

1. Explain exception handling in python?

An exception is an error which happens at the time of execution of a program. However, while running a program, Python generates an exception that should be handled to avoid your program to crash. In Python language, exceptions trigger automatically on errors, or they can be triggered and intercepted by your code.

The exception indicates that, although the event can occur, this type of event happens infrequently. When the method is not able to handle the exception, it is thrown to its caller function. Eventually, when an exception is thrown out of the main function, the program is terminated abruptly.

Common Examples of Exception

- Division by Zero
- Accessing a file which does not exist.
- Addition of two incompatible types
- Trying to access a nonexistent index of a sequence
- Removing the table from the disconnected database server.
- ATM withdrawal of more than the available amount

Exception handling is managed by the following 5 keywords:

- 1.try
- 2.catch
- 3.finally
- 4.throw

Python Try Statement

A try statement includes keyword try, followed by a colon (:) and a suite of code in which exceptions may occur. It has one or more clauses. During the execution of the try statement, if no exceptions occurred then, the interpreter ignores the exception handlers for that specific try statement.

In case, if any exception occurs in a try suite, the try suite expires and program control transfers to the matching except handler following the try suite.

Syntax:

try:

statement(s)

The catch Statement

Catch blocks take one argument at a time, which is the type of exception that it is likely to catch. These arguments may range from a specific type of exception which can be varied to a catch-all category of exceptions.

Rules for catch block:

- You can define a catch block by using the keyword `catch`
- Catch Exception parameter is always enclosed in parentheses
- It always represents the type of exception that catch block handles.
- An exception handling code is written between two `{ }` curly braces.
- You can place multiple catch block within a single try block.
- You can use a catch block only after the try block.
- All the catch block should be ordered from subclass to superclass exception.

Example:

```
try
}

catch (ArrayIndexOutOfBoundsException e) {

System.err.println("Caught first " + e.getMessage()); } catch (IOException e) {

System.err.println("Caught second " + e.getMessage());

}
```

Finally Statement in Python

- Finally block always executes irrespective of an exception being thrown or not. The `finally` keyword allows you to create a block of code that follows a try-catch block.
- Finally, clause is optional. It is intended to define clean-up actions which should be that executed in all conditions.

try:

raise KeyboardInterrupt

finally:

print 'welcome, world!'

Output

Welcome, world!

KeyboardInterrupt

Finally, clause is executed before try statement.

Raise Statement in Python

The raise statement specifies an argument which initializes the exception object. Here, a comma follows the exception name, and argument or tuple of the argument that follows the comma.

Syntax:

```
raise [Exception [, args [, traceback]]]
```

In this syntax, the argument is optional, and at the time of execution, the exception argument value is always none.

Example:

A Python exception can be any value like a string, class, number, or an object. Most of these exceptions which are raised by Python core are classes with an argument which is an instance of the class. Other Important Python Exceptions

Exception	Description
ArithmeticException	Arithmetic error, such as divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
ArrayStoreException	Assignment helps you to the array element of an incompatible type.
ClassCastException	Invalid cast
MlegalMonitorStateException	Illegal monitor operation, like waiting on an unlocked thread.
ClassNotFoundException	Class not found.
InstantiationException	Occurs when you attempt to create an object of an interface or abstract class.

2. Differentiate between errors and exceptions in python?

Error	Exception
An error cannot be handled at runtime.	An exception can be handled at runtime
An error can occur both at	Although all the Exceptions occur at

Error	Exception
compile time and during runtime.	runtime. But checked Exceptions can be detected at compile time.
There are 3 types of Errors: Syntax Error , Runtime Error and Logical Error	There are 2 types of Exceptions: Checked Exceptions and Unchecked Exceptions
An error has the capacity to terminate your program and maybe your system as well.	An exception has the capacity to distract the normal flow of the program and change its direction to somewhere else when an exceptional case has occurred.
An Error is such an event that no one can control or guess when it is going to happen.	An Exception can be guessed, handled, and utilized in order to change the original flow of the program.
An Error can be thought of as an explosion that happens when there is no defense or checks against a particular failure condition.	An Exception can be thought of as a last line of defense to prevent errors.

3. Write a Python program to demonstrate Multiple Inheritance.

if a child class inherits from more than one class, i.e. this child class is derived from multiple classes, we call it multiple inheritance in Python. This newly derived child class will inherit the properties of all the classes that it is derived from. A very simple real life example would be the one discussed above - a child inheriting properties of the father and mother.

example of Multiple Inheritance in Python

Moving on to the syntax of multiple inheritance in Python, it is much similar to that of single inheritance. For this example, we're not adding any code to the class, the 'pass' statement will simply skip the code and move ahead.

```

class BaseClass1:

    # Body of base class 1 (parent class 1)

class BaseClass2:

    # Body of base class 2 (parent class 2)

class DerivedClass(BaseClass1, BaseClass2):

    # Body of derived class (child class)

    # Inherited properties of baseclass1 and baseclass2

```

Here, we have defined two classes which would be the parent classes of our new derived class. The derived class will have the properties of BaseClass1 as well as BaseClass2 as specified in the code above, i.e. the child has properties of both parents. Let us now look at an example of multiple inheritance in Python.

```

# creating class for father
class Dad():
    # writing a method for parent class 1
    def singing(self):
        print("Dad sings well")

# creating a class for mother
class Mom():
    # method for parent class 2
    def coding(self):
        print("Mom codes well")

# creating derived class
class Child(Dad, Mom):
    def play(self):
        print("Kid loves to play")

# creating object of the new derived class
child = Child()
# calling methods of parent classes and derived class
child.singing()
child.coding()
child.playing()

```

Output:

```

Dad sings well
Mom codes well
Kid loves to play

```

As we can see in the output, the child class that was derived from Dad() and Mom() classes have the properties of both the parent / base classes.

4. How are classes created in python?

Object-Oriented Programming in Python involves object creation through **Class**, **Inheritance**, **Polymorphism**, and **Encapsulation**. A class is like a blueprint that bundles different types of items under one roof. Python can have multiple classes in one Python script. Each class contains its own attributes, instances, and methods to maintain the state of the class. Python classes have the ability to protect the data from outside threats. Classes are accessed from an entity called **objects**. It has methods called **member functions** to modify the state of the class. Assume your class to be a house that contains house details like room, kitchen, doors, etc. Likewise, Class has different attributes called **data members** that tell us about the class.

Class Real World Example

Let us understand a real-world example to know the importance of object-oriented Class. Assume there are 50 students in one class and the teacher wants to store, manage, and maintain marks of each subject scored by 50 students. In order to maintain this large data, classes are introduced because they combine data together and provide data organization. The class creates objects for its functioning. The class will define properties related to students like name, roll no, marks, etc. under one roof and then access the information by using objects created.

Now, let us see how classes are created, what important points should one keep in mind before creating a class, and then what another functionality class provides to us.

Creating a Class in Python

A program of a class is generally easy to read, maintain, and understand. A class, functions as a template that defines the basic characteristics of a particular object.

Important Points

1. Classes are created using **class** keyword.
2. A **colon (:)** is used after the class name.
3. The class is made up of attributes (data) and methods (functions).
4. Attributes that apply to the whole class are defined first and are called **class attributes**.
5. Attributes can be accessed using the dot **(.)** operator via objects.

Let us understand the concept of the **'Dog'** class using a simple code.

Example: Creating Python Class

```
#class is defined using class keywordclass Dog:
```

```

#data members of class

color = "black" #attribute 1

name = "Polo"    #attribute 2


#class constructor

def __init__(self):

    pass


#user defined function of class

def func():

    pass

```

We take a class and named it "Dog". We defined two attributes or two instances of the Dog class that store `color` and `name`. This is the simplest template of a class. Further, we defined a constructor that uses `__init__` for its declaration. After this, the user can create their own function called member functions of the class and perform different operations on the attributes defined inside the class. We left these two functions empty and will learn more about them in another article.

As you can see in the above example, all the different attributes or properties of the dog are placed together as a bundle. These attributes are like class variables that are local to his class. When a class defined, a new `namespace` is created and used as the local scope thus, all assignments to local variables (attributes here) go into this new namespace and then accessed further with the help of an object. We will learn more about class objects in the next article.

5. What are different types of inheritance supported by Python? Explain?

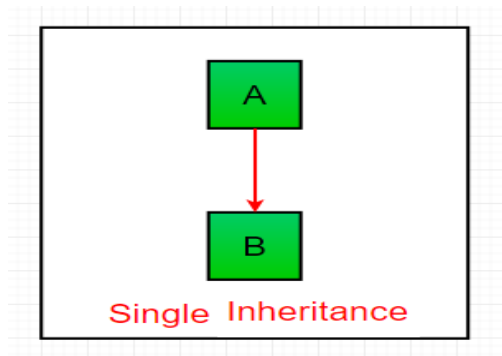
Inheritance is defined as the mechanism of inheriting the properties of the base class to the child class.

Types of Inheritance in Python

Types of Inheritance depend upon the number of child and parent classes involved. There are four types of inheritance in Python:

1.Single Inheritance:

Single inheritance enables a derived class to inherit properties from a single parent class, thus enabling code reusability and the addition of new features to existing code.



Example:

Python program to demonstrate

single inheritance

Base class

class Parent:

def func1(self):

print("This function is in parent class.")

Derived class

class Child(Parent):

def func2(self):

print("This function is in child class.")

Driver's code

object = Child()

object.func1()

object.func2()

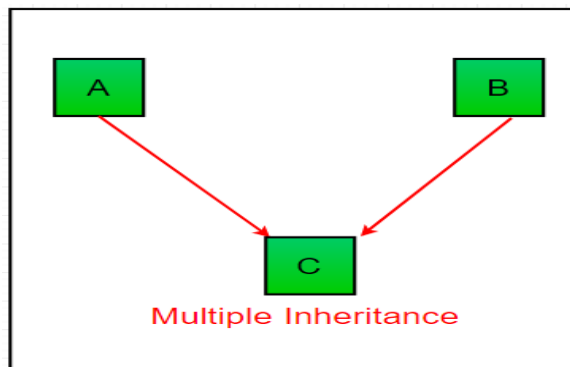
Output:

This function is in parent class.

This function is in child class.

2. Multiple Inheritance:

When a class can be derived from more than one base class this type of inheritance is called multiple inheritances. In multiple inheritances, all the features of the base classes are inherited into the derived class.



Example:

Python program to demonstrate

multiple inheritance

Base class1

class Mother:

 mothername = ""

 def mother(self):

 print(self.mothername)

Base class2

class Father:

 fathername = ""

 def father(self):

 print(self.fathername)

Derived class

class Son(Mother, Father):

 def parents(self):

 print("Father :", self.fathername)

 print("Mother :", self.mothername)

Driver's code

```
s1 = Son()

s1.fathername = "RAM"

s1.mothername = "SITA"

s1.parents()
```

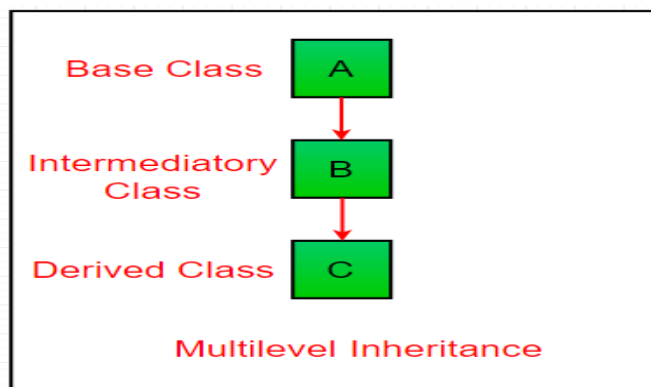
Output:

Father : RAM

Mother : SITA

3.Multilevel Inheritance :

In multilevel inheritance, features of the base class and the derived class are further inherited into the new derived class. This is similar to a relationship representing a child and a grandfather.



Example:

```
# Python program to demonstrate
# multilevel inheritance

# Base class
class Grandfather:

    def __init__(self, grandfathername):

        self.grandfathername = grandfathername

# Intermediate class
class Father(Grandfather):

    def __init__(self, fathername, grandfathername):

        self.fathername = fathername
```

```

        # invoking constructor of Grandfather class

        Grandfather.__init__(self, grandfathername)

# Derived class

class Son(Father):

    def __init__(self, sonname, fathername, grandfathername):

        self.sonname = sonname

        # invoking constructor of Father class

        Father.__init__(self, fathername, grandfathername)

    def print_name(self):

        print('Grandfather name :', self.grandfathername)

        print("Father name :", self.fathername)

        print("Son name :", self.sonname)

# Driver code

s1 = Son('Prince', 'Rampal', 'Lal mani')

print(s1.grandfathername)

s1.print_name()

```

Output:

Lal mani

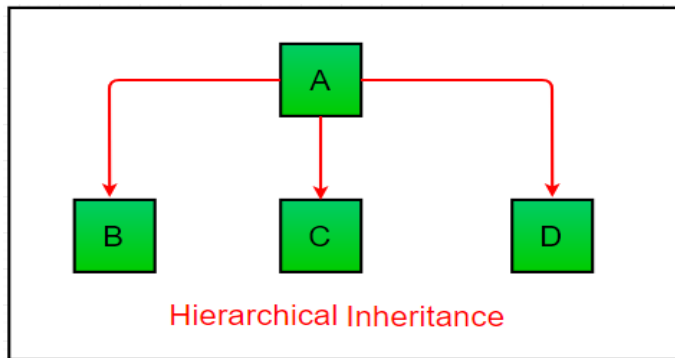
Grandfather name : Lal mani

Father name : Rampal

Son name : Prince

4.Hierarchical Inheritance:

When more than one derived class are created from a single base this type of inheritance is called hierarchical inheritance. In this program, we have a parent (base) class and two child (derived) classes.



Example:

Python program to demonstrate

Hierarchical inheritance

Base class

class Parent:

def func1(self):

print("This function is in parent class.")

Derived class1

class Child1(Parent):

def func2(self):

print("This function is in child 1.")

Derived class2

class Child2(Parent):

def func3(self):

print("This function is in child 2.")

Driver's code

object1 = Child1()

object2 = Child2()

object1.func1()

object1.func2()

object2.func1()

object2.func3()

Output:

This function is in parent class.

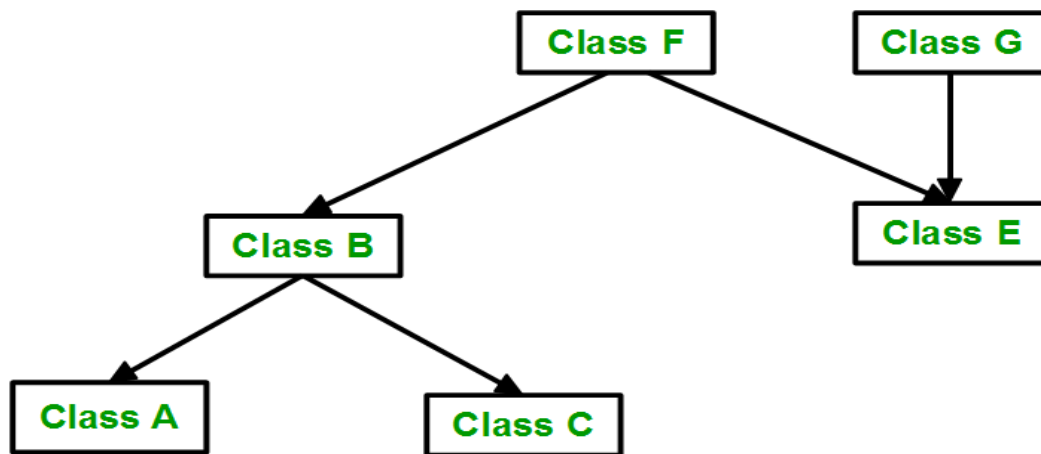
This function is in child 1.

This function is in parent class.

This function is in child 2.

5.Hybrid Inheritance:

Inheritance consisting of multiple types of inheritance is called hybrid inheritance.



Example:

Python program to demonstrate

hybrid inheritance

```
class School:
```

```
    def func1(self):
```

```
        print("This function is in school.")
```

```
class Student1(School):
```

```
    def func2(self):
```

```
        print("This function is in student 1. ")
```

```
class Student2(School):
```

```
    def func3(self):
```

```
        print("This function is in student 2.")
```

```

class Student3(Student1, School):

    def func4(self):

        print("This function is in student 3.")

# Driver's code

object = Student3()

object.func1()

object.func2()

```

Output:

This function is in school.

This function is in student 1.

6. How to create, raise and handle user defined exceptions in Python?

Creating Exception Class

Our Exception class should Implement Exceptions to raise exceptions. In this exception class, we can either put pass or give an implementation. Let us define init function.

Code:

```

class JustException(Exception):

    def __init__(self, message):

        print(message)

```

In the example above, JustException is a user-defined Exception. This class implements the Exception class. We have defined the `__init__` function that takes the message of the String type as a parameter and prints the message. This JustException class can implement everything like a normal class in Python.

Raising User-defined Exception

When a certain condition is met, we use the raise keyword to throw an exception. To raise a User-defined Exception, we can do the following:

Code:

```

raise JustException("Raise an Exception")

```

The raise keyword raises the exception mentioned after it. In this case, JustException is raised.

Output:

Raise an Exception

Traceback (most recent call last):

File "main.py", line 5, in <module>

```
raise JustException("Raise an Exception")
```

__main__.JustException: Raise an Exception

Explanation:

As we can see, the message we passed to the `__init__` class is displayed in the exception and output.

Catching Exception

As a usual exception, this exception can be handled using the try-except block.

Code:

try:

```
raise JustException("Raise an Exception")
```

except Exception as e:

```
print("Exception Raised")
```

Output:

Raise an Exception

Exception Raised

Explanation:

In the try block, we are raising the exception with the message. This message is printed from the `__init__` function. In except block, we print the message rather than ending the program in an undesired state.

Example of Creating User-Defined Exception

Let us take an example of a program used to enroll students in a particular course. To decide which students can enroll, there is a cut-off and maximum score. If a student's percentage is in the acceptable range, they are enrolled, or else an exception is raised. In this case, no student enrolling should have a percentage less than the cut-off as well as their score cannot be higher than the maximum score.

Code:

```

# Base class

class PercentageError(Exception):

    pass

# Exception class for percentage > 100

class InvalidPercentageError(PercentageError):

    def __init__(self):

        super().__init__("Percentage is invalid")

# Exception class for percentage < 80

class LessPercentageError(PercentageError):

    def __init__(self):

        super().__init__("The Percentage is lesser than the Cut-off, Please try again!")

# class to check percentage range

class checkPercentage(PercentageError):

    def __init__(self, per):

        if per<80:

            raise LessPercentageError

        if per>100:

            raise InvalidPercentageError

        print("Congrats you're Enrolled")

# different cases and output

try:

    print("For Percenatge: 93")

    checkPercentage(93)

except Exception as e:

    print(e)

try:

    print("\nFor Percenatge: 102")

    checkPercentage(102)

```



```

except Exception as e:

    print(e)

try:

    print("\nFor Percenatge: 58")

    checkPercentage(58)

except Exception as e:

    print(e)

```

Output:

```

For Percentage: 93

Congrats you're Enrolled

For Percentage: 102

Percentage is invalid

For Percentage: 58

The percentage is lesser than the Cut-off, Please try again!

```

Explanation:

- PercentageError class is a base class that implements the Python Exception class. This class can be implemented by the other Exception classes. We use pass as syntactically some statement is required but we don't want to enter any command in it.
- InvalidPercentage Error Exception is raised if the percentage value is greater than 100. LessPercentageError Exception is raised if the percentage value is less than the cut-off range. Both these classes have Initializers that send respective messages to the super class.
- checkPercentage class has an Initializer (Constructor) function that takes the percentage argument as per. It raises exception incase if per is not in desired range i.e InvalidPercentageError if greater than 100 or LessPercentageError if less than 80. We use the try-except block to see the output in all the 3 cases.

7. How will you create a Package & import it? Explain it with an example program

We organize a large number of files in different folders and subfolders based on some criteria, so that we can find and manage them easily. In the same way, a package in

Python takes the concept of the modular approach to next logical level. As you know, a module can contain multiple objects, such as classes, functions, etc. A package can contain one or more relevant modules. Physically, a package is actually a folder containing one or more module files.

Let's create a package named mypackage, using the following steps:

- Create a new folder named D:\MyApp.
- Inside MyApp, create a subfolder with the name 'mypackage'.
- Create an empty `__init__.py` file in the mypackage folder.
- Using a Python-aware editor like IDLE, create modules `greet.py` and `functions.py` with the following code:

```
greet.py

def SayHello(name):
    print("Hello ", name)
```

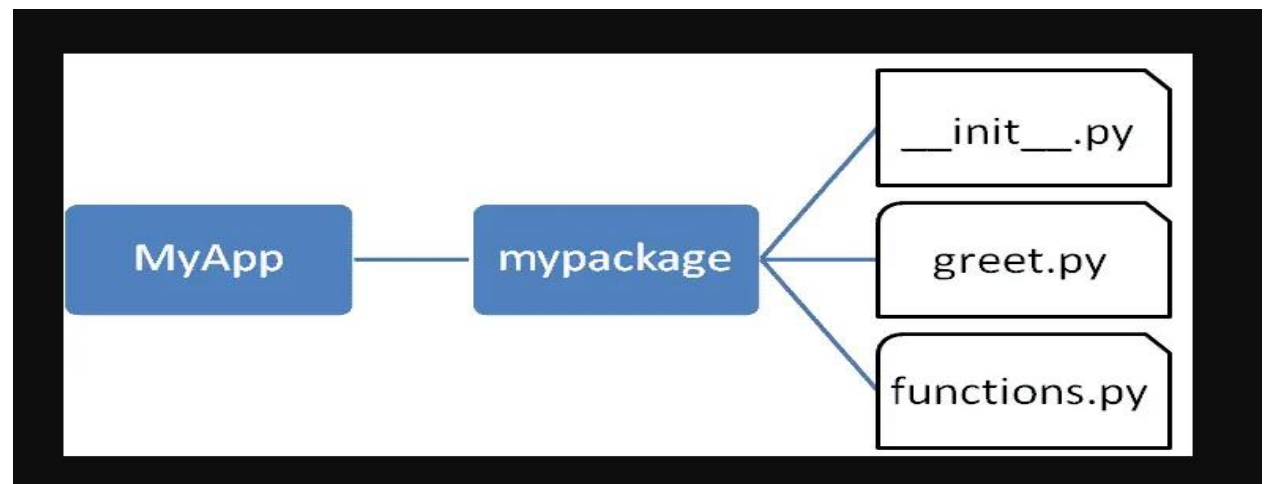
```
functions.py

def sum(x,y):
    return x+y

def average(x,y):
    return (x+y)/2

def power(x,y):
    return x**y
```

That's it. We have created our package called mypackage. The following is a folder structure:



Importing Module From a Package in Python

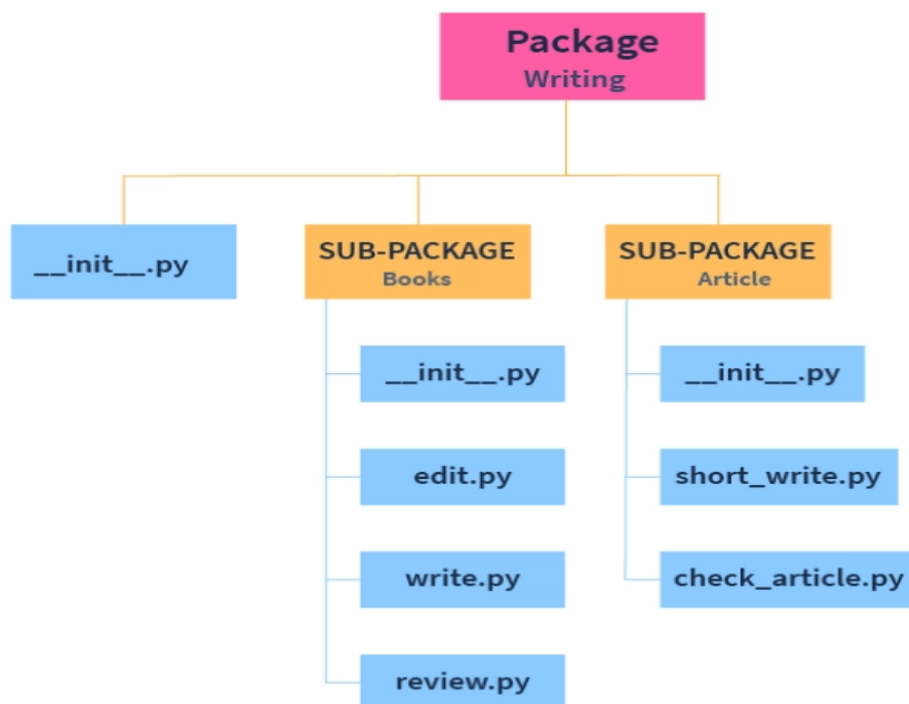
Packages help in ensuring the reusability of code. To access any module or file from a Python package, use an import statement in the Python source file where you want to access it. Import of modules from Python Packages is done using the dot operator (.). Importing modules and packages helps us make use of existing functions and code that can speed up our work.

Syntax:

```
import module1[, module2,... moduleN]
```

Where import is a keyword used to import the module, module1 is the module to be imported. The other modules enclosed in brackets are optional and can be mentioned only when more than 1 module is to be imported.

Consider the writing package given below.



To use the edit module from the writing package in a new file test.py that you have created, you write the following code in the test.py:

```
import Writing.Book.edit
```

To access a function called plagiarism_check() of the edit module, you use the following code:

```
Writing.Book.edit.plagiarism_check()
```

The calling method seems lengthy and confusing, right? Another way to import a module would be to simply import the package prefix instead of the whole package and call the function required directly.

```
from Writing.Book import edit
```

```
plagiarism_check()
```

However, the above method may cause problems when 2 or more packages have similarly named functions, and the call may be ambiguous. Thus, this method is avoided in most cases.

8. Write Program to print the number of lines, words and characters present in the given file.

The basic idea is to traverse each line in a file and count the number of words and characters.

[data.txt](#)

[main.py](#)

```
1 number_of_words = 0
2 number_of_lines = 0
3 number_of_characters = 0
4
5 with open("data.txt", 'r') as file:
6     for l in file:
7         number_of_words += len(l.split())
8         number_of_lines += 1
9         number_of_characters = len(l)
10
11 print("No of words: ", number_of_words)
12 print("No of lines: ", number_of_lines)
13 print("No of characters: ", number_of_characters)
```

Output

```
No of words: 704
No of lines: 14
No of characters: 998
```

In the above code snippet:

- **Lines 1–4:** We declare and initialize variables with value **0** to store the total count of words, characters, and lines.
- **Line 5:** We open the file in reading mode **r**.
- **Line 6:** We loop through each line in the file using the **for** loop.
- **Line 7:** We get the list of words present in a line using the **split()** method. We calculate the length/number of words passing the result of the **split()** method to the **len()** function and add it to the **number_of_words** variable.

- **Line 8:** We add value `1` to `number_of_lines` variable as we progress through each line in a file.
- **Line 9:** We get the count of characters using the `len()` function and add the result to the `number_of_characters` variable.

Once we traverse each line in a file, we get the count of lines, words, and characters present in a file.

9. Discuss the following methods associated with the file object with an example.

i) read () ii) read line() iii) read lines() iv) tell() v) seek() vi) write()

i) read() Method

Syntax:

```
file_object.read(size)
```

- The `size` represents the **number of bytes to read** from a file. It returns file content in a string object.
- If `size` is not specified, it **reads all content from a file**
- If the size argument is negative or not specified, read all data until EOF is reached.
- An empty string is returned when EOF is encountered immediately.
- When used in non-blocking mode, less data than requested may be returned, even if no size parameter was given.

Example:

```
# read(size)
with open(r'E:\pynative\files\test.txt', 'r') as fp:
    # read 14 bytes
    # fp is file object
    print(fp.read(14))

    # read all remaining content
    print(fp.read())
```

Output:

```
My First Line

My Second Line
My Third Line
```

ii) readline() Method

Syntax:

```
file_object.readline(size)
```

- Read one line from a file at a time. It returns the line in a string format.
- If the **size** is given it reads the number of bytes (including the trailing newline) from a file.
- If the size argument is negative or not specified, it read a single line
- An empty string is returned when EOF is encountered immediately.

Example:

```
# readline(size)
with open(r'E:\pynative\files\test.txt', 'r') as fp:
    # read single line
    print(fp.readline())

    # read next line
    print(fp.readline())
```

Output:

```
My First Line

My Second Line
```

iii) readlines() Method

Syntax:

```
file_object.readlines(size)
```

- Read all lines from a file and return them in the form of a [list](#) object.
- If the `sizehint` argument is present, instead of reading the entire file, whole lines totaling approximately `sizehint` bytes (possibly after rounding up to an internal buffer size) are read

Example:

```
# read(size)
with open(r'E:\pynative\files\test.txt', 'r') as fp:
    # read all lines
    print(fp.readlines())
```

Output:

```
['My First Line\n', 'My Second Line\n', 'My Third Line']
```

iv,v) seek() and tell() method

The `seek()` function sets the position of a file pointer and the `tell()` function returns the current position of a file pointer.

Example:

```
with open(r'E:\pynative\files\test.txt', "r") as fp:
    # Moving the file handle to 6th character
    fp.seek(6)
    # read file
    print(fp.read())

    # get current position of file pointer
    print(fp.tell())
```

vi) write() Method

Write a string to the file. If buffering is used, the line may not show up in the file until the `flush()` or `close()` method is called.

Example:

```
with open(r'E:\pynative\files\test.txt', 'w') as fp:  
    fp.write('My New Line')
```

10. What is module? Explain how you can use modules in your program. Explain with example?

Module is a file that contains code to perform a specific task. A module may contain variables, functions, classes etc. Let's see an example,

Let us create a module. Type the following and save it as example.py

```
# Python Module addition  
  
def add(a, b):  
  
    result = a + b  
    return result
```

Here, we have defined a function add() inside a module named example. The function takes in two numbers and returns their sum.

Import modules in Python

We can import the definitions inside a module to another module or the interactive interpreter in Python.

We use the import keyword to do this. To import our previously defined module example, we type the following in the Python prompt.

```
import example
```

This does not import the names of the functions defined in example directly in the current symbol table. It only imports the module name example there.

Using the module name we can access the function using the dot . operator. For example:

```
addition.add(4,5) # returns 9
```

from...import statement

We can import specific names from a module without importing the module as a whole. For example,

```
# import only pi from math module
```



```
from math import pi

print(pi)

# Output: 3.141592653589793
```

Here, we imported only the pi attribute from the math module.

Import all names

In Python, we can import all names(definitions) from a module using the following construct:

```
# import all names from the standard module math
from math import *

print("The value of pi is", pi)
```

Here, we have imported all the definitions from the math module. This includes all names visible in our scope except those beginning with an underscore(private definitions).

Importing everything with the asterisk (*) symbol is not a good programming practice. This can lead to duplicate definitions for an identifier. It also hampers the readability of our code.

11. Write a python program to implement packages like numpy / matplotlib.

12. How to create a text file and write content in to the text file. Explain with a python script.

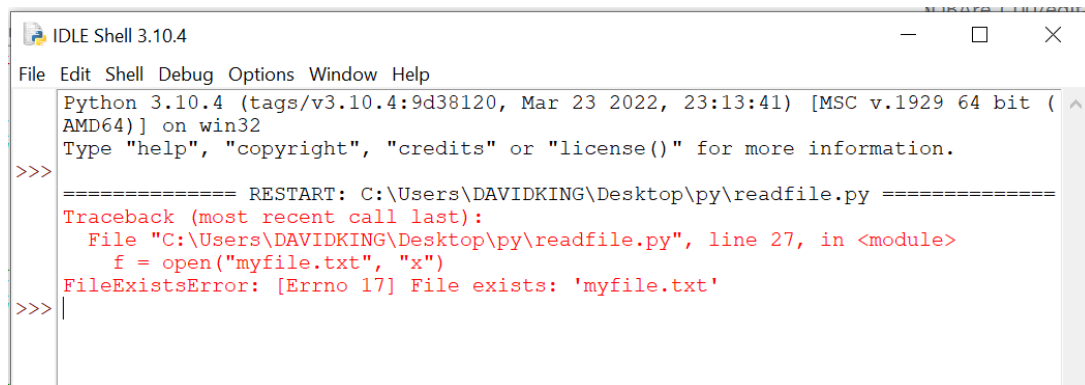
Create Files

In Python, you use the open() function with one of the following options – "x" or "w" – to create a new file:

- **"x" – Create:** this command will create a new file if and only if there is no file already in existence with that name or else it will return an error. Example of creating a file in Python using the "x" command:

```
#creating a text file with the command function "x"
f = open("myfile.txt", "x")
```

We've now created a new empty text file! But if you retry the code above – for example, if you try to create a new file with the same name as you used above (if you want to reuse the filename above) you will get an error notifying you that the file already exists. It'll look like the image below:



```
IDLE Shell 3.10.4
File Edit Shell Debug Options Window Help
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\DAVIDKING\Desktop\py\readfile.py =====
Traceback (most recent call last):
  File "C:\Users\DAVIDKING\Desktop\py\readfile.py", line 27, in <module>
    f = open("myfile.txt", "x")
FileExistsError: [Errno 17] File exists: 'myfile.txt'
>>>
```

- **"w" – Write:** this command will create a new text file whether or not there is a file in the memory with the new specified name. It does not return an error if it finds an existing file with the same name – instead it will overwrite the existing file. Example of how to create a file with the "w" command:

```
#creating a text file with the command function "w"
f = open("myfile.txt", "w")
```

This "w" command can also be used create a new file but unlike the the "x" command the "w" command will overwrite any existing file found with the same file name.

With the code above, whether the file exists or the file doesn't exist in the memory, you can still go ahead and use that code. Just keep in mind that it will overwrite the file if it finds an existing file with the same name.

To Write to a File

There are two methods of writing to a file in Python, which are:

The write() method:

This function inserts the string into the text file on a single line.

Based on the file we have created above, the below line of code will insert the string into the created text file, which is "myfile.txt."

```
file.write("Hello There\n")
```

The writelines() method:

This function inserts multiple strings at the same time. A list of string elements is created, and each string is then added to the text file.

Using the previously created file above, the below line of code will insert the string into the created text file, which is "myfile.txt."

```
f.writelines(["Hello World ", "You are welcome to Fcc\n"])
```

Example:

```
#This program shows how to write data in a text file.
```

```
file = open("myfile.txt","w")
```

```
L = ["This is Lagos \n","This is Python \n","This is Fcc \n"]
```

```
# i assigned ["This is Lagos \n","This is Python \n","This is Fcc \n"] to #variable L, you can use any letter or word of your choice.
```

```
# Variable are containers in which values can be stored.
```

```
# The \n is placed to indicate the end of the line.
```

```
file.write("Hello There \n")
```

```
file.writelines(L)
```

```
file.close()
```

```
# Use the close() to change file access modes
```

13. Describe the different access modes of the files with an example?

File Access Modes

Access modes govern the type of operations possible in the opened file. It refers to how the file will be used once its opened. These modes also define the location of the File Handle in the file. File handle is like a cursor, which defines from where the data has to be read or written in the file. There are 6 access modes in python.

Read Only ('r') : Open text file for reading. The handle is positioned at the beginning of the file. If the file does not exists, raises the I/O error. This is also the default mode in which a file is opened.

Read and Write ('r+') : Open the file for reading and writing. The handle is positioned at the beginning of the file. Raises I/O error if the file does not exist.

Write Only ('w') : Open the file for writing. For the existing files, the data is truncated and over-written. The handle is positioned at the beginning of the file. Creates the file if the file does not exist.

Write and Read ('w+') : Open the file for reading and writing. For an existing file, data is truncated and over-written. The handle is positioned at the beginning of the file.

Append Only ('a'): Open the file for writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.

Append and Read ('a+') : Open the file for reading and writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.

Here's few simple examples of how to open a file in different modes,

```
file1 = open("test.txt")      # equivalent to 'r' or 'rt'
file1 = open("test.txt", 'w') # write in text mode
file1 = open("img.bmp", 'r+b') # read and write in binary mode
```

Reading Files in Python

After we open a file, we use the `read()` method to read its contents. For example,

```
# open a file
file1 = open("test.txt", "r")

# read the file
read_content = file1.read()
print(read_content)
```

Output

```
This is a test file.
Hello from the test file.
```

In the above example, we have read the `test.txt` file that is available in our current directory. Notice the code,

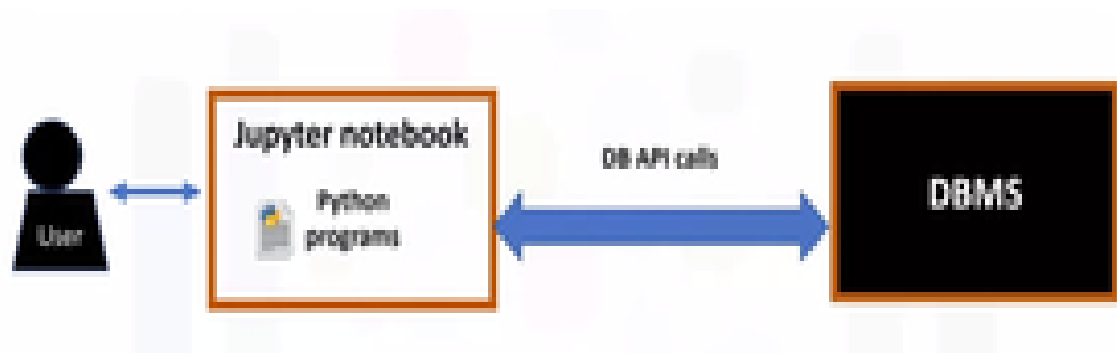
```
read_content = file1.read
```

Here, `file1.read()` reads the `test.txt` file and is stored in the `read_content` variable.

14. Explain DB-API based python database programming with example

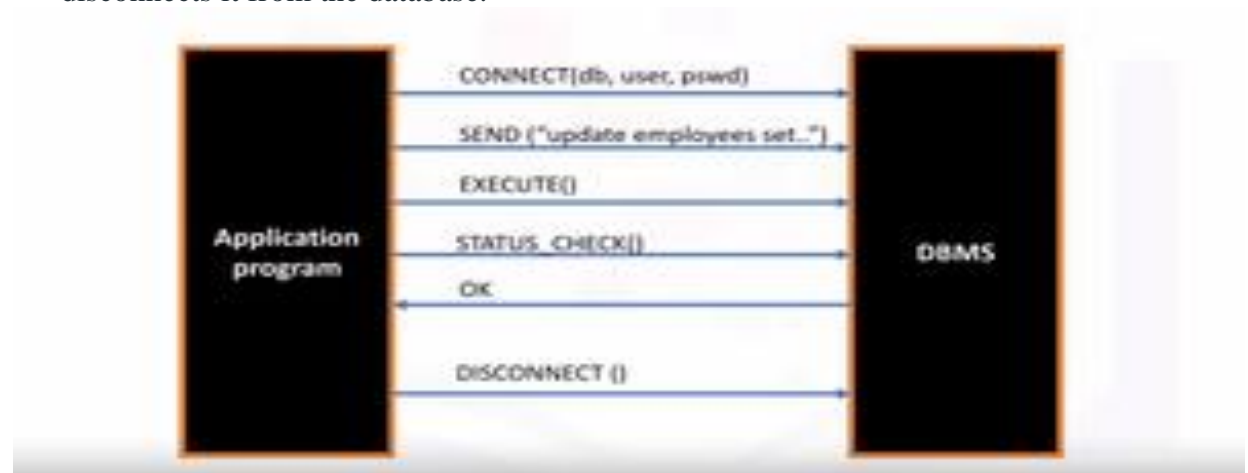
Databases are powerful tools for data scientists. **DB-API** is Python's standard API used for accessing databases. It allows you to write a single program

that works with multiple kinds of relational databases instead of writing a separate program for each one. This is how a typical user accesses databases using Python code written on a Jupyter notebook, a Web-based editor.



There is a mechanism by which the Python program communicates with the DBMS:

- The application program begins its database access with one or more API calls that connect the program to the DBMS.
- Then to send the SQL statement to the DBMS, the program builds the statement as a text string and then makes an API call to pass the contents to the DBMS.
- The application program makes API calls to check the status of its DBMS request and to handle errors.
- The application program ends its database access with an API call that disconnects it from the database.



The two main concepts in the Python DB-API are:

1) Connection objects used for

- Connect to a database
- Manage your transactions.

Following are a few connection methods:

- **cursor()**: This method returns a new cursor object using the connection.
- **commit()**: This method is used to commit any pending transaction to the database.
- **rollback()**: This method causes the database to roll back to the start of any pending transaction.
- **close()**: This method is used to close a database connection.
- **2) Query objects are used to run queries.**

- This is a python application that uses the DB-API to query a database.

```
from dbmodule import connect

# Create connection object
connection = connect('databasename', 'username', 'pswd')

# Create a cursor object
cursor = connection.cursor()

# Run queries
cursor.execute('select * from mytable')
results = cursor.fetchall()

# Free resources
cursor.close()
connection.close()
```

1. First, we import the database module by using the connect API from that module. To open a connection to the database, you use the **connection function** and pass in the parameters that are the database name, username, and password. The **connect function** returns the connection object.
2. After this, we create a **cursor object** on the connection object. The cursor is used to run queries and get the results.
3. After running the queries using the cursor, we also use the cursor to **fetch the results of the query**.
4. Finally, when the system is done running the queries, it frees all resources by **closing the connection**. Remember that it is always important to close connections to avoid unused connections taking up resources.

15. Explain in detail about TKinter with python programming?

Python offers multiple options for developing GUI (Graphical User Interface). Out of all the GUI methods, tkinter is the most commonly used method. It is a standard Python interface to the Tk GUI toolkit shipped with Python. Python with tkinter is the fastest and easiest way to create the GUI applications. Creating a GUI using tkinter is an easy task.

To create a tkinter app:

1. Importing the module – tkinter
2. Create the main window (container)
3. Add any number of widgets to the main window
4. Apply the event Trigger on the widgets.

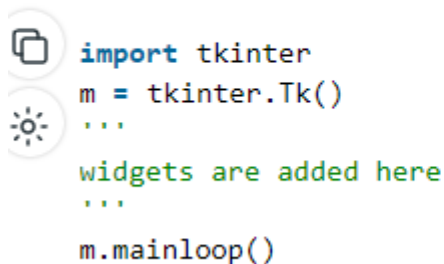
Importing tkinter is same as importing any other module in the Python code. Note that the name of the module in Python 2.x is 'Tkinter' and in Python 3.x it is 'tkinter'.

import tkinter

There are two main methods used which the user needs to remember while creating the Python application with GUI.

1. **Tk(screenName=None, baseName=None, className='Tk', useTk=1):**
To create a main window, tkinter offers a method 'Tk(screenName=None, baseName=None, className='Tk', useTk=1)'. To change the name of the window, you can change the className to the desired one. The basic code used to create the main window of the application is: **m=tkinter.Tk() where m is the name of the main window object**
2. **mainloop():** There is a method known by the name mainloop() is used when your application is ready to run. mainloop() is an infinite loop used to run the application, wait for an event to occur and process the event as long as the window is not closed.

```
m.mainloop()
```

A code snippet for Tkinter showing the import of the module, creation of a Tk object, adding widgets, and starting the main loop.

```
import tkinter
m = tkinter.Tk()
...
widgets are added here
...
m.mainloop()
```

tkinter also offers access to the geometric configuration of the widgets which can organize the widgets in the parent windows. There are mainly three geometry manager classes class.

1. **pack() method:**It organizes the widgets in blocks before placing in the parent widget.
2. **grid() method:**It organizes the widgets in grid (table-like structure) before placing in the parent widget.
3. **place() method:**It organizes the widgets by placing them on specific positions directed by the programmer.

Common Tkinter Widgets

Here are some of the commonly used Tkinter widgets:

1. **Button:** A simple button that can be clicked to perform an action.
2. **Label:** This is used to display simple text.
3. **Entry:** Simple input field to take input from the user. This is used for single-line input.
4. **Text:** This can be used for multi-line text fields, to show text, or to take input from the user.
5. **Canvas:** This widget can be used for drawing shapes, images, and custom graphics.
6. **Frame:** This acts as a container widget that can be used to group other widgets together.
7. **Checkbutton:** A checkbox widget used for creating toggle options on or off.

8. **Radiobutton**: This is used to create Radio buttons.
9. **Listbox**: This is used to create a List widget used for displaying a list of values.
10. **Scrollbar**: This widget is used for scrolling content in other widgets like Text and Listbox.

Here's a simple example where we have used some of the widgets,

```
import tkinter as tk

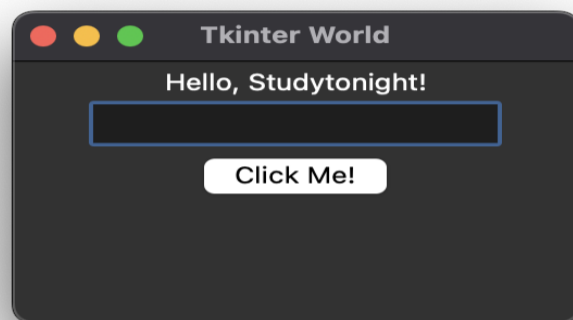
root = tk.Tk()
root.title("Tkinter World")

label = tk.Label(root, text="Hello, Studytonight!")
label.pack()

entry = tk.Entry(root)
entry.pack()

button = tk.Button(root, text="Click Me!")
button.pack()

root.mainloop()
```



11.

There are a number of widgets which you can put in your tkinter application. Some of the major widgets are explained below:

1. **Button**: To add a button in your application, this widget is used.

The general syntax is:

`w=Button(master, option=value)`

master is the parameter used to represent the parent window.

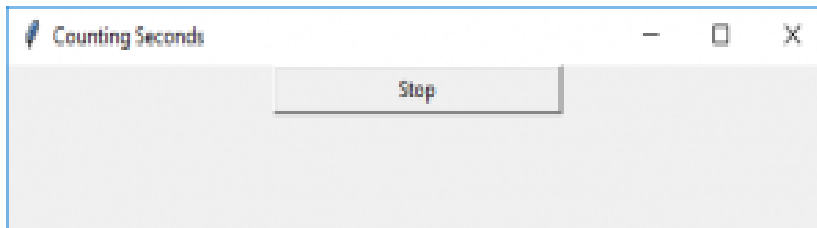
There are number of options which are used to change the format of the Buttons. Number of options can be passed as parameters separated by commas. Some of them are listed below.

- **activebackground**: to set the background color when button is under the cursor.

- **activeforeground:** to set the foreground color when button is under the cursor.
- **bg:** to set the normal background color.
- **command:** to call a function.
- **font:** to set the font on the button label.
- **image:** to set the image on the button.
- **width:** to set the width of the button.
- **height:** to set the height of the button.

```
import tkinter as tk
r = tk.Tk()
r.title('Counting Seconds')
button = tk.Button(r, text='Stop', width=25, command=r.destroy)
button.pack()
r.mainloop()
```

Output:



2.Canvas: It is used to draw pictures and other complex layout like graphics, text and widgets.

The general syntax is:

w = Canvas(master, option=value)

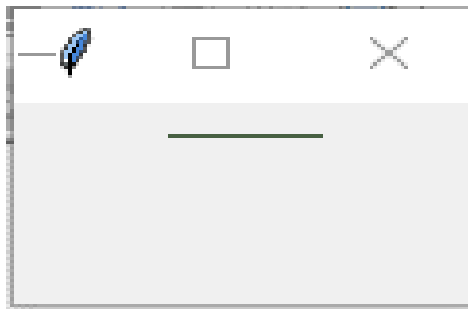
master is the parameter used to represent the parent window.

There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

- **bd:** to set the border width in pixels.
- **bg:** to set the normal background color.
- **cursor:** to set the cursor used in the canvas.
- **highlightcolor:** to set the color shown in the focus highlight.
- **width:** to set the width of the widget.
- **height:** to set the height of the widget.

```
from tkinter import *
master = Tk()
w = Canvas(master, width=40, height=60)
w.pack()
canvas_height=20
canvas_width=200
y = int(canvas_height / 2)
w.create_line(0, y, canvas_width, y )
mainloop()
```

Output:-



3.CheckButton: To select any number of options by displaying a number of options to a user as toggle buttons. The general syntax is:

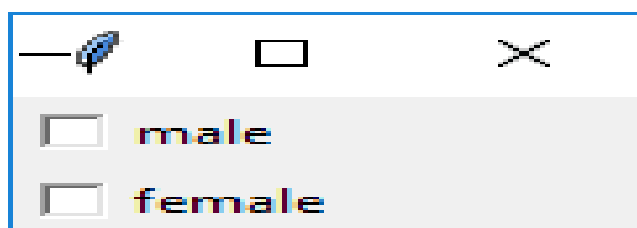
`w = CheckButton(master, option=value)`

There are number of options which are used to change the format of this widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

- **Title:** To set the title of the widget.
- **activebackground:** to set the background color when widget is under the cursor.
- **activeforeground:** to set the foreground color when widget is under the cursor.
- **bg:** to set the normal background color.
- **command:** to call a function.
- **font:** to set the font on the button label.
- **image:** to set the image on the widget.

```
from tkinter import *
master = Tk()
var1 = IntVar()
Checkbutton(master, text='male', variable=var1).grid(row=0, sticky=W)
var2 = IntVar()
Checkbutton(master, text='female', variable=var2).grid(row=1, sticky=W)
mainloop()
```

Output:-



4.Entry: It is used to input the single line text entry from the user.. For multi-line text input, Text widget is used.

The general syntax is:

```
w=Entry(master, option=value)
```

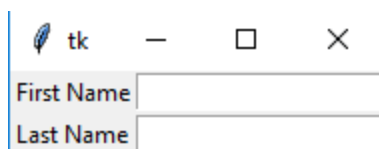
master is the parameter used to represent the parent window.

There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

- **bd:** to set the border width in pixels.
- **bg:** to set the normal background color.
- **cursor:** to set the cursor used.
- **command:** to call a function.
- **highlightcolor:** to set the color shown in the focus highlight.
- **width:** to set the width of the button.
- **height:** to set the height of the button.

```
from tkinter import *
master = Tk()
Label(master, text='First Name').grid(row=0)
Label(master, text='Last Name').grid(row=1)
e1 = Entry(master)
e2 = Entry(master)
e1.grid(row=0, column=1)
e2.grid(row=1, column=1)
mainloop()
```

Output:-



5.Frame: It acts as a container to hold the widgets. It is used for grouping and organizing the widgets. The general syntax is:

```
w = Frame(master, option=value)
```

master is the parameter used to represent the parent window.

There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

- **highlightcolor:** To set the color of the focus highlight when widget has to be focused.
- **bd:** to set the border width in pixels.
- **bg:** to set the normal background color.
- **cursor:** to set the cursor used.
- **width:** to set the width of the widget.

- **height:** to set the height of the widget.

```
from tkinter import *

root = Tk()
frame = Frame(root)
frame.pack()
bottomframe = Frame(root)
bottomframe.pack( side = BOTTOM )
redbutton = Button(frame, text = 'Red', fg = 'red')
redbutton.pack( side = LEFT)
greenbutton = Button(frame, text = 'Brown', fg='brown')
greenbutton.pack( side = LEFT )
bluebutton = Button(frame, text = 'Blue', fg = 'blue')
bluebutton.pack( side = LEFT )
blackbutton = Button(bottomframe, text = 'Black', fg = 'black')
blackbutton.pack( side = BOTTOM)
root.mainloop()
```

Output:-



6.Label: It refers to the display box where you can put any text or image which can be updated any time as per the code.

The general syntax is:

w=Label(master, option=value)

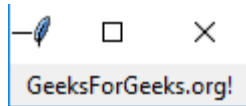
master is the parameter used to represent the parent window.

There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

- **bg:** to set the normal background color.
- **bg** to set the normal background color.
- **command:** to call a function.
- **font:** to set the font on the button label.
- **image:** to set the image on the button.
- **width:** to set the width of the button.
- **height**” to set the height of the button.

```
from tkinter import *
root = Tk()
w = Label(root, text='GeeksForGeeks.org!')
w.pack()
root.mainloop()
```

Output:-



7.Listbox: It offers a list to the user from which the user can accept any number of options.

The general syntax is:

```
w = Listbox(master, option=value)
```

master is the parameter used to represent the parent window.

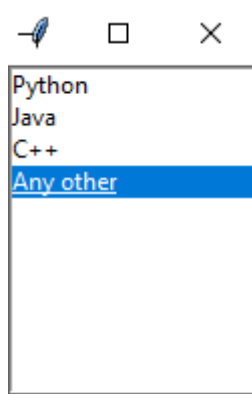
There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

- **highlightcolor:** To set the color of the focus highlight when widget has to be focused.
- **bg:** to set the normal background color.
- **bd:** to set the border width in pixels.
- **font:** to set the font on the button label.
- **image:** to set the image on the widget.
- **width:** to set the width of the widget.
- **height:** to set the height of the widget.

```
from tkinter import *
```

```
top = Tk()
Lb = Listbox(top)
Lb.insert(1, 'Python')
Lb.insert(2, 'Java')
Lb.insert(3, 'C++')
Lb.insert(4, 'Any other')
Lb.pack()
top.mainloop()
```

Output:-



8.MenuButton: It is a part of top-down menu which stays on the window all the time. Every menubutton has its own functionality. The general syntax is:
`w = MenuButton(master, option=value)`

master is the parameter used to represent the parent window.

There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

- **activebackground:** To set the background when mouse is over the widget.
- **activeforeground:** To set the foreground when mouse is over the widget.
- **bg:** to set the normal background color.
- **bd:** to set the size of border around the indicator.
- **cursor:** To appear the cursor when the mouse over the menubutton.
- **image:** to set the image on the widget.
- **width:** to set the width of the widget.
- **height:** to set the height of the widget.
- **highlightcolor:** To set the color of the focus highlight when widget has to be focused.

```
from tkinter import *
```

```
top = Tk()
mb = Menubutton ( top, text = "GfG")
mb.grid()
mb.menu = Menu ( mb, tearoff = 0 )
mb["menu"] = mb.menu
cVar = IntVar()
aVar = IntVar()
mb.menu.add_checkbutton ( label = 'Contact', variable = cVar )
mb.menu.add_checkbutton ( label = 'About', variable = aVar )
mb.pack()
top.mainloop()
```

Output:-



9.Menu: It is used to create all kinds of menus used by the application.

The general syntax is:

`w = Menu(master, option=value)`

master is the parameter used to represent the parent window.

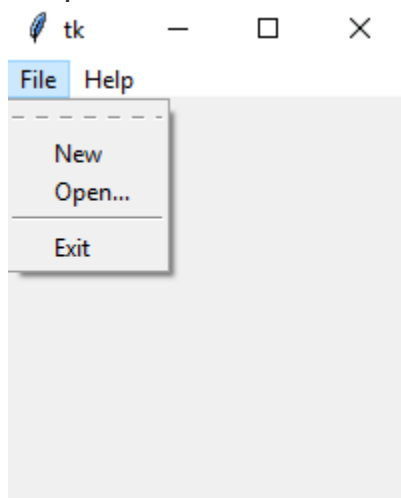
There are number of options which are used to change the format of this widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

- **title:** To set the title of the widget.
- **activebackground:** to set the background color when widget is under the cursor.
- **activeforeground:** to set the foreground color when widget is under the cursor.
- **bg:** to set the normal background color.
- **command:** to call a function.
- **font:** to set the font on the button label.
- **image:** to set the image on the widget.

```
from tkinter import *

root = Tk()
menu = Menu(root)
root.config(menu=menu)
filemenu = Menu(menu)
menu.add_cascade(label='File', menu=filemenu)
filemenu.add_command(label='New')
filemenu.add_command(label='Open...')
filemenu.add_separator()
filemenu.add_command(label='Exit', command=root.quit)
helpmenu = Menu(menu)
menu.add_cascade(label='Help', menu=helpmenu)
helpmenu.add_command(label='About')
mainloop()
```

Output:-



10.Message: It refers to the multi-line and non-editable text. It works same as that of Label.

The general syntax is:

`w = Message(master, option=value)`

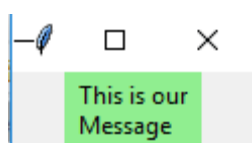
master is the parameter used to represent the parent window.

There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

- **bd:** to set the border around the indicator.
- **bg:** to set the normal background color.
- **font:** to set the font on the button label.
- **image:** to set the image on the widget.
- **width:** to set the width of the widget.
- **height:** to set the height of the widget.

```
from tkinter import *
main = Tk()
ourMessage = 'This is our Message'
messageVar = Message(main, text = ourMessage)
messageVar.config(bg='lightgreen')
messageVar.pack( )
main.mainloop( )
```

Output:-



11.RadioButton: It is used to offer multi-choice option to the user. It offers several options to the user and the user has to choose one option.

The general syntax is:

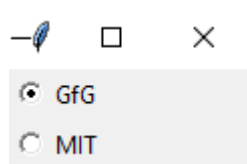
`w = RadioButton(master, option=value)`

There are number of options which are used to change the format of this widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

- **activebackground:** to set the background color when widget is under the cursor.
- **activeforeground:** to set the foreground color when widget is under the cursor.
- **bg:** to set the normal background color.
- **command:** to call a function.
- **font:** to set the font on the button label.
- **image:** to set the image on the widget.
- **width:** to set the width of the label in characters.
- **height:** to set the height of the label in characters.

```
from tkinter import *
root = Tk()
v = IntVar()
Radiobutton(root, text='GfG', variable=v, value=1).pack(anchor=W)
Radiobutton(root, text='MIT', variable=v, value=2).pack(anchor=W)
mainloop()
```

Output:-



12.Scale: It is used to provide a graphical slider that allows to select any value from that scale. The general syntax is:

`w = Scale(master, option=value)`

master is the parameter used to represent the parent window.

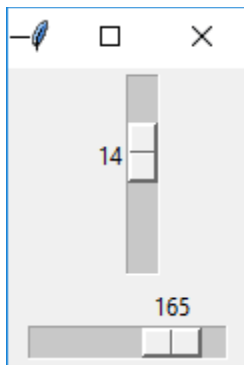
There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

- **cursor:** To change the cursor pattern when the mouse is over the widget.
- **activebackground:** To set the background of the widget when mouse is over the widget.
- **bg:** to set the normal background color.
- **orient:** Set it to HORIZONTAL or VERTICAL according to the requirement.

- **from_:** To set the value of one end of the scale range.
- **to:** To set the value of the other end of the scale range.
- **image:** to set the image on the widget.
- **width:** to set the width of the widget.

```
from tkinter import *
master = Tk()
w = Scale(master, from_=0, to=42)
w.pack()
w = Scale(master, from_=0, to=200, orient=HORIZONTAL)
w.pack()
mainloop()
```

Output:-



13.Scrollbar: It refers to the slide controller which will be used to implement listed widgets.

The general syntax is:

`w = Scrollbar(master, option=value)`

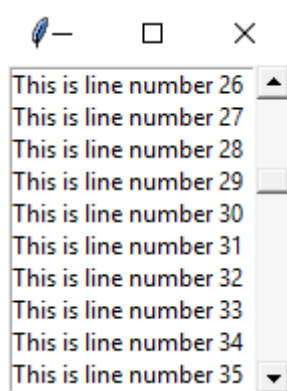
master is the parameter used to represent the parent window.

There are number of options which are used to change the format of the widget. Number of options can be passed as parameters separated by commas. Some of them are listed below.

- **width:** to set the width of the widget.
- **activebackground:** To set the background when mouse is over the widget.
- **bg:** to set the normal background color.
- **bd:** to set the size of border around the indicator.
- **cursor:** To appear the cursor when the mouse over the menubutton.

```
from tkinter import *
root = Tk()
scrollbar = Scrollbar(root)
scrollbar.pack( side = RIGHT, fill = Y )
mylist = Listbox(root, yscrollcommand = scrollbar.set )
for line in range(100):
    mylist.insert(END, 'This is line number' + str(line))
mylist.pack( side = LEFT, fill = BOTH )
scrollbar.config( command = mylist.yview )
mainloop()
```

Output:-



14.Text: To edit a multi-line text and format the way it has to be displayed.

The general syntax is:

`w =Text(master, option=value)`

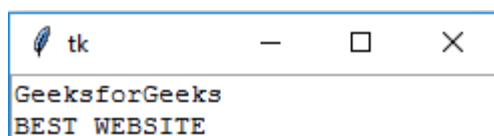
There are number of options which are used to change the format of the text.

Number of options can be passed as parameters separated by commas. Some of them are listed below.

- **highlightcolor:** To set the color of the focus highlight when widget has to be focused.
- **insertbackground:** To set the background of the widget.
- **bg:** to set the normal background color.
- **font:** to set the font on the button label.
- **image:** to set the image on the widget.
- **width:** to set the width of the widget.
- **height:** to set the height of the widget.

```
from tkinter import *
root = Tk()
T = Text(root, height=2, width=30)
T.pack()
T.insert(END, 'GeeksforGeeks\nBEST WEBSITE\n')
mainloop()
```

Output:-



16. Explain in detail about Object Relational Managers?

An object-relational mapper (ORM) is a code library that automates the transfer of data stored in relational database tables into objects that are more commonly used in application code.

ORMs provide a high-level abstraction upon a relational database that allows a developer to write Python code instead of SQL to create, read, update and delete data and schemas in their database. Developers can use the programming language they are comfortable with to work with a database instead of writing SQL statements or stored procedures.

The Need for ORM:

- In most software applications, data is represented as objects in the application code, but it needs to be stored and retrieved from relational databases.
- Mapping objects to database tables and vice versa can be complex and error-prone, especially for large and complex data models.

Advantages of ORM:

- **Abstraction:** Developers can work with objects in their code instead of writing complex SQL queries.
- **Portability:** ORM frameworks are typically database-agnostic, meaning you can switch databases without changing your application code.
- **Productivity:** ORM reduces the amount of boilerplate code needed for database interactions, making development faster.
- **Maintainability:** Code is more readable and maintainable because it closely resembles the application's object model.

Basic ORM Workflow:

1. **Define Entities:** Create classes to represent the entities in your application.
2. **Mapping:** Configure how these classes and their attributes map to the database tables and columns.
3. **CRUD Operations:** Use methods provided by the ORM to Create, Read, Update, and Delete records in the database.
4. **Querying:** Perform database queries using the ORM's query language or methods, which are often more object-oriented than raw SQL.

5. Transactions: Ensure data integrity by using transactions for complex operations involving multiple database changes.

17. What is tkinter TK()

Tkinter is Python's default GUI library. It is based on the Tk toolkit, originally designed for the Tool Command Language (Tcl). Due to Tk's popularity, it has been ported to a variety of other scripting languages, including Perl (Perl/Tk), Ruby

(Ruby/Tk), and Python (Tkinter). With the GUI development portability and flexibility of Tk, along with the simplicity of scripting language integrated with the power of systems language, you are given the tools to rapidly design and implement a wide variety of commercial-quality GUI applications.

In Tkinter, the `tk()` function is used to create the main application window or the root window for a graphical user interface (GUI) application. Tkinter is a standard Python library for creating GUI applications, and the `tk()` function is a part of it.

simple example of how you can use the `tk()` function to create a basic Tkinter Widget:

```
import Tkinter top = Tkinter.Tk()
```

```
label = Tkinter.Label(top, text='Hello World!')
```

```
label.pack()
```

```
Tkinter.mainloop()
```

o/p:



Description: The provided Python program uses the Tkinter library to create a basic graphical user interface (GUI) application. where the variable `top` represents this window. Inside the window, a label widget is created using `Tkinter.Label()`, with the text "Hello World!" assigned to it. The `pack()` method is then used to position and display the label within the main window. Finally, the program enters the Tkinter event loop with `Tkinter.mainloop()`, which keeps the GUI responsive,

allowing users to interact with it. When executed, this program results in a graphical window containing the text "Hello World!"

18. What is the need of Tkinter module in Python?

The Tkinter module in Python is used for creating graphical user interfaces (GUIs) for desktop applications. Here are several reasons why you might need Tkinter in your Python projects:

- **User-Friendly Interfaces:** Tkinter provides a simple and easy-to-use way to create graphical interfaces for your Python applications. It allows you to build windows, dialogs, buttons, text entry fields, labels, and other GUI elements, making your software more user-friendly.
- **Cross-Platform Compatibility:** Tkinter is included with standard Python distributions on most platforms (Windows, macOS, and Linux), which means you can create cross-platform applications without worrying about additional installations.
- **Wide Adoption:** Tkinter is one of the most widely used GUI libraries for Python, which means there's a large community and ample resources available for learning and troubleshooting.
- **Integration with Python:** Tkinter seamlessly integrates with Python code, allowing you to create interactive applications where user actions trigger Python functions.
- **Customization:** You can customize the appearance of your GUI elements using Tkinter to match the design and branding of your application.
- **Event-Driven Programming:** Tkinter is event-driven, meaning your program can respond to user events such as button clicks or keyboard input, making it well-suited for interactive applications.
- **Rapid Prototyping:** Tkinter is particularly useful for rapidly prototyping and building small to medium-sized desktop applications, utilities, or tools.
- **Open Source:** Tkinter is open-source and actively maintained, which means it's continually improved and updated.

Example

program:

```
import
```

```
tkinter
```

```
as tk
```

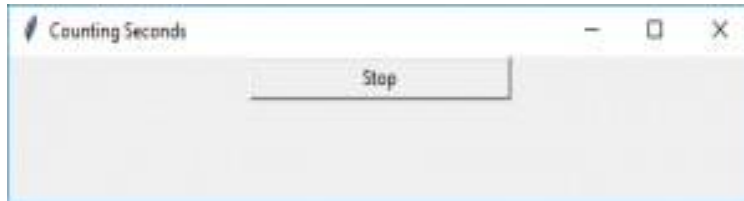
```
r = tk.Tk() r.title('Counting Seconds')
```

```
button = tk.Button(r, text='Stop', width=25, command=r.destroy)
```

```
button.pack()
```

```
r.mainloop()
```

o/p:



Description: This Python program utilizes the Tkinter library to create a basic graphical user interface (GUI) application. Upon execution, it generates a window with the title "Counting Seconds." Within this window, a button labeled "Stop" is displayed. When this button is clicked, it triggers the **r.destroy** function, which closes the application's main window and terminates the program. The **r.mainloop()** call initiates the Tkinter event loop, enabling user interaction and ensuring the GUI remains responsive.

19. Develop a Python program to manage your class database. (Name, DOB, Reg.No, Addr,Phone No, Mark)

```
class
```

```
    StudentDataba
```

```
    se:def_init_
```

```
    (self):
```

```
        self.students = []
```

```
    def add_student(self, name, dob, reg_number, address, phno,marks):
```

```
        student_info = { "Name": name,"DOB": dob,"Registration Number":
                           reg_number,"Address": address,"Phone Number": phno,"Marks":
                           marks}
```

```
        self.students.append(student_info)
```

```

def display_students(self):

    for i, student in enumerate(self.students, start=1):print(f"Student {i}:")

        for key, value in
            student.items():
                print(f"{key}: {value}")

        print()

if __name__ == "__main__"
    database =
        StudentDatabase()

    while True:
        print("Options:")
        print("1 - Add
        Student")

        print("2 - Display
        Students")print("3 -
        Exit")

        choice = input("Enter your choice: ")

        if choice == "1":

            name = input("Enter
            Name: ")dob =
            input("Enter DOB: ")

            reg_number = input("Enter Registration Number: ")
            address = input("Enter Address: ")

```



```

        phno = input("Enter Phone Number: ")
        marks = input("Enter Marks: ")

        database.add_student(name, dob, reg_number, address, phno, marks)

        print("Student added successfully!\n")

    elif choice == "2":

        database.display_students()

    elif choice == "3":

        break

    else:

        print("Invalid choice. Please try again.\n")

```

o/p:

Options:

1 - Add Student

2 - Display Students

3 - Exit

Enter your

choice: 1

Enter

Name: John

Doe Enter

DOB: 2000-

01-15

Enter Registration

Number: 12345 Enter

Address: 123 Main Street

Enter Phone Number:

555-123-4567Enter

Marks: 95

Student added successfully!

Options:

1 - Add Student

2 - Display Students

3 - Exit

Enter your choice: 3

Description:

- the **StudentDatabase** class is created to manage the database of students. It has methods for adding students and displaying the list of students.
- The program provides a menu-driven interface with options to add students, display the list of students, and exit the program.
- When you choose to add a student, you are prompted to enter their information, which is then added to the database.
- When you choose to display students, the program prints out the details of all the students in the database.
- You can exit the program by selecting the "Exit" option.