a class then extend an exception class
  + We can override exception message
  + If there is no method in the exception class then if we execute

**Throw new MyExceptionClass();**

* + Then it shows **script thrown exception**

Day 39:

* **Wrapper Class:**
  + Class used to create a datatype with other datatypes as member
  + Wrapper class is used as an inner class
  + Inner class is used to represent data doesn’t have any business logic
  + Use public class and variable for inner class to access it

Day 40:

* To expose the method in flow then we should use annotation for the method as **@InvocableMethod**
* To expose the variable in flow then we should use annotation for the variable as **@Invocablevariable**
* Use **@Invocablevariable(required=true)** to make the variable as required

Day 41:

* **Governor Limit:**
* **Limit class:**
  + **Limits.getLimitQueries**
  + **Limits.getQueries**
* **Custom Label:**
  + Helps to avoid hardcoding values
  + Advantage:No SOQL required
  + Can be deployed in production environments
  + Disadvantages:Single Value
  + **Label.MycustomLabel**

Day42:

* **Cusom Settings:**
  + **Helps to avoid hard coding values in apex**
  + **Two types:**
    - List
    - Hierarchy-Supports profile/user specific value
  + **Advantage:**
    - Multiple values can be stored
    - No SOQL required
    - Profile based value setting
  + **Disadvantage:**
    - Only metadata can be migrated
    - Values are not accessible from test classes
    - Does not support lookup field
  + Quick Find box🡪Schema settings🡪Enable manage list custom setting type
  + **CusomSettings.getAll()**
  + **CusomSettings.getAll().values🡪**return records of custom settings

Day43:

* **Cusom Settings:**
  + Hierarchy-Supports profile/user specific value
  + **CusomSettings.getInstance()🡪Current users instance**
  + **CusomSettings.getInstance(UserId or ProfileId)**
* **Cusom Metadata:**
  + Object Name ends with **\_\_mdt**
  + **Advantage:**
    - Support multiple values
    - Support lookupfield column(only to another metadata not to another objects)
    - Deploy with prod with values
    - No SOQL Required
    - Values can be accessed from test classes
    - Supports validation rule,pagelayout
  + **Disadvantage:**
    - Only 100 metadata are allowed
    - Max 10 MB total size
    - Not easy to create/update/delete
    - Does not support formula field
    - Can create only 100 fields per custom metadata type
    - Global picklist are not supported
  + **Metadata.getAll()**

**Day 44:**

* **Apex Sharing:**
  + Configuration:Sharing setting,Manual sharing
  + Programmatic:Apex sharing
* Types of sharing:
  + Managed Sharing:
    - Sharing set by the lightning platform
    - Record owner,sharing rule,role hierarchy
  + User Managed sharing:
    - See PDF
  + Apex Managed Sharing
    - See PDF
* Share Object:
  + It includes records supporting all three types of sharing
  + For Standardobject: standardobjectnameshare
  + For customobject: customobjectName\_\_share
  + Access:**Read,Edit,ALL**
  + **ParentID:**Record to be shared,can be create/cannot update
  + Rowcause:See PDF
* Create Apex sharing reason(For custom object)🡪Switch to salesforce classic
  + Setup🡪left side🡪build🡪create🡪choose objects🡪page down🡪Apex sharing reason🡪new

Day45:

* Test Class:
  + @isTest
  + If we want to see org data in test method use @isTest(seeAllData=True)(Not a best practice)
  + @testSetup
  + System.assertEquals(Exp Ans,actual value,error)

Day46:

* Test.starttest()
* Test.stopTest()
* Reset governor limit

Day47:

* Test.loadData(Schema.sObjectType,Static resource)