# Python interview questions

Table of Contents

## Ques. What is Python?

- Python is a high-level, interpreted, general-purpose programming language.
- Python is an **interpreter** language. It means it executes the code line by line
- It was created by **Guido van Rossum**, and released in **1991**
- It is used for:
    - web development (server-side)
    - software development
    - mathematics
    - system scripting

**↑ Back to Top**

## Ques. Features of Python?

1. **Easy:-** Python is very easy to learn and understand; using this Python tutorial, any beginner can understand the basics of Python.
2. **Interpreted:-** It is interpreted(executed) line by line. This makes it easy to test and debug.
3. **Object-Oriented:-** The Python programming language supports classes and objects. We discussed these above.
4. **Free and Open Source:-** The language and its source code are available to the public for free; there is no need to buy a costly license.
5. **Portable:-** Since it is open-source, you can run Python on Windows, Mac, Linux or any other platform. Your programs will work without needing to the changed for every machine.
6. **GUI Programming:-** You can use it to develop a GUI (Graphical User Interface). One way to do this is through Tkinter.
7. **Large Library:-** Python provides you with a large standard library. You can use it to implement a variety of functions without needing to reinvent the wheel every time. Just pick the code you need and continue. This lets you focus on other important tasks.

**↑ Back to Top**

## Ques. Python Frameworks?

1. Django
2. Flask
3. Pyramid
4. Tornado
5. Bottle
6. web2py
7. NumPy
8. SciPy
9. Pylons

**↑ Back to Top**

## Ques. File Extensions in Python?

- **.py–** The normal extension for a Python source file
- **.pyc-** The compiled bytecode
- **.pyd-** A Windows DLL file
- **.pyo-** A file created with optimizations
- **.pyw-** A Python script for Windows
- **.pyz-** A Python script archive

**↑ Back to Top**

## Ques. What is the difference between .py and .pyc files?

- **.py** files contain the source code of a program. Whereas, **.pyc** file contains the bytecode of your program. We get bytecode after compilation of .py file (source code). .pyc files are not created for all the files that you run. It is only created for the files that you import. **↑ Back to Top**

## Ques. What is an Interpreted language?

An Interpreted language executes its statements line by line. Languages such as Python, Javascript, R, PHP, and Ruby are prime examples of Interpreted languages. **↑ Back to Top**

## Ques. What is a dynamically typed language?

Type-checking can be done at two stages:-

1. **Static-** Data Types are checked before execution.
2. **Dynamic-** Data Types are checked during execution. Python is an interpreted language, executes each statement line by line and thus type-checking is done on the fly, during execution. Hence, Python is a Dynamically Typed Language. **↑ Back to Top**

## Ques. Python Comments?

**single Line Comments:-** Comments starts with a #

```
#This is a comment
print("Hello, World!")

print("Hello, World!") #This is a comment
```

**Multi Line Comments(OR)Docstring**

- To add a multiline comment you could insert a # for each line.

```
#This is a comment
#written in
#more than just one line
print("Hello, World!")
```

- you can add a multiline string (triple quotes) in your code, and place your comment inside it.

```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

**↑ Back to Top**

## Ques. What is python Variables?

- Variables are containers for storing data values.
- A variable name must start with a letter or the underscore character.
- A variable name can only contain alpha-numeric characters and underscores

- A variable name cannot start with a number.
- Variable names are case-sensitive (age, Age and AGE are three different variables)
- Creating Variables

```
x = 5
y = "John"
print(x)
print(y)

Output:-
5
John
```

**Variables Casting**

- We want to specify the data type of a variable, this can be done with casting.

```
x = str(3)
y = int(3)
z = float(3)

print(x)
print(y)
print(z)

Output:-
3
3
3.0
```

**You can get the data type of a variable with the type() function.**

```
x = 5
y = "John"
print(type(x))
print(type(y))

Output:-
<class 'int'>
<class 'str'>
```

**Single or Double Quotes:-**

- String variables can be declared either by using single or double quotes:

```
x = "John"
print(x)
#double quotes are the same as single quotes:
x = 'John'
print(x)

Output:-
John
John
```

#### Case-Sensitive

- Variable names are case-sensitive.

```
a = 4
A = "Sally"

print(a)
print(A)

Output:-
4
Sally
```

#### Assign Multiple Values

- Python allows you to assign values to multiple variables in one line.

```
x, y, z = "Orange", "Banana", "Cherry"

print(x)
print(y)
print(z)

Output:-
Orange
Banana
Cherry
```

#### One Value to Multiple Variables

- And we can assign the same value to multiple variables in one line.

```
x = y = z = "Orange"
```

```
print(x)
print(y)
print(z)

Output:-
Orange
Orange
Orange
```

#### Unpack a Collection

- If we have a collection of values in a list, tuple etc. Python allows you to extract the values into variables. This is called unpacking.

```
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits

print(x)
print(y)
print(z)

Output:-

apple
banana
cherry
```

↑ Back to Top

## Ques. Global Variables?

- Variables that are created outside of a function are known as global variables.
- Global variables can be used by everyone, both inside of functions and outside.
- Create a variable outside of a function, and use it inside the function.

```
x = "awesome"

def myfunc():
  print("Python is " + x)

myfunc()

Output:- Python is awesome
```

- If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

```
x = "awesome"

def myfunc():
  x = "fantastic"
  print("Python is " + x)

myfunc()

print("Python is " + x)

Output:-
Python is fantastic
Python is awesome
```

↑ Back to Top

### Ques. global Keyword?

- To create a global variable inside a function, you can use the global keyword.

```
def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)

Output:- Python is fantastic
```

- Also, use the global keyword if you want to change a global variable inside a function.

```
x = "awesome"

def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)

Output:- Python is fantastic
```

↑ Back to Top

### Ques. What are the common built-in data types in Python?

- Variables can store data of different types, and different types can do different things.
- Python has the following data types built-in by default, in these categories:

1. **Text Type:-** str
2. **Numeric Types:-** int, float, complex
3. **Sequence Types:** list, tuple, range
4. **Mapping Type:** dict
5. **Set Types:** set, frozenset
6. **Boolean Type:** bool
7. **Binary Types:** bytes, bytearray, memoryview
8. **None Type:** NoneType

- **Numeric:-**
    - Integers :- int stores integers eg a=100, b=25, c=526, etc.
    - Float :- float stores floating-point numbers eg a=25.6, b=45.90, c=1.290, etc.
    - Complex Numbers:- complex stores numbers eg a=3 + 4j, b=2 + 3j, c=complex(4,6), etc.
    - long:- long stores higher range of integers eg a=908090999L, b=-0x1990999L, etc.
- **Sequence Type:-**
    - String
    - List
    - Tuple
    - range
- **Boolean:-** There can be only two types of value in the Boolean data type of Python, and that is True or False.
- **Set Type:-**
    - set:-
    - frozenset:-
- **Dictionary**
- **long:-** long stores higher range of integers eg a=908090999L, b=-0x1990999L, etc.
- **Mapping Types:-**
    - dict:- Stores comma-separated list of key: value pairs.
- **Binary Types:-**
    - bytes
    - bytearray
    - memoryview
- We can get the data type of any object by using the **type()** function.

```
x = 5
print(type(x))

output:- <class 'int'>
```

↑ Back to Top

## Ques. Primitive Data Structures/Non-Primitive Data Structures

- Primitive Data Structures
  - Integers
  - Float
  - Strings
  - Boolean
- Non-Primitive Data Structures
  - Arrays
  - Lists
  - Tuples
  - Dictionary
  - Sets ↑ Back to Top

## Ques. Type Casting/Type Conversion?

- The process of converting the value of one data type (integer, string, float, etc.) to another data type is called type conversion.
- Jab bhi hum ek data type ki value ko dusre data type ki value mai convert karte hai to usi ko hum bolte hai type conversion.

```
# Integers
x = int(1)
y = int(2.8)
z = int("3")
print(x)
print(y)
print(z)

Output:-
1
2
3

# Floats
x = int(1)
y = int(2.8)
z = int("3")
print(x)
print(y)
print(z)
```

```
Oytput:-
1
2
3

# Strings
x = str("s1")
y = str(2)
z = str(3.0)
print(x)
print(y)
print(z)

Output:-
s1
2
3.0
```

Python has two types of type conversion.

1. **Implicit Type Conversion: -** In Implicit type conversion, Python automatically converts one data type to another data type. This process doesn't need any user involvement.

```
num_int = 123
num_flo = 1.23

num_new = num_int + num_flo

print("datatype of num_int:",type(num_int))
print("datatype of num_flo:",type(num_flo))

print("Value of num_new:",num_new)
print("datatype of num_new:",type(num_new))

Output:-
datatype of num_int: <class 'int'>
datatype of num_flo: <class 'float'>
Value of num_new: 124.23
datatype of num_new: <class 'float'>
```

2. **Explicit Type Conversion: -** In Explicit Type Conversion, users convert the data type of an object to required data type. We use the predefined functions like int(), float(), str(), etc to perform explicit type conversion.

```
num_int = 123
num_str = "456"

print("Data type of num_int:",type(num_int))
print("Data type of num_str before Type Casting:",type(num_str))
```

```
num_str = int(num_str)
print("Data type of num_str after Type Casting:",type(num_str))

num_sum = num_int + num_str

print("Sum of num_int and num_str:",num_sum)
print("Data type of the sum:",type(num_sum))

Output:-
Data type of num_int: <class 'int'>
Data type of num_str before Type Casting: <class 'str'>

Data type of num_str after Type Casting: <class 'int'>

Sum of num_int and num_str: 579
Data type of the sum: <class 'int'>
```

↑ Back to Top

## Ques. Python Operators?

- Arithmetic Operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators
- Arithmetic Operators

| Operator | Name | Example |
|---|---|---|
| + | Addition | x + y |
| - | Subtraction | x-y |
| * | Multiplication | x * y |
| / | Division | x / y |
| % | Modulus | x % y |
| ** | Exponentiation | x ** y |
| // | Floor division | x // y |

- Assignment operators

| Operator | Example | Same As | Try it |
|---|---|---|---|
| = | x = 5 | x = 5 | x = 5<br>print(x)<br>5 |
| += | x += 3 | x = x + 3 | x = 5<br>x += 3<br>print(x)<br>8 |

```
-=  x -= 3  x = x - 3
*=  x *= 3  x = x * 3
/=  x /= 3  x = x / 3
%=  x %= 3  x = x % 3
//= x //= 3 x = x // 3
**= x **= 3 x = x ** 3
&=  x &= 3  x = x & 3
|=  x |= 3  x = x | 3
^=  x ^= 3  x = x ^ 3
>>= x >>= 3 x = x >> 3
<<= x <<= 3 x = x << 3
```

- Python Comparison Operators

↑ Back to Top

## Ques. difference between membership and identity operators?

- **Membership operators:-** Membership operators are operators used to validate the membership of a value.

1. **in operator :** The 'in' operator is used to check if a value exists in a sequence or not.

```
x = ["apple", "banana"]
print("banana" in x)

Output:- True

# Ex:-
list1=[0,2,4,6,8]
list2=[1,3,5,7,9]
#Initially we assign 0 to not overlapping
check=0

for item in list1:
    if item in list2:
        #Overlapping true so check is assigned 1
        check=1
```

```
if check==1:
    print("overlapping")
else:
    print("not overlapping")

Output:- not overlapping
```

2. **'not in' operator-** Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.

```
x = ["apple", "banana"]
print("pineapple" not in x)

Output:- True

# Ex-2
a = 70
b = 20
list = [10, 30, 50, 70, 90 ];

if ( a not in list ):
 print("a is NOT in given list")
else:
 print("a is in given list")

if ( b not in list ):
 print("b is NOT present in given list")
else:
  print("b is in given list")
```

- **Identity operators** evaluate whether the value being given is of a specific type or class. These operators are commonly used to match the data type of a variable.

1. **is operator:-** The is operator returns true if the variables on either side of the operator point to the same object. Otherwise, it returns false.

```
x = 'Educative'
if (type(x) is str):
    print("true")
else:
    print("false")
Output:- True
```

2. **is not operator:-** The is not operator returns false if the variables on either side of the operator point to the same object. Otherwise, it returns true.

```
x = 6.3
if (type(x) is not float):
    print("true")
else:
    print("false")

Outpur:- False
```

### Ques. What is If Else?

- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
output:- a is greater than b
```

- **Short Hand If**

```
a = 200
b = 33

if a > b: print("a is greater than b")
output:- "a is greater than b"
```

- **Short Hand If ... Else**

```
a = 2
b = 330

print("A") if a > b else print("B")
output:- B
```

- One line if else statement, **with 3 conditions:**

```
a = 330
b = 330

print("A") if a > b else print("=") if a == b else print("B")
Output:- =
```

- The **and** keyword is a logical operator, and is used to combine conditional statements.

```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
output:- Both conditions are True
```

- The **Or** keyword is a logical operator, and is used to combine conditional statements.

```
a = 200
b = 33
c = 500
if a > b or a > c:
  print("At least one of the conditions is True")
output:- At least one of the conditions is True
```

- **Nested If** You can have if statements inside if statements, this is called nested if statements.

```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
Output:-
Above ten,
and also above 20!
```

- **The pass Statement** if statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error.

```
a = 33
b = 200
```

```
if b > a:
  pass
Output:-
```

### Ques. Python While Loops?

- With the while loop we can execute a set of statements as long as a condition is true.

```
i = 1
while i < 6:
  print(i)
  i += 1

output:-
1
2
3
4
5
```

- **The break Statement:-** With the break statement we can stop the loop even if the while condition is true:

```
i = 1
while i < 6:
  print(i)
  if (i == 3):
    break
  i += 1

Output:-
1
2
3
```

- **The continue Statement:-** With the continue statement we can stop the current iteration, and continue with the next.

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)

Output:-
1
```

```
2
4
5
6
```

↑ Back to Top

## Ques. What is Python JSON?

- JSON **JavaScript Object Notation** is a format for structuring data. It is mainly used for storing and transferring data between the browser and the server.
- Python has a built-in package called json, which can be used to work with JSON data.
- **Convert JSON to Python:-** If you have a JSON string, you can parse it by using the **json.loads()** method.

```
import json
# some JSON:
x = '{ "name":"John", "age":30, "city":"New York"}'
y = json.loads(x)
# the result is a Python dictionary:
print(y["age"])

Output:- 30
```

- **Convert Python to JSON:-** If you have a Python object, you can convert it into a JSON string by using the **json.dumps()** method.

```
 import json

# a Python object (dict):
x = {
  "name": "John",
  "age": 30,
  "city": "New York"
}

# convert into JSON:
y = json.dumps(x)

# the result is a JSON string:
print(y)

Output:- {"name": "John", "age": 30, "city": "New York"}
```

- Format the Result:- Use the **indent** parameter to define the numbers of indents:

```
import json

x = {
  "name": "John",
  "age": 30,
  "married": True,
  "divorced": False,
  "children": ("Ann","Billy"),
  "pets": None,
  "cars": [
    {"model": "BMW 230", "mpg": 27.5},
    {"model": "Ford Edge", "mpg": 24.1}
  ]
}

# use four indents to make it easier to read the result:
print(json.dumps(x, indent=4))

Output:-
{
    "name": "John",
    "age": 30,
    "married": true,
    "divorced": false,
    "children": [
        "Ann",
        "Billy"
    ],
    "pets": null,
    "cars": [
        {
            "model": "BMW 230",
            "mpg": 27.5
        },
        {
            "model": "Ford Edge",
            "mpg": 24.1
        }
    ]
}
```

- Use the **sort_keys** parameter to specify if the result should be sorted or not:

```
print(json.dumps(x, indent=4, sort_keys=True))

Output:-
{
    "age": 30,
    "cars": [
        {
            "model": "BMW 230",
```

```
            "mpg": 27.5
        },
        {
            "model": "Ford Edge",
            "mpg": 24.1
        }
    ],
    "children": [
        "Ann",
        "Billy"
    ],
    "divorced": false,
    "married": true,
    "name": "John",
    "pets": null
}
```

↑ Back to Top

### Ques. What is PEP 8?

- PEP stands for **Python Enhancement Proposal**.
- It is a set of rules that specify how to format Python code for maximum readability.
- PEP8 is a document that provides various guidline to write the readable in python.
- PEP8 describe how the developers can write the beautiful code. ↑ Back to Top

### Ques. How to get Id()?

id() function takes a single parameter object.

```
a = 5
print(id(a))
```

↑ Back to Top

### Ques. a=1, b=1 does both have same Id or not?

Two variables in Python have same id, but not in lists.

```
a = 10
b = 10
print(id(a))
print(id(b))

Output:-
9788992
9788992
-------------------------------------------
# In List
```

```
a = [1, 2, 3]
b = [1, 2, 3]
print(a is b)

Output:- False
------------------------------------------------
# In tuples
a = (1, 2, 3)
b = (1, 2, 3)
print(a is b)

Output:- True
```

↑ Back to Top

### Ques. Is indentation required in python?

Indentation is necessary for Python. It specifies a block of code. All code within loops, classes, functions, etc is specified within an indented block. It is usually done using four space characters. If your code is not indented necessarily, it will not execute accurately and will throw errors as well. ↑ Back to Top

### Ques. Python Deep Copy and Shallow Copy?

**Deep Copy:-** In deep copy any changes made to a copy of object do not reflect in the orginal object.

```
import copy

old_list = [[1, 1, 1], [2, 2, 2], [3, 3, 3]]
new_list = copy.deepcopy(old_list)

new_list[1][0] = 'BB'

print("Old list:", old_list)
print("New list:", new_list)

Output:-
Old list: [[1, 1, 1], [2, 2, 2], [3, 3, 3]]
New list: [[1, 1, 1], ['BB', 2, 2], [3, 3, 3]]
```

**Shallow copy:-** A shallow copy creates a new object which stores the reference of the original elements.

```
import copy

old_list = [[1, 1, 1], [2, 2, 2], [3, 3, 3]]
new_list = copy.copy(old_list)

new_list[1][1] = 'AA'

print("Old list:", old_list)
```

```
print("New list:", new_list)

Output:-
Old list: [[1, 1, 1], [2, 'AA', 2], [3, 3, 3]]
New list: [[1, 1, 1], [2, 'AA', 2], [3, 3, 3]]
```

↑ Back to Top

### Ques. What is Lambda/Anonymous Function?

- A lambda function is a small anonymous function.
- In Python, an anonymous function is a function that is defined without a name.
- While normal functions are defined using the **def** keyword in Python, anonymous functions are defined using the **lambda** keyword.
- **Notice** that the anonymous function does not have a return keyword. This is because the anonymous function will automatically return the result of the expression in the function once it is executed.

```
def add(a,b):
print(a+b)
add(5,10)

# Using Lambda function
x = lambda a: a + 10
print(x(5))

Output:- 15
```

- You can use lambda function in **filter()**

```
# filter() function is used to filter a given iterable (list like object) using
another function that defines the filtering logic.
# Syntex:- filter(object, iterable)
# The object here should be a lambda function which returns a boolean value.
mylist = [2,3,4,5,6,7,8,9,10]
list_new  = list(filter(lambda x : (x%2==0), mylist))
print(list_new)

Output:- [2, 4, 6, 8, 10]
```

- We can use lambda function in **map()**

```
# map() function applies a given function to all the itmes in a list and returns
the result. Similar to filter(), simply pass the lambda function and the list (or
any iterable, like tuple) as arguments.

mylist = [2,3,4,5,6,7,8,9,10]
```

```
list_new  = list(map(lambda x : x%2, mylist))
print(list_new)

Output:- [0, 1, 0, 1, 0, 1, 0, 1, 0]
```

- You can use lambda function in **reduce()** as well

```
# reduce() function performs a repetitive operation over the pairs of the elements
in the list. Pass the lambda function and the list as arguments to the reduce()
function. For using the reduce() function, you need to import reduce from
functools librray.

from functools import reduce
list1 = [1,2,3,4,5,6,7,8,9]
sum = reduce((lambda x,y: x+y), list1)
print(sum)

Output:- 45 //i.e 1+2, 1+2+3 , 1+2+3+4 and so on.
---------------------------------------------------------------------------
# How to use lambda function to manipulate a Dataframe
# You can also manipulate the columns of the dataframe using the lambda function.
It's a great candidate to use inside the apply method of a dataframe. I will be
trying to add a new row in the dataframe in this section as example.

import pandas as pd
df = pd.DataFrame([[1,2,3],[4,5,6]],columns = ['First','Second','Third'])
df['Forth']= df.apply(lambda row: row['First']*row['Second']* row['Third'],
axis=1)
df

Output:-
|   | First | Second | Third | Forth |
| 0 |  1    |    2   |   3   |  6    |
| 1 |  4    |  5     |   6   |  120  |
```

↑ Back to Top

## Ques. What is Decorators?

- A decorator is a design pattern in Python that allows a user to add new functionality to an existing object without modifying its structure. Decorators are usually called before the definition of a function you want to decorate.
- Decorators are used to add some design patterns to a function without changing its structure.
- A decorator function is a function that accepts a function as parameter and return a function(decorator ek function hai jo as a argument leta bhi function hai and return bhi function karta hai).
- Decorators allow us to wrap another function in order to extend the behaviour of the wrapped function, without permanently modifying it.

```
def decor(fun):
    def inner():
        print('befor inhance')
        fun()
        print('after inhance')
    return inner

def num():
    print('hello mohit')

result = decor(num)
result()

# Example2:-
def decor(fun1):
    def inner():
        print('befor inhance')
        fun1()
        print('after inhance')
    return inner

@decor
def num():
    print('hello mohit')

num()


Output:-
befor inhance
hello mohit
after inhance
-----------------------------------------------------
def num_decor(num):
    def inner():
        a = num()
        add = a + 5
        return add
    return inner

@num_decor
def num():
    return 10

print(num())

Output:- 15

# Example:-
def num_decor(num):
    def inner():
        a = num()
        add = a + 5
```

```
        return add
    return inner

def num():
    return 10

result = num_decor(num)
print(result())

Output:- 15
```

↑ Back to Top

## Ques. What is Generator Functions?

- Generator are functions that returns a sequens of value. we use yield statement to return the from function.
- Yield statement returns the element from a generator function into a genrater object.(EX:- yield a)
- This function is used to retrieve element by element from a generator object.(Ex:- next(gen_obj))
- A generator is a special type of function which does not return a single value, instead, it returns an iterator object with a sequence of values.
- In a generator function, a **yield** statement is used rather than a return statement.
- The generator function cannot include the return keyword. If you include it, then it will terminate the function.
- The difference between yield and return is that yield returns a value and pauses the execution while maintaining the internal states, whereas the return statement returns a value and terminates the execution of the function.

```
def mygenerator():
    print('First item')
    yield 10

    print('Second item')
    yield 20

    print('Last item')
    yield 30

gen = mygenerator()
print(next(gen))
print(gen.__next__())  # 2 option to write the next function
print(next(gen))
print(next(gen))

Output:-
First item
10
Second item
20
Last item
```

```
30
Traceback (most recent call last):
File "<string>", line 22, in <module>
StopIteration

------------------------------------------------------------
# 2nd Option
gen = mygenerator()
while True:
    try:
        print ("Received on next(): ", next(gen))
    except StopIteration:
        break
Output:-
First item
Received on next():  10
Second item
Received on next():  20
Last item
Received on next():  30

# Example 2:-
def bhai(a,b):
    yield a+b
    yield a-b
result = bhai(3,2)
print(next(result))
print(next(result))

Output:-
5
1

# Example 3:-
def numberPrint():
    n = 1
    while n <= 10:
        sq = n*n
        yield sq
        n += 1
values = numberPrint()
for i in values:
    print(i)

Output:-
1
4
9
16
25
36
49
64
```

```
81
100
```

### Ques. What is Monkey Patching?

- The term monkey patching refers to dynamic(or run time) modifictaion of class or method.
- A class or method can be changed at the runtime.

```
class A:
   def hello(self):
      print ("The hello() function is being called")

def monkey_f():
    print ("monkey_f() is being called")

#normal class method call
obj = A()
obj.hello()

#calling class method after monkey patch
obj.hello = monkey_f
obj.hello()

Output:-
The hello() function is being called
monkey_f() is being called
```

↑ Back to Top

### Ques. What is Magic Method Or Dunder Methods?

- Magic methods in Python are the special methods that start and end with the double underscores. They are also called dunder methods.
- Built-in classes in Python define many magic methods.
- The dir() function can be used to see the number of magic methods inherited by a class.

```
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__',
'__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__',
'__floor__', '__floordiv__', '__format__', '__ge__', '__getattribute__',
'__getnewargs__', '__gt__', '__hash__', '__index__', '__init__',
'__init_subclass__', '__int__', '__invert__', '__le__', '__lshift__', '__lt__',
'__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__or__', '__pos__',
'__pow__', '__radd__', '__rand__', '__rdivmod__', '__reduce__', '__reduce_ex__',
'__repr__', '__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__', '__ror__',
'__round__', '__rpow__', '__rrshift__', '__rshift__', '__rsub__', '__rtruediv__',
'__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__',
'__truediv__', '__trunc__', '__xor__', 'bit_length', 'conjugate', 'denominator',
'from_bytes', 'imag', 'numerator', 'real', 'to_bytes']
```

```
class emp():
    def __init__(self,name,salery):
        self.name = name
        self.salery = salery

    def __len__(self):            # Magic method
        return len(self.name)

obj = emp('mohit', 422573)
print(len(obj))

Output:- 5
```

↑ Back to Top

## Ques. What are pickling and unpickling in Python?

- pickling and unpickling should be done using binary files since they support byte steam.
- Python pickle module is used for serializing and de-serializing python object structures. The process to converts any kind of python objects (list, dict, etc.) into byte streams (0s and 1s) is called pickling or serialization or flattening or marshalling. We can converts the byte stream (generated through pickling) back into python objects by a process called as unpickling. **Pickling:**
- pickling is a process of converting a class object into a byte stream so that it can be stored into a file. this is also called as object serialization.
- Pickling is the name of the serialization process in Python. Any object in Python can be serialized into a byte stream and dumped as a file in the memory.
- The function used for the above process is **pickle.dump()**.

**unpickling:**

- unpickling is a process whereby byte stream is converted back into a class object. it is inverse operation of pickling. this is also called as de-serialization.
- The function used for the above process is **pickle.load()**.

```
import pickle
class Student:
    def __init__(self, name, roll, address):
        self.name = name
        self.roll = roll
        self.address = address

    def display(self):
        print(f"name {self.name} roll is {self.roll} address {self.address}")

with open('student.dat', mode='wb') as f:
    stu1 = Student('mohit', 001, 'mainpuri')
    stu2 = Student('rohit', 001, 'agra')
    pickle.dump(stu1, f)
```

```
    pickle.dump(stu2, f)
    print('pickling Done')

with open('student.dat', mode='rb') as f:
    obj1 = pickle.load(f)
    obj2 = pickle.load(f)
    print('unpikling Done!!!')
    obj1.display()
    obj2.display()
```

↑ Back to Top

### Ques. **What is Scope in Python?**

- Every object in Python functions within a scope. A scope is a block of code where an object in Python remains relevant. Namespaces uniquely identify all the objects inside a program.

### Ques. What are global, protected and private attributes in Python?

- **Global variables** are public variables that are defined in the global scope. To use the variable in the global scope inside a function, we use the **global** keyword.
- **Protected attributes** are attributes defined with an **single underscore** prefixed to their identifier eg. _mohit. They can still be accessed and modified from outside the class they are defined in but a responsible developer should refrain from doing so.
- **Private attributes** are attributes with **double underscore** prefixed to their identifier eg. __mohit. They cannot be accessed or modified from the outside directly and will result in an AttributeError if such an attempt is made. ↑ Back to Top

### Python Collections (Arrays)?

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered and unindexed. No duplicate members.
- **Dictionary** is a collection which is unordered, changeable and indexed. No duplicate members. ↑ Back to Top

### Ques. How would you convert a list to an array?

- Python list is a linear data structure that can hold heterogeneous elements. Python does not have a built-in array data type. If you want to create an array in Python, then use the numpy library.
- To *install numpy* in your system, type the following command.
- python3 -m pip install numpy
- **1. Using numpy.array()**

```
import numpy as np

elon_list = [11, 21, 19, 18, 29]
```

```
elon_array = np.array(elon_list)

print(elon_array)
print(type(elon_array))

Output:-
[11 21 19 18 29]
<class 'numpy.ndarray'>

# 2. Using numpy.asarray()
import numpy as np

elon_list = [11, 21, 19, 18, 29]
elon_array = np.asarray(elon_list)

print(elon_array)
print(type(elon_array))

Output:- [11 21 19 18 29]
<class 'numpy.ndarray'>
```

### Ques. What do * (single asterisk) and ** (double asterisk) do for parameters in Python?

### Ques. What is * args and ** kwargs in Python?

1. *args (Non Keyword Arguments)
2. **kwargs (Keyword Arguments)

- **Single Asterisk:-** Single Asterisk allows the developer to pass a variable number of Positional parameters and automatically converts the input values in the form of tuples. At the same time.

```
def print_colors(*args):
    print(args)
print_colors('red','blue','green','yellow')

Output:- ('red', 'blue', 'green', 'yellow')
```

- **Double Asterisks** Double Asterisk allows the users to pass a variable number of Keyword parameters in the form of a Dictionary.

```
def print_numbers(**kwargs):
  for key, value in kwargs.items():
      print (f"{key} is a {value}")
print_numbers(mohit="TL", two="two",three=3,four="four")

Output:-
mohit is a TL
two is a two
```

```
three is a 3
four is a four
```

↑ Back to Top

### Ques. What is built-in module in Python?

https://docs.python.org/3/py-modindex.html
Example

```
>>> import html
>>> import html.parser
import mysql.connector
```

ye buil in packege hote hai
agar or karne hai to pip ki help sa karenge

↑ Back to Top

### Ques. How is memory managed in Python?

- Python also has an inbuilt garbage collector, which recycles all the unused memory and so that it can be made available to the heap space.
- Memory management in python is managed by Python private heap space. All Python objects and data structures are located in a private heap. The programmer does not have access to this private heap. The python interpreter takes care of this instead.
- The allocation of heap space for Python objects is done by Python's memory manager. The core API gives access to some tools for the programmer to code. ↑ Back to Top

### Ques. What is Map, Filter, Reduce?

sdafasf fasf das f

#### Ques. What is an Iterable?

- An Iterable is an object that implements the **iter**() method and returns an iterator object or an object that implements **getitem**() method (and should raise an IndexError when indices are exhausted).

#### What is Iterators?

- iterator obj print the value one by one.
- An iterator is an object that contains a countable number of values.
- An iterator is an object that can be iterated upon, meaning that you can traverse through all the values.
- Technically, in Python, an iterator is an object which implements the iterator protocol, which consist of the methods **iter**() and **next**().

```

```

```
# Example1
mytuple = ("apple", "banana", "cherry")
myit = iter(mytuple)

print(next(myit))
print(next(myit))
print(next(myit))

Output:-
apple
banana
cherry

# Example 2
mystr = "banana"
myit = iter(mystr)

print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))

Output:-
b
a
n
a
n
a

# Looping Through an Iterator
# The for loop actually creates an iterator object and executes the next() method
for each loop
mytuple = ("apple", "banana", "cherry")

for x in mytuple:
  print(x)

Output:-
apple
banana
cherry
```

## Ques. Difference between Iterators and iterable?

- Every iterator is also an iterable, but not every iterable is an iterator.

| Iterable | Iterator |
|---|---|

| Iterable | Iterator |
|---|---|
| An Iterable is basically an object that any user can iterate over. | An Iterator is also an object that helps a user in iterating over another object (that is iterable). |
| We can generate an iterator when we pass the object to the iter() method. | We use the **next**() method for iterating. This method helps iterators return the next item available from the object. |
| Every iterator is basically iterable. | Not every iterable is an iterator. |

Ques. Difference between generator and iterators in python?

| Iterator | Generator |
|---|---|
| Class is used to implement an iterator | Function is used to implement a generator. |
| Local Variables aren't used here. | All the local variables before the yield function are stored. |
| Iterators are used mostly to iterate or convert other objects to an iterator using iter() function. | Generators are mostly used in loops to generate an iterator by returning all the values in the loop without affecting the iteration of the loop |
| Iterator uses iter() and next() functions | Generator uses yield keyword |
| Every iterator is not a generator | Every generator is an iterator |

String to array:-

```
thislist = ["apple", "banana", "cherry"]
str1 = ' '.join(thislist)
counter = dict.fromkeys(str1, 0)
print(counter)
for item in str1:
    counter[item] += 1
print(counter)

import collections
print(collections.Counter("hello"))
```

# Use of "get()" function

### Ques. What is NumPy?

- NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

### Ques. Why is NumPy Faster Than Lists?

### Ques. What is difference betweenr repr() and str()?

the built-in str() and repr() functions both produce a textual representation of an object.

- **str():-** The str() function returns a user-friendly description of an object.
- **repr():-** The repr() method returns a developer-friendly string representation of an object.

```
import datetime
today = datetime.datetime.now()
print(str(today))
print(repr(today))

Output:-
2021-08-14 08:18:25.138663
datetime.datetime(2021, 8, 14, 8, 18, 25, 138663)
```

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

banana = Fruit("Banana")
print(banana)

Output:- <__main__.Fruit object at 0x7f97c77704c0>
-----------------------------------------------------
class Fruit:
    def __init__(self, name):
        self.name = name

    def __str__(self):
        return f'I am a {self.name}'

banana = Fruit("Banana")
print(banana)

Output:- I am a Banana
```

## Ques. What is Scope Resolution in Python?

- scope resolution in python follows the LEGB rules.

1. Local(L): Defined inside function/class
2. Enclosed(E): Defined inside enclosing functions(Nested function concept)
3. Global(G): Defined at the uppermost level
4. Built-in(B): Reserved names in Python builtin modules

- Local Scope in Python

```
pi = 'global pi variable'
def inner():
```

```
    pi = 'inner pi variable'
    print(pi)

inner()

Output:- inner pi variable
```

- Local and Global Scopes in Python

```
pi = 'global pi variable'
def inner():
    pi = 'inner pi variable'
    print(pi)

inner()
print(pi)

Output:-
inner pi variable
global pi variable
```

- Local, Enclosed, and Global Scopes in Python

```
pi = 'global pi variable'

def outer():
    pi = 'outer pi variable'
    def inner():
        # pi = 'inner pi variable'
        nonlocal pi
        print(pi)
    inner()

outer()
print(pi)

Output:-
outer pi variable
global pi variable
```

## Oops Interview Questions

What are Python namespaces? A namespace in python refers to the name which is assigned to each object in python. The objects are variables and functions. As each object is created, its name along with space(the address of the outer function in which the object is), gets created. The namespaces are maintained in python like a dictionary where the key is the namespace and value is the address of the object. There 4 types of namespace in python-

Built-in namespace– These namespaces contain all the built-in objects in python and are available whenever python is running. Global namespace– These are namespaces for all the objects created at the level of the main program. Enclosing namespaces– These namespaces are at the higher level or outer function. Local namespaces– These namespaces are at the local or inner function.

```
a = "mohit kumar"
new_list = [1,2,3,4,3,2,1,5,6,7]
list4 = []
for item in a:
    list4.append(item)
print(list4)
list5 = []
[list5.append(item*5) for item in new_list if item not in list5]
print(list5)
dict1 = {}
# for item in List:
#   dict1[len(item)] = item
# print(dict1)
dict1 = {len(item): item  for item in List}
print(dict1)
abc = sorted(dict1)
list6 = [dict1[item] for item in abc]
print(list6)
```

f

g. WAP to count from a string? l. Count occurrence of each number in list. s. Write a custom insert() function for list eg. def custom_insert (list, index, item). bb. Can we pass list or tuple as a key in dictionary? cc. Unique dictionary from list. ee. What is property decorator? gg. Iterators and iterable difference. hh. Difference between generator and iterator in python. ii. What is lazy evaluation in python? b. Static file contains which type of file normally? c. How can be file read in specific location? d. How do you remove a file from a folder? e. If user upload excel file check file format if it is valid or invalid. f. What is context manager? g. What is Garbage Collector? g. What is the difference between Static method and Class method? h. Private variable and how we can access that? i. Inheritance and types of inheritance. j. Difference multilevel and multiple inheritance and drawback?

1. Exception Handling a. How can we handle errors in python? b. How many ways exception in python?
2. Multithreading a. What is GIL?
3. Numpy a. Tell me some python libraries .. Numpy b. Convert a list into numpy

https://pynative.com/python-constructors/

Local Scope The Variables which are defined in the function are a local scope of the variable. These variables are defined in the function body. Example:- def myfunc(): x = 300 print(x)

myfunc()

Output:- 300

Global Scope The Variable which can be read from anywhere in the program is known as a global scope. These variables can be accessed inside and outside the function. x = 300

def myfunc(): print(x)

myfunc()

print(x)

Output:- 300 300

NonLocal or Enclosing Scope Built-in Scope Global Keyword

AbstractUser vs AbstractBaseUser The default User model in Django uses a username to uniquely identify a user during authentication. If you'd rather use an email address, you'll need to create a custom User model by either subclassing AbstractUser or AbstractBaseUser.

Options:

AbstractUser: Use this option if you are happy with the existing fields on the User model and just want to remove the username field. AbstractBaseUser: Use this option if you want to start from scratch by creating your own, completely new User model.

https://books.agiliq.com/projects/django-orm-cookbook/en/latest/null_vs_blank.html