What is Function? Explain how to create a function with suitable example?(declaration,defining and calling a function)

A function is a block of code which only runs when it is called.
You can pass data, known as parameters, into a function.
A function can return data as a result.

Creating a Function

In Python a function is defined using the **def** keyword:
Ex:-

```
def my_function():
    print("You are working with a function ")
```

Calling a Function

To call a function, use the function name followed by parenthesis:
Ex:-

```
def my_function():
    print("Hello from a function")
```

```
my_function()
```

Output:-

```
You are working with a function
```

Parameters or Arguments:

The terms *parameter* and *argument* can be used for the same thing:information that are passed into a function.
A parameter is the variable listed inside the parentheses in the function definition.
An argument is the value that is sent to the function when it is called.
Ex:-

```
def my_function(fname, lname):
    print(fname + " " + lname)
```

```
my_function("Susmitha", "Sen")
```

Output:-

```
Susmitha Sen
```

Return Values:

To let a function return a value, use the return statement:
Ex:-

```
def my_function(x):
    return 5 * x
```

```
print(my_function(3))
print(my_function(5))
print(my_function(9))
```

Output:

```
15
25
45
```

Discuss Built-in functions in Python.

Function	Description
abs()	Returns the absolute value of a number
all()	Returns True if all items in an iterable object are true
any()	Returns True if any item in an iterable object is true
ascii()	Returns a readable version of an object. Replaces none-ascii characters with escape character
bin()	Returns the binary version of a number
bool()	Returns the boolean value of the specified object
bytearray()	Returns an array of bytes
bytes()	Returns a bytes object
callable()	Returns True if the specified object is callable, otherwise False
chr()	Returns a character from the specified Unicode code.
classmethod()	Converts a method into a class method
compile()	Returns the specified source as an object, ready to be executed
complex()	Returns a complex number
delattr()	Deletes the specified attribute (property or method) from the specified object
dict()	Returns a dictionary (Array)
dir()	Returns a list of the specified object's properties and methods
divmod()	Returns the quotient and the remainder when argument1 is divided by argument2
enumerate()	Takes a collection (e.g. a tuple) and returns it as an enumerate object
eval()	Evaluates and executes an expression
exec()	Executes the specified code (or object)
filter()	Use a filter function to exclude items in an iterable object
float()	Returns a floating point number
format()	Formats a specified value
frozenset()	Returns a frozenset object

getattr()	Returns the value of the specified attribute (property or method)
globals()	Returns the current global symbol table as a dictionary
hasattr()	Returns True if the specified object has the specified attribute (property/method)
hash()	Returns the hash value of a specified object
help()	Executes the built-in help system
hex()	Converts a number into a hexadecimal value
id()	Returns the id of an object
input()	Allowing user input
int()	Returns an integer number
isinstance()	Returns True if a specified object is an instance of a specified object
issubclass()	Returns True if a specified class is a subclass of a specified object
iter()	Returns an iterator object
len()	Returns the length of an object
list()	Returns a list
locals()	Returns an updated dictionary of the current local symbol table
map()	Returns the specified iterator with the specified function applied to each item
max()	Returns the largest item in an iterable
memoryview()	Returns a memory view object
min()	Returns the smallest item in an iterable
next()	Returns the next item in an iterable
object()	Returns a new object
oct()	Converts a number into an octal
open()	Opens a file and returns a file object
ord()	Convert an integer representing the Unicode of the specified character
pow()	Returns the value of x to the power of y
print()	Prints to the standard output device
property()	Gets, sets, deletes a property

range()	Returns a sequence of numbers, starting from 0 and increments by 1 (by default)
repr()	Returns a readable version of an object
reversed()	Returns a reversed iterator
round()	Rounds a numbers
set()	Returns a new set object
setattr()	Sets an attribute (property/method) of an object
slice()	Returns a slice object
sorted()	Returns a sorted list
staticmethod()	Converts a method into a static method
str()	Returns a string object
sum()	Sums the items of an iterator
super()	Returns an object that represents the parent class
tuple()	Returns a tuple
type()	Returns the type of an object
vars()	Returns the __dict__ property of an object
zip()	Returns an iterator, from two or more iterators

Python has a set of built-in functions.

abs():- Returns the absolute value of a number

```
x = abs(-7.25)
print(x)
```

output:-7.25

chr():-Returns a character from the specified Unicode code.

```
x = chr(97)
print(x)
```

Output:- a

input() :- Allowing user input

```
print("Enter your name:")
x = input()
print("Hello, " + x)
```

Output:-Enter your name:Smith
Hello Smith

len():-Returns the length of an object

```
mylist = ["apple", "orange", "cherry"]
x = len(mylist)
print(x)
```

pow() :- Returns the value of x to the power of y

```
x = pow(4, 3)
print(x)
```

Output:- 64

reversed() :-

```
alph = ["a", "b", "c", "d"]
ralph = reversed(alph)
for x in ralph:
        print(x)
```

Output:- d
c
b
a

sum():-Sums the items of an iterator

```
a = (1, 2, 3, 4, 5)
x = sum(a)
print(x)
```

Output:-15

sorted():-Returns a sorted list

```
a = ("b", "g", "a", "d", "f", "c", "h", "e")
x = sorted(a)
print(x)
```

Output:- ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']

slice():-Returns a slice object

```
a = ("a", "b", "c", "d", "e", "f", "g", "h")
x = slice(2)
print(a[x])
```

Output:-('a','b')

Explain varible scope in Python?

A variable is only available from inside the region it is created. This is called **scope**.

Local Scope:

A variable created inside a function belongs to the *local scope* of that function, and can only be used inside that function.

A variable created inside a function is available inside that function:

Ex:-

```
def myfunc():
        x = 300
        print(x)
```

```
myfunc()
```

Output:-

```
300
```

Global Scope:

A variable created in the main body of the Python code is a global variable and belongs to the global scope.
Global variables are available from within any scope, global and local.
A variable created outside of a function is global and can be used by anyone:

Ex:-

```
x = 300
def myfunc():
        print(x)
```

```
myfunc()

print(x)
```

Output:-

```
300
300
```

'Global' Keyword:

If you need to create a global variable, but are stuck in the local scope, you can use the global keyword.
The global keyword makes the variable global.
If you use the global keyword, the variable belongs to the global scope:

Ex:-

```
def myfunc():
        global x
        x = 300
```

```
myfunc()

print(x)
```

Output:- 300

Also, use the global keyword if you want to make a change to a global variable inside a function.
To change the value of a global variable inside a function, refer to the variable by using the global keyword:

Ex:-

```
x = 300
def myfunc():
        global x
        x = 200

myfunc()

print(x)
```

Output: 200

Explain function recursion in python.

Or

Write a python program to find a factorial for a given number.

Python also accepts function recursion, which means a defined function can call itself.

Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.

A complicated function can be split down into smaller sub-problems utilizing recursion.

Sequence creation is simpler through recursion than utilizing any nested iteration.

Recursive functions render the code look simple and effective.

Disadvantages of using recursion

A lot of memory and time is taken through recursive calls which makes it expensive for use.

Recursive functions are challenging to debug.

The reasoning behind recursion can sometimes be tough to think through.

Syntax:

```
def func(): <--
            |
            | (recursive call)
            |
    func() ----
```

Ex:

```
# Program to print factorial of a number
# recursively.
# Recursive function
def recursive_factorial(n):
  if n == 1:
      return n
  else:
      return n * recursive_factorial(n-1)
 # user input
num = 6
 # check if the input is valid or not
if num < 0:
        print("Invalid input ! Please enter a positive number.")
elif num == 0:
        print("Factorial of number 0 is 1")
else:
        print("Factorial of number", num, "=", recursive_factorial(num))
```

output: Factorial of number 6 = 720

Explain defining classes in Python?

Python is an object oriented programming language.

Almost everything in Python is an object, with its properties and methods.

A Class is like an object constructor, or a "blueprint" for creating objects.

Create Class:

To create a class, use the keyword class:

Create a class named MyClass, with a property named x:

```
class MyClass:
  x = 5

print(MyClass)
```

Output:<class '__main__.MyClass'>

Create Object:

Now we can use the class named MyClass to create objects:

Create an object named p1, and print the value of x:

```
class MyClass:
  x = 5

p1 = MyClass()
print(p1.x)
```

Output:5

The __init__() Function:

The examples above are classes and objects in their simplest form, and are not really useful in real life applications.

To understand the meaning of classes we have to understand the built-in __init__() function.

All classes have a function called __init__(), which is always executed when the class is being initiated.

Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created:

Create a class named Person, use the __init__() function to assign values for name and age:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)
print(p1.name)
print(p1.age)
```

Output:

John

36

Explain how to inherit classes in Python.

Python Inheritance:

Inheritance allows us to define a class that inherits all the methods and properties from another class.

Parent class is the class being inherited from, also called base class.

Child class is the class that inherits from another class, also called derived class.

Create a Parent Class:

Any class can be a parent class, so the syntax is the same as creating any other class:

Create a class named Person, with firstname and lastname properties, and a printname method:

Create a Child Class:

To create a class that inherits the functionality from another class, send the parent class as a parameter when creating the child class:

Create a class named **Student**, which will inherit the properties and methods from the **Person** class:

Use the pass keyword when you do not want to add any other properties or methods to the class.

Now the Student class has the same properties and methods as the Person class.

Use the Student class to create an object, and then execute the printname method:

Ex:

```
class Person:
        def __init__(self, fname, lname):
                self.firstname = fname
                self.lastname = lname

        def printname(self):
                print(self.firstname, self.lastname)

class Student(Person):
                pass

x = Student("Mike", "Olsen")
x.printname()
```

Output: Mike Olsen

<u>Explain the concept of Polymorphism in Python.</u>

Polymorphism means multiple forms. In python we can find the same operator or function taking multiple forms. It also useful in creating different classes which will have class methods with same name.

That helps in

- re using a lot of code
- decreases code complexity.

Polymorphism is also linked to inheritance as we will see in some examples below.

Polymorphism in operators:

The + operator can take two inputs and give us the result depending on what the inputs are. In the below examples we can see how the integer inputs yield an integer and if one of the input is float then the result becomes a float. Also for strings, they simply get concatenated. This happens automatically because of the way the + operator is created in python.

Ex:

```
a = 23

b = 11

c = 9.5

s1 = "Hello"

s2 = "There!"
```

```
print(a + b)
print(type(a + b))
print(b + c)
print(type (b + c))
print(s1 + s2)
print(type(s1 + s2))
```

Output:

34

20.5

HelloThere!

Polymorphism in in-built functions:

We can also see that different python functions can take inputs of different types and then process them differently. When we supply a string value to **len()** it counts every letter in it. But if we five tuple or a dictionary as an input, it processes them differently.

Ex:

```
str = 'Hi There !'
tup = ('Mon','Tue','wed','Thu','Fri')
lst = ['Jan','Feb','Mar','Apr']
dict = {'1D':'Line','2D':'Triangle','3D':'Sphere'}
print(len(str))
print(len(tup))
print(len(lst))
print(len(dict))
```

Output:

10
5
4
3

Explain the the python package DateTime.

Python Dates

A date in Python is not a data type of its own, but we can import a module named **`datetime`** to work with dates as date objects.

Import the datetime module and display the current date:

```
import datetime

x = datetime.datetime.now()
print(x)
```

output: 2022-08-23 21:39:17.747166

The strftime() Method

The `datetime` object has a method for formatting date objects into readable strings.
The method is called `strftime()`, and takes one parameter, `format`, to specify the format of the returned string:

#Return the year and name of weekday:

```
import datetime
x = datetime.datetime.now()
print(x.year)
print(x.strftime("%A"))
```

output:
2022
Tuesday
format codes:

Directive	Description	Example
%a	Weekday, short version	Wed
%A	Weekday, full version	Wednesday
%w	Weekday as a number 0-6, 0 is Sunday	3
%d	Day of month 01-31	31
%b	Month name, short version	Dec
%B	Month name, full version	December
%m	Month as a number 01-12	12
%y	Year, short version, without century	18
%Y	Year, full version	2018
%H	Hour 00-23	17
%I	Hour 00-12	05
%p	AM/PM	PM
%M	Minute 00-59	41
%S	Second 00-59	08
%f	Microsecond 000000-999999	548513
%z	UTC offset	+0100
%Z	Timezone	CST

%j	Day number of year 001-366	365
%U	Week number of year, Sunday as the first day of week, 00-53	52
%W	Week number of year, Monday as the first day of week, 00-53	52
%c	Local version of date and time	Mon Dec 31 17:41:00 2018
%C	Century	20
%x	Local version of date	12/31/18
%X	Local version of time	17:41:00
%%	A % character	%
%G	ISO 8601 year	2018
%u	ISO 8601 weekday (1-7)	1
%V	ISO 8601 weeknumber (01-53)	01

Explain Math module in Python.

Python has also a built-in module called math, which extends the list of mathematical functions.
To use it, you must import the math module:

```
import math
```

When you have imported the math module, you can start using methods and constants of the module.
The math.sqrt() method for example, returns the square root of a number:

```
import math
x = math.sqrt(64)

print(x)
```

output: 8.0
The math.ceil() method rounds a number upwards to its nearest integer, and the math.floor() method rounds a number downwards to its nearest integer, and returns the result:

```
import math

x = math.ceil(1.4)
y = math.floor(1.4)

print(x) # returns 2
print(y) # returns 1
```

output:
2
1
Python math Module
Python has a built-in module that you can use for mathematical tasks.

The **math** module has a set of methods and constants.

Math Methods

Method	Description
math.acos()	Returns the arc cosine of a number
math.acosh()	Returns the inverse hyperbolic cosine of a number
math.asin()	Returns the arc sine of a number
math.asinh()	Returns the inverse hyperbolic sine of a number
math.atan()	Returns the arc tangent of a number in radians
math.atan2()	Returns the arc tangent of y/x in radians
math.atanh()	Returns the inverse hyperbolic tangent of a number
math.ceil()	Rounds a number up to the nearest integer
math.comb()	Returns the number of ways to choose k items from n items without repetition and order
math.copysign()	Returns a float consisting of the value of the first parameter and the sign of the second parameter
math.cos()	Returns the cosine of a number
math.cosh()	Returns the hyperbolic cosine of a number
math.degrees()	Converts an angle from radians to degrees
math.dist()	Returns the Euclidean distance between two points (p and q), where p and q are the coordinates of that point
math.erf()	Returns the error function of a number
math.erfc()	Returns the complementary error function of a number
math.exp()	Returns E raised to the power of x
math.expm1()	Returns $E^x - 1$
math.fabs()	Returns the absolute value of a number
math.factorial()	Returns the factorial of a number
math.floor()	Rounds a number down to the nearest integer
math.fmod()	Returns the remainder of x/y
math.frexp()	Returns the mantissa and the exponent, of a specified number

math.fsum()	Returns the sum of all items in any iterable (tuples, arrays, lists,etc.)
math.gamma()	Returns the gamma function at x
math.gcd()	Returns the greatest common divisor of two integers
math.hypot()	Returns the Euclidean norm
math.isclose()	Checks whether two values are close to each other, or not
math.isfinite()	Checks whether a number is finite or not
math.isinf()	Checks whether a number is infinite or not
math.isnan()	Checks whether a value is NaN (not a number) or not
math.isqrt()	Rounds a square root number downwards to the nearest integer
math.ldexp()	Returns the inverse of math.frexp() which is x * (2**i) of the given numbers x and i
math.lgamma()	Returns the log gamma value of x
math.log()	Returns the natural logarithm of a number, or the logarithm of number to base
math.log10()	Returns the base-10 logarithm of x
math.log1p()	Returns the natural logarithm of 1+x
math.log2()	Returns the base-2 logarithm of x
math.perm()	Returns the number of ways to choose k items from n items with order and without repetition
math.pow()	Returns the value of x to the power of y
math.prod()	Returns the product of all the elements in an iterable
math.radians()	Converts a degree value into radians
math.remainder()	Returns the closest value that can make numerator completely divisible by the denominator
math.sin()	Returns the sine of a number
math.sinh()	Returns the hyperbolic sine of a number
math.sqrt()	Returns the square root of a number
math.tan()	Returns the tangent of a number
math.tanh()	Returns the hyperbolic tangent of a number

math.trunc()	Returns the truncated integer parts of a number

Explain Packages in Python?

A package contains all the files you need for a module.

Modules are Python code libraries we can include in our project.

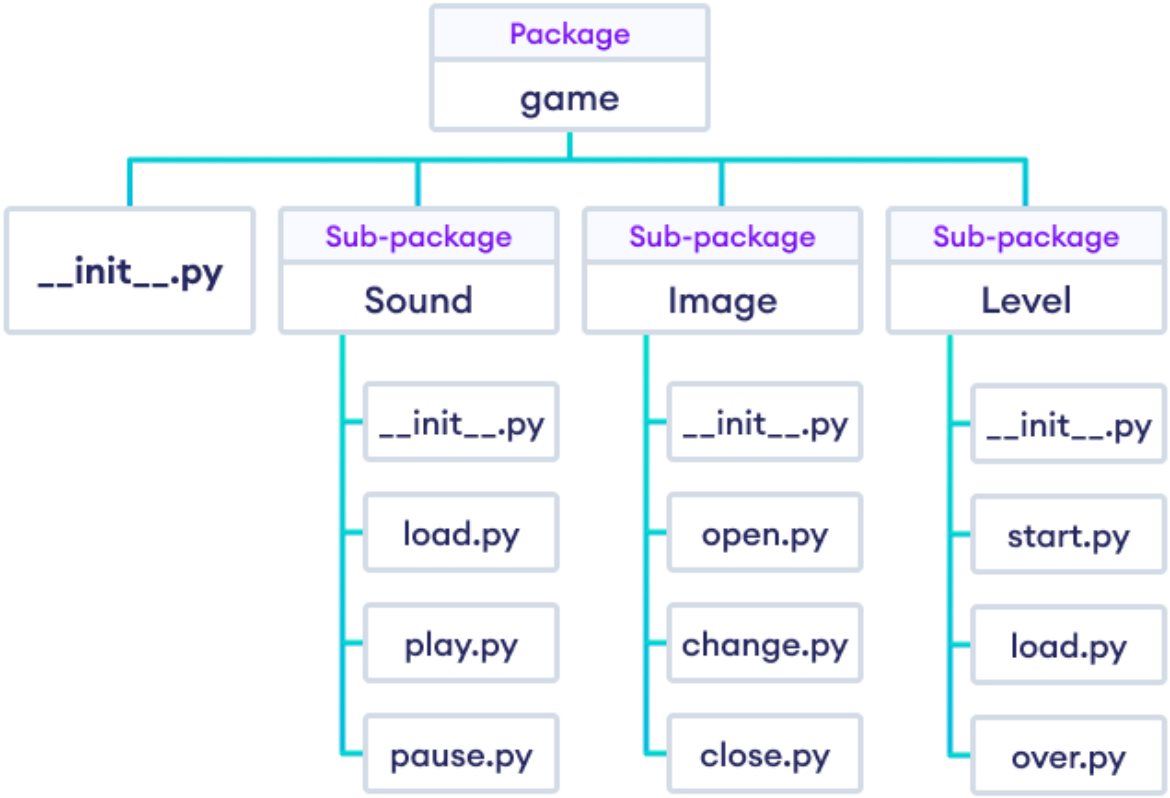
A package is a container that contains various functions to perform specific tasks. For example, the math package includes the sqrt() function to perform the square root of a number.

While working on big projects, we have to deal with a large amount of code, and writing everything together in the same file will make our code look messy. Instead, we can separate our code into multiple files by keeping the related code together in packages.

Now, we can use the package whenever we need it in our projects. This way we can also reuse our code.

Package Model Structure in Python Programming

Suppose we are developing a game. One possible organization of packages and modules could be as shown in the figure below.



Importing module from a package

In Python, we can import modules from packages using the dot (.) operator.

For example, if we want to import the start module in the above example, it can be done as follows:

```
import Game.Level.start
```

Now, if this module contains a function named select_difficulty(), we must use the full name to reference it.

```
Game.Level.start.select_difficulty(2)
```

Import Without Package Prefix

If this construct seems lengthy, we can import the module without the package prefix as follows:

```
from Game.Level import start
```

Ex:

```
import camelcase
c = camelcase.CamelCase()
txt = "lorem ipsum dolor sit amet"
```

```
print(c.hump(txt))

#This method capitalizes the first letter of each word.
```

Output:
Lorem Ipsum Dolor Sit Amet

The top Python libraries: NumPy, Pandas, Matplotlib, TensorFlow, PyTorch, Scikit-learn, Requests, Keras, Seaborn, Plotly, NLTK, Beautiful Soup, Pygame, Gensim, spaCy, SciPy, Theano, PyBrain, Bokeh, and Hebel.

Discuss Exceptions in Python.

Exception Handling:

When an error occurs, or exception as we call it, Python will normally stop and generate an error message.

- The **`try`** block lets you test a block of code for errors.
- The **`except`** block lets you handle the error.
- The **`else`** block lets you execute code when there is no error.
- The **`finally`** block lets you execute code, regardless of the result of the try- and except blocks.

Try block:

These exceptions can be handled using the try statement:

Ex:

The `try` block will generate an exception, because `x` is not defined:

```
#The try block will generate an error, because x is not defined:

try:

  print(x)

except:

  print("An exception occurred")
```

Output:

An exception occurred

Since the try block raises an error, the **except block** will be executed.

Without the try block, the program will crash and raise an error:

#This will raise an exception, because x is not defined:

```
print(x)
```

output:

NameError: name 'x' is not defined

[[[***Some Syntax Errors***

Ex1:

```
name = "Ankit
print(name)
```

SyntaxError: EOL while scanning string literal

Ex2;

```
print(7+3
```

SyntaxError: unexpected EOF while parsing

Ex3:

```
 a = 7
b = 0
print(a/b)
```

ZeroDivisionError: division by zero

Ex4 :

```
r = [23,25,27,29,31]
print(r[5])
```

]]]

1) ZeroDivisionError: division by zero

EX:

```
try:
   a = int(input('Enter 1st number :'))
   b = int(input('Enter 2nd number :'))
   print(a/b)
except ZeroDivisionError as em:  # em = err msg
   print('Error msg: ',em)
```

output:

Enter 1st number :6
Enter 2nd number :0
Error msg: division by zero

2) TypeError

```
try:
   a = int(input('Enter 1st no: '))
   b = int(input('Enter 2nd no: '))
except ValueError as em:
   print('Error Msg -',em)
else:
   print(f'Product of {a} and {b} is  {a*b}')
finally:
   print("This block will work regradless of occurrence of error")
```

output:

Enter 1st no: 13
Enter 2nd no: 9
Sum of 13 and 9 is 22
This block will work regradless of occurence of error

Many Exceptions

You can define as many exception blocks as you want, e.g. if you want to execute a special block of code for a special kind of error:

```
#Print one message if the try block raises a NameError and another for other errors:
#The try block will generate a NameError, because x is not defined:
try:
  print(x)
except NameError:
  print("Variable x is not defined")
except:
  print("Something else went wrong")
```

Output: Variable x is not defined

Else block

You can use the `else` keyword to define a block of code to be executed if no errors were raised:
In this example, the try block does not generate any error:

```
#The try block does not raise any errors, so the else block is executed:
try:
  print("Hello")
except:
  print("Something went wrong")
else:
  print("Nothing went wrong")
```

output:
Hello
Nothing went wrong

Finally block:

The `finally` block, if specified, will be executed regardless if the try block raises an error or not.

```
#The finally block gets executed no matter if the try block raises any errors or not:
#The finally block gets executed no matter if the try block raises any errors or not:
try:
  print(x)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```

output:
Something went wrong
The 'try except' is finished

Ex:

```
#The try block will raise an error when trying to write to a read-only file:
  try:
  f = open("demofile.txt")
  try:
        f.write("Lorum Ipsum")
```

```
except:
      print("Something went wrong when writing to the file")
finally:
      f.close()
except:
     print("Something went wrong when opening the file")
```

Output:
Something went wrong when writing to the file

Raise an exception:

As a Python developer you can choose to throw an exception if a condition occurs.
To throw (or raise) an exception, use the `raise` keyword.

Ex:
Raise an error and stop the program if x is lower than 0:

```
x = -1

if x < 0:
    raise Exception("Sorry, no numbers below zero")
```

output: Exception Sorry, no numbers below zero"
The `raise` keyword is used to raise an exception.
You can define what kind of error to raise, and the text to print to the user.

Example:

Raise a TypeError if x is not an integer:

```
x = "hello"
if not type(x) is int:
          raise TypeError("Only integers are allowed")
```

output: TypeError:Only integers are allowed

Explain Types of Exceptions in Python?

Exception Name	Description
BaseException	The base class for all built-in exceptions.
Exception	The base class for all non-exit exceptions.
ArithmeticError	Base class for all errors related to arithmetic operations.
ZeroDivisionError	Raised when a division or modulo operation is performed with zero as the divisor.
OverflowError	Raised when a numerical operation exceeds the maximum limit of a data type.
FloatingPointError	Raised when a floating-point operation fails.
AssertionError	Raised when an assert statement fails.
AttributeError	Raised when an attribute reference or assignment fails.
IndexError	Raised when a sequence subscript is out of range.
KeyError	Raised when a dictionary key is not found.
MemoryError	Raised when an operation runs out of memory.
NameError	Raised when a local or global name is not found.
OSError	Raised when a system-related operation (like file I/O) fails.
TypeError	Raised when an operation or function is applied to an object of inappropriate type.
ValueError	Raised when a function receives an argument of the right type but inappropriate value.
ImportError	Raised when an import statement has issues.
ModuleNotFoundError	Raised when a module cannot be found.