What is Python? List and explain feature of Python.

Python is a dynamic, high-level, free open source, and interpreted programming language. It supports object-oriented programming as well as procedural-oriented programming.

- Free and Open Source - Python language is freely available at the official website
- Easy to code - Python is a high-level programming language. Python is very easy to learn the language as compared to other languages like C, C#, Javascript, Java, etc.
- Easy to Read - learning Python is quite simple
- Object-Oriented Language - Python supports object-oriented language and concepts of classes, object encapsulation, etc.
- GUI Support
- Large Community Support
- Easy to Debug
- Portable language
- Frontend and backend development

Write a programe to perform pallindrome of string and number.

```
num = 123
org = num
rev = 0

while num != 0:
    rem = num % 10
    rev = (rev * 10) + rem
    num = int(num/10)

if rev == org:
    print("Is palindrome")
else:
    print("Not palindrome")
```

Explain type conversion of variable in Python.

- In python we can convert the data type of variable using some built in function

- This is useful if you want to convert any value to a different type
  - int() : converts value to integer
  - float() : converts value to float
  - str() : converts value to string

- we can also convert sequence data types like list, tuple and set
  - list() : converts a sequence to list
  - tuple() : converts a sequence to tuple
  - set() : converts a sequence to set

```
a = 10

x = str(a)       # "10"
x = int(a)       # 10
x = float(a)     # 10.0
```

Explain if…else statement with example.

- In Python, the `if...else` statement is used for conditional execution of code. It allows you to execute a specific block of code if a certain condition is true, and another block of code if the condition is false. Here's an explanation with an example:

```
# Example: Checking if a number is even or odd

num = 7

if num % 2 == 0:
    print(num," is even")
else:
    print(num," is odd")

# output : 7 is odd
```

- In this example, we're checking whether the number stored in the variable `num` is even or odd. The condition `num % 2 == 0` checks if the remainder of the division of `num` by 2 is equal to 0. If the condition is true, the code within the first block will be executed, which prints that the number is even. If the condition is false, the code within the `else` block will be executed, which prints that the number is odd.

Explain break with example

- In python break statement is used to exit the loop when a certain condition is true

```
# example of break

for i in range(1,11):
    print(i)

    if i == 5:
        break

# output : 1 2 3 4 5
```

- In this example we are using a for loop to print numbers from 1 to 10 (< 11)
- Here, If condition is given as i == 5, so, when the value of i will be 5 the loop will be terminated
- The loop terminates after printing "5" because the `break` statement is encountered when `i` is equal to 5. As a result, the loop stops, and the iteration that would print numbers 6 through 9 is not executed.

Explain list, set and tuples with example of each

- List: A list is an ordered collection of items that can hold elements of various data types. Lists are mutable, meaning you can modify their contents after they are created.

- my_list = [1, 2, 'hello', 3.14, True]

- Set: A set is an unordered collection of unique elements. Sets are useful for eliminating duplicate values. Sets are mutable.

- my_set = {3, 1, 4, 1, 5}
- #Duplicate value '1' is ignored

- Tuple: A tuple is an ordered collection of elements, similar to a list. However, tuples are immutable, meaning their contents cannot be changed once they are created. Tuples are often used to represent data that shouldn't be modified.

- my tuple = (10, 'apple', 3.14)

Difference between bracket, braces and parantheses

- Parentheses `()` - Parentheses are used primarily for grouping and controlling the order of operations in expressions and function calls. They are also used to define tuples, call functions, and create generator expressions.

```
result = (2 + 3) * 4
print("Hello, world!")
my_tuple = (1, 2, 3)
```

- Brackets `[]` - Brackets are used for creating lists, indexing lists

```
my_list = [1, 2, 3]
first_element = my_list[0]
 # Accesses the first element of the list
```

- Braces `{}` - Braces are mainly used to define dictionaries and sets .

```
my_dict = {'name': 'Alice', 'age': 30}
my_set = {1, 2, 3}
```

How to define and call functions in python?

- Defining a Function:
- To define a function, you use the `def` keyword followed by the function name, a pair of parentheses `()`, and a colon `:`
- Any code you want the function to execute is indented inside the function definition.

```
def show():
    print("Hello world")
```

- Calling a Function:
- To call a function, you use the function name followed by parentheses `()` containing the required arguments (if any).

```
show()
```

Python program to perform recursion

```
# example of a recursive function for factorial of a number:

def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

num = 5
result = factorial(num)
print("The factorial of",num,"is",result)

# output : The factorial of 5 is 120
```

Explain various string operations that can be performed using operators in Python.

- Concatenation (+) - You can concatenate (join together) two or more strings using the + operator

```
str1 = "Hello"
str2 = "World"
result = str1  + str2
print(result)       # Output: "Hello World"
```

- Repetition (*) - You can repeat a string multiple times using the * operator

```
text = "Repeat me! "
print( text * 3)
# Output: "Repeat me! Repeat me! Repeat me! "
```

- Membership (in, not in): - You can check if a substring exists in a string using the in and not in operators.

```
text = "Hello, World"
print("Hello" in text) # Output: True
```

- Indexing and Slicing - You can access individual characters in a string by indexing, You can extract substrings from a string using slicing

```
text = "Python"
text[0]
text[0:4]
```

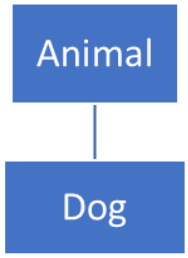
Explain inheritance with its types.

- Inheritance is a concept in object-oriented programming (OOP) that allows a new class to inherit properties and behaviours (attributes and methods) from an existing class. The existing class is referred to as the "base class" or "parent class," and the new class is the "derived class" or "subclass" or “Child class”

- Single Inheritance

```
class Animal:
    pass

class Dog(Animal):
    pass
```

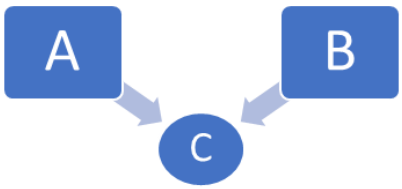


- Multiple Inheritance - -A class is derived from two or more class

```
class A:
    pass

class B:
    pass

class C(A, B):
    pass
```

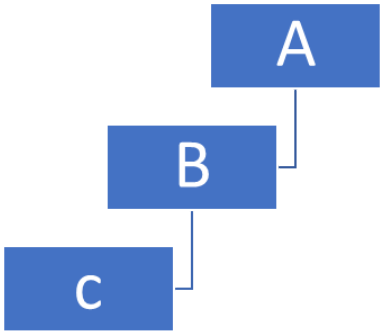


- Multilevel Inheritance - -A class is derived from another derived class

```
class A:
    pass

class B(A):
    pass

class C(B):
    pass
```



What is dictionary? How is it created?

- Dictionary in python is used to store key value pair
- Each key is associated with a value
- Dictionaries are created using {} and with key-value pairs separated by colons “:”

```
my_dict = {
"name": "Alice",
"age": 30,
"occupation": "engineer"
}
```

- Here “name” “age” and “occupation” are all keys
- Whereas “Alice”, 30 and “engineer” are values associated to the keys

Write a programe for Febonacci Series

```
n = int(input("Enter the range :
"))
a = 0
b = 1
c = a + b

for i in range(n):
    print(a)
    a = b
    b = c
    c = a + b

# output : Enter the range : 5
# 0 1 1 2 3
```

Explain Python Modules

- Modules is same as code library
- There are two types of modules (user defined and built in)
- Userdefined modules are the one which the user creates
- Builtin modules are already avaliable in the python
- eg. Math module
- dir() funtion is used to list down all the functions in the modules

min()
Returns minimum/lowest value
print(min(5,6,7,2,45))
=> 2

max()
Returns maximum/greatest value
print(max(5,6,7,2,45))
=>45

abs()
Returns absolute(positive) value of the number
abs(-30)
=>30

pow()
pow(x,y) returns the value of x to power of y
pow(4,2)
=>16

What is method overriding?

- When a child class have a method which is already defined in parent class this method gets overridden by the child class method this is called as method overriding

```
class Parent():
    def show(self):
    print("inside parent")

class Child(Parent):
    def show(self):
    print("inside Child")

obj = Child()
obj.show() #this will call the method of child class

# output : inside Child
```

- Here we have a child class which is derived by the parent class
- Both the classes have show() method
- When object of Child class is created the show() method of a Parent class is overridden by the show() function of the Child class

Explain For loop in Python

- For loop loops a block of code as per the range mentioned to it
- In for loop, we can use range, list, set, tuple etc
- range in for loop takes 3 parameters, out of which 2 are optional
- range(start, end, increment)
  - start : start of the range, if not mentioned range starts with 0
  - end : end of the range (does not count end)
  - increment : increments the range by 1

```
for i in range(1,6):
    print(i)

# starts from 1 and ends at < 6 (less than 6 = 5)
# output 1 2 3 4 5
```

```
for i in range(6):
    print(i)

# starts with 0 and ends with 5
# output 0 1 2 3 4 5
```

```
for i in range(1,6,2):
    print(i)

# starts with 1 and ends with 5, increments with 2
# output 1 3 5
```

Difference Between Array, List, Tuple, Set and Dictitonary

Array	List	Tuple	Set	Dictionary
[]	[]	()	{}	{}
ordered	ordered	ordered	unordered	Ordered
changeable	Changeable	unchangeable	Unchangeable but you can add or remove elements	changeable
Elements of Similar datatype	Elements of different datatype	Elements of different datatype	Elements of different datatype	Elements of different datatype
Allows duplicates	Allow duplicates	Allow duplicates	No duplicates allowed	No Duplicate keys

Explain Array and List functions

append()	Adds element at the end of the array or list a.append(‘abc’) this will add element ‘abc’ at the end of the array or list
pop()	will delete the element at the given index a.pop(2) this will remove the element at index 2
len()	Will return the length of/ number of elements in the array or list, print(len(my_list))
remove()	It will remove the particular element in the array or list a.remove(“abc”) this will remove ‘abc’ from the array or list note : in pop() index is given
clear()	It will clear the whole array or list
copy()	It is used to copy the whole array or list into another list b = a.copy() elements of ‘a’ are stored in ‘b’

count()	Returns the number of times the element occurs in the list or array a.count(10) this will return the count of how many 10s are present in the array or list
extend()	It is used to merge two list or array a.extend(b) elements of ‘b’ gets added into ‘a’
index()	Will return the index of the element a.index(‘cherry’) will give the index of element cherry in ‘a’
insert()	Will insert new element at the given position a.insert(2,”orange”) this will insert orange at index 2
reverse()	This is used to reverse the array or list a.reverse()
sort()	Sorts the list or array in ascending order (shortest to largerst) a.sort()

Explain Set functions

add()	Adds element in the set a.add(‘orange’) this will add element ‘orange’ in the set
update()	Adds elements of one set to another a.update(b)
len()	Will return the length of/ number of elements in the set print(len(a))
remove()	It will remove the particular element in set a.remove(“orange”) this will remove ‘orange’ from the set will show error if the element is not in set
discard()	Works just like remove() But this will not show error if the element is not in set a.discard(“orange”)
pop()	Will remove a random element as set does not have indexes a.pop()
clear()	will clear the whole set a.clear()

del	Will remove the set completely del my_set
union()	Will return new set will all the items from both set c = a.union(b)
difference()	Returns set containing the difference between two sets z = a.difference(b)
difference_update()	Removes the items in a set that are also in another set, eliminates common items a.difference_update(b)
intersection()	Returns common items in two sets z = x.intersection(y)
intersection_update()	Removes items from the set which are not in another set, keeps only common items a.intersection_update(b)

isdisjoint()	Checks if the two sets have any common element in it or not z = x.isdisjoint(y) returns true if there is no intersection
issubset()	Checks if the set is subset of another set or not z = x.issubset(y) returns true if set1 is subset of set2
issuperset()	Checks if the set is superset of another set or not z = x.issuperset(y) returns true if set1 is superset of set2

Explain Dictionary functions

values()	It will return the list with the values of dictionary x = my_dict.values()
items()	Will return list of all the items (key: value pair) x = my_dict.items()
keys()	Will return list of all the keys of dictionary x = my_dict.keys()
update()	Used to change the value of key in dictionary my_dict.update({“year” : 2000}) here we changed the value of year to 2000 it can also be used to add new elements in the dictionary
pop()	Will remove item with key name from the dictionary my_dict.pop(“year”) this will remove year key from the dictionary

Explain Dictionary functions

popitem()	Removes last added item from the dictionary my_dict.popitem()
del	Removes specific key or even the whole dictionary del my_dict[“year”] removes year del my_dict deletes whole dictionary
clear()	Clears the whole dictionary
copy()	Copies all the elements of one dictionary to another a = b.copy()

Explain concept of OOPs in Python

- In python Class is defined using “class” key work followed by class name and colons(:)
- The methods and properties are defined inside of this class definition
- All these methods and properties are public by default
- We can access these by creating objects of the class

```
class New:
    def my_function(self):
    print("hello world")

obj = New()
obj.my_function()
```

- Constructor - In python constructor is defined using __init__ function
- Constructor gets called every time class object is created
- We can pass parameters in constructor while creating objects

```
class New:
    def __init__(self):
    print("hello world")

obj = New()
```

- inheritance - Inheritance allows us to create a new class based on the existing class
- This new class is called as “subclass” or “derived” or “child class” it carries all the public and protected methods and properties of the parent class

```
class Parent:
    def my_function(self):
     print("hello world")

class Child(Parent):
    pass

obj = Child()
obj.my_function()
```

Difference Between Public, Private and protected data functions and members

Public	All methods and properties are public by default Public methods and properties can be accessed out of the class by the class objects
Protected	Protected methods and properties can be defined by using a single underscore (_) e.g., _name def _myfunction(): protected methods and properties can only be accessed within the class and by the class which are inherited by the class
Private	private methods and properties can be defined by using a double underscore (__) e.g., __name def__myfunction(): private methods and properties can only be accessed within the class

Explain File Handling in short

- open() funtion is used to open files while working with files
- there are some types of opening files
  - ‘r’ read mode, will show error if the file does not exist
  - ‘w’ write mode, will create file if the file does not exist
  - ‘a’ append mode, opens file for appending will create file if file does not exist

f = open("demo.txt","r")

f = open("D:\\myfiles\welcome.txt","r")
 # if the file is in another folder, we have to write the whole path

open()	Used to open file Cannot open file in read format if file does not exist f.open(“demo.txt”, ”r”)
close()	Close the file after opening f.close()
read()	Used to read the content in the file, we can specify how many characters to read f.read(5) this will read 5 characters
readline()	This function will only read one line in the file We can read multiple lines by calling readline() multiple times f.readline()
write()	Used to write or append in the file only if the file is opened in “w” or “r” format f.write(“helloworld”) this will write “hello world” in the file

Explain Regular Expressions

- A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern.
- Regular Expression can be used to check if a string contains the specified pattern
- ‘re’ module can be used to work with regular expressions

import re

Character	Description	Example
[]	A set of characters	"[a-m]"
\	Signals a special sequence (can also be used to escape special characters)	"\d"
.	Any character (except newline character)	"he..o"
^	Starts with	"^hello"
$	Ends with	"planet$"
*	Zero or more occurrences	"he.*o"
+	One or more occurrences	"he.+o"
?	Zero or one occurrences	"he.?o"
{}	Exactly the specified number of occurrences	"he.{2}o"
|	Either or	"falls|stays"
()	Capture and group	

Explain GUI in Python

- GUI, which stands for Graphical User Interface, refers to the visual elements and interactive components of a software application that allow users to interact with the program using graphical elements such as windows, buttons, icons, and menus, rather than solely relying on text-based commands.

- Tkinter is a standard builtin library of Python

- Features of Tkinter
  - simple to learn - Tkinter is easy to use

  - widgets - Tkinter provides GUI widgets that can be used as a interface
    - buttons - to trigger actions
    - labels - to display text
    - entry - for user input
    - frames
    - menus - dropdown list
    - canvas - for drawing shapes
    - checkboxes and radio buttons - for choices
    - scrollbars

  - layout management

  - customization - we can customize colors, fonts and sizes

Tkinter Program to create a window with labels and buttons

```
import tkinter as tk

def button_click():
    label.config(text="Button Clicked!")

# Create a window
root = tk.Tk()
root.title("Tkinter Example")

# Create a label
label = tk.Label(root, text="Hello, Tkinter!")
label.pack()

# Create a button
button = tk.Button(root, text="Click Me!",
command=button_click)
button.pack()

# Start the GUI event loop
root.mainloop()
```