Python theory Questions and Answers

1. How memory management works in python?

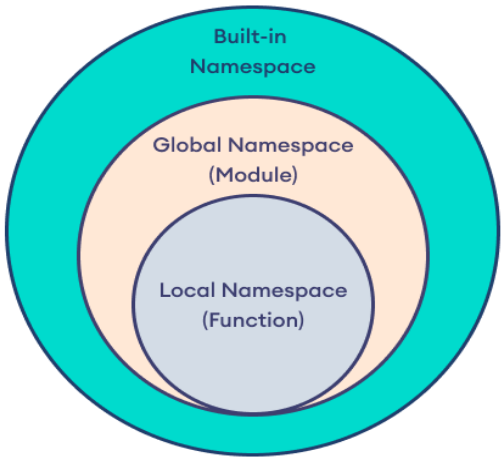
Python memory management allocates, deallocates, and tracks memory usage for objects in a private heap using references counting and a garbage collector.

2. What are namespaces in python?

In python, a namespace is a mapping between names and objects.

Python has several types of namespaces, including:

1. **Local namespace:** The local namespace contains the names that are defined inside a function or class.
2. **Global namespace:** The global namespace contains the names defined at the top level of a module or script, as well as any names imported from other modules.
3. **Built-in namespace:** The built-in namespace contains the names of all the built-in functions and modules in python.
4. **Enclosing namespace:** The enclosing namespace occur when a function contains other functions.



When you use a name (identifier) in your python code, the interpreter searches for the name in the local namespace first. If the name is not found in the local namespace, the interpreter then searches the global namespace, and finally, the built-in namespace.

3. Difference between mutable and immutable data types?

MUTABLE	IMMUTABLE
Objects can be modified after creation.	Objects can't be modified after creation.
Memory address of the object can change after modification. **To check ID:** print(id(list1))	Memory address of the object doesn't change after creation. **To check ID:** print(id(str1))
Slower to access compared to "immutable objects"	Faster to access compared to "mutable objects".
Ex: List, Set, Dictionary.	**Ex:** Integers, Strings, float, Tuple, Boolean, Frozenset.

4. Difference between List and Tuple?

LIST	TUPLE
List is **mutable.**	Tuple is **immutable.**
List **consumes** more memory.	Tuple **consumes** less memory.
List is a group of **comma separated values** stores within **square brackets** and **square brackets [] are mandatory.**	Tuple is a group of **comma separated values** stores within **Parenthesis** and **parenthesis () are optional.**
List have **many built-in methods**.	Tuple have **less built-in methods.**
If the **content is not fixed** and keep on changing, then we go for List.	If the **content is fixed** and never changes then we should go for Tuple.
List objects can't use as keys for dictionaries because, keys should be **Hashable and immutable.**	Tuple objects can be used as keys for dictionaries because, keys should be **Hashable and immutable.**

5. Difference between List and Set?

LIST	SET
List is an ordered collection of elements.	Set is an unordered collection of unique elements.
List allows duplicate values.	Set doesn't allow duplicate values.
List is mutable.	Set is mutable.
List elements accessed by index number.	Set elements can't be accessed by index number.
Uses square brackets [] to stores elements.	Uses curly brackets {} or set () constructor to stores elements.

6. Difference between Set and Dictionary?

SET	DICTIONARY
Set is mutable. So, we can update the sets.	Dictionary is mutable. So, we can update them but keys are not duplicated.
Set is created using curly brackets {}.	Dictionary is created using curly brackets {}.
Set doesn't allow duplicate values.	Dictionary doesn't allow duplicate keys.
Set is an unordered collection of non-homogeneous data.	Dictionary is an unordered collection of data in the form of key value pairs.

7. Difference between sort () and sorted ()?

Sort ()	Sorted ()
Sort () is a method of the list object in python.	Sorted () is a built-in function in python.
Sort () doesn't return anything (i.e., it returns **"None"**)	Sorted () returns a new sorted list.
Sort () only works with mutable sequences, such as lists.	Sorted () can work with any iterable, such as lists, tuples, and strings.
Sort () sorts the list in place, i.e., it modifies the original list.	Sorted () return a new sorted list and leaves the original list unchanged.

8. can, we have mutable objects in dictionary keys?

No, we cannot have mutable objects as keys in a dictionary in Python.

If you try to use a mutable object as a dictionary key, you will get a TypeError with the message "unhashable type".

9. what are all the data types we can use as a dictionary values?

In python, we can use almost any date type as values in a dictionary, including:

- Numeric types (integers, floats, complex numbers)
- Strings and bytes
- Boolean values (True, False)
- Sequences (lists, tuples)
- Sets and frozensets
- Other dictionaries
- Functions and callable objects
- Classes and instances
- Modules and Packages
- Custom objects and instances

(or)

In python, we can use almost any data type as values in a dictionary, including numbers, strings, Boolean values, lists, tuples, sets, other dictionaries, functions, classes, modules, and custom objects.

10. Difference between python 2.0 and python 3.0?

Feature	Python 2.0	Python 3.0
Print statement	Uses "print" statement without parentheses. **Syntax: print "Hello"**	Uses "print ()" function with parentheses. **Syntax: print ("Hello")**
Division operator **("/")**	Performs integer division for integers.	Always returns a float.
Unicode Handling	Stings are ASCII by default.	Strings are Unicode by default.
Range function	Returns a list.	Returns a generator object.

11. What are all the IDE's used for python scripting?

There are several Integrated Development Environments (IDEs) that can be used for python scripting:

1) PyCharm
2) Spyder
3) Visual studio code
4) Jupyter Notebook
5) IDLE (Integrated Development and Learning Environment)
6) Sublime Text

12. What is the difference between pass, continue, and break statement? Give one example?

Pass:

A Pass statement in python is a null operation that is used when the statement is required syntactically.

Ex:

```
list1 = [1,2,3,4,5,6,7]
for i in list1:
    if i==4:
        print("it is a even number!")
        pass
    else:
        print(i)
```

Continue:

A Continue statement is used to skip the current iteration and continue the next iteration inside a loop.

Ex:

```
list1 = [1,2,3,4,5,6,7]
for i in list1:
    if i==4:
        continue
    else:
        print(i)
```

Break:

A Break statement is used to terminates the loop.

Ex:

```
list1 = [1,2,3,4,5,6,7]
for i in list1:
    if i==4:
        break
    else:
        print(i)
```

13. What is function in python? why use it?

Function:

A function is a block of code, which only runs when it is called.

Functions are used to break down large programs into smaller, more manageable pieces of code that can be reused and tested independently.

(or)

A function in python is a reusable block of code that performs a specific task, and it is used to make the code more organized, modular, and easier to maintain.

14. Difference between function and method?

Function:

A function is a reusable block of code that performs a specific task in python.

Method:

A method is a function that belongs to a class and can access and modify class attributes.

(or)

A method is a function that is associated with an object or a class and can access and modify its attributes.

15. What are Local variables and Global variables? Explain with examples?

Local Variables:

Local variables are variables defined inside a function and are only accessible within the function.

Ex:

```
def display():
    x = 10        # x is a local variable
    print(x)
display()       # o/p:: 10
```

Global Variables:

Global variables are variables defined outside of any function and can be accessed and modified form anywhere in the code.

Ex:

```
x=10     # x is a global variable

def display():

    print(x)

display()     #o/p: 10
```

16. What is an Argument? Explain different types of arguments?

Argument:

Arguments are the given inputs to the function. These arguments can be optional.

Types of arguments:

There are four types of arguments in python:

1. **Positional arguments:**
   Arguments that are passed based on their position (or) order and are matched up with the parameters of the function based on their position as well.
2. **Keyword arguments:**
   Arguments that are passed with a keyword (parameter name) and its corresponding value, allowing the function to receive the arguments in any order.
3. **Default arguments:**
   Arguments that have default values assigned to them and are used when a value is not provided for that argument in the function call.
4. **Variable – length arguments / Arbitrary arguments:**
   Arguments that allow a function to accept any number of arguments. There are two types of variable-length arguments,
   ***args:** which allows a function to accept any number of positional arguments.

Note: Here **args** name is not mandatory, but * required.

****kwargs:** which allows a function to accept any number of Keyword arguments.

Note: Here **kwargs** name is not mandatory, but ** required.

17. Difference between *args and **kwargs?

***args:**

*args is used to pass a variable number of non-keyworded arguments (i.e,. positional arguments) to a function.

Ex: The arguments are passed as a tuple and can be accessed by the parameter name that precedes the,

```
def display(*args):
    print (args)
display (1,2,3,4)
```

Note: o/p is in the form of "tuple".

****kwargs:**

**kwargs is used to pass a variable number of keyworded arguments (i.e,. named arguments) to a function.

Ex: The arguments are passed as dictionary and can be accessed by the parameter name that precedes the,

```
def display(**kwargs):
    print(kwargs)
display (name = "john", age= "30")
```

Note: o/p is in the form of "dictionary".

18. What is the use of return and difference between return and print?

Use of return:

In python, the **return statement** is used to return a value from a function to the caller.

Print:

In python, **print () is a built-in function** that is used to output values to the console.

Return:

In python, **return is a keyword** used in functions to return a value or values back to the caller.

19. What is lambda function? Explain with one example?

Lambda function/Anonymous function:

A lambda function is an anonymous function that can be defined in a single line of code using the **lambda** keyword.

Ex:

```
add_numbers = lambda x, y: x + y

result = add_numbers (5, 7)

print(result) # Output: 12
```

20. What is map, filter, reduce explain with an example?

Map:

"map ()" is a built-in function that applies a given function to each item of an iterable (such as a list) and returns a new iterable with the results.

Ex:

➔ with lambda,

```
list1 = [1, 2, 3, 4, 5]

result = map (lambda x: x**2, list1)

print(list(result))
```

➔ without lambda,

```
def square(x):
    return x ** 2

list1 = [1, 2, 3, 4, 5]

result = map (square, list1)

print ("Original list:", list1)

print ("Squared list:", list(result))
```

Filter:

"filter ()" is a built-in function that returns an iterator containing only the elements from an iterable (such as a list) that satisfy a certain condition.

Ex:

➔ with lambda,

```
list2 = [1, 2, 3, 4, 5]

even_numbers = filter (lambda x: x % 2 == 0, list2)

print(list(even_numbers))
```

➔ without lambda,

```
list2 = [1, 2, 3, 4, 5]

def is_even(list2):
```

```
    if list2 % 2 == 0:
        return True
    else:
        return False
even_numbers = filter (is_even, list2)
print(list(even_numbers))
```

Reduce:

"reduce ()" is a function in the python **functools** module that applies a specified function to the elements of an iterable (such as a list) in a cumulative way to produce a single value.

Ex:

➔ with lambda

```
from functools import reduce
list1 = [1, 2, 3, 4, 5]
product_of_numbers = reduce (lambda x, y: x * y, list1)
print(product_of_numbers)
```

➔ without lambda

```
from functools import reduce
list1 = [1, 2, 3, 4, 5]
def add_numbers (x, y):
    return x + y
sum_of_numbers = reduce (add_numbers, list1)
print(sum_of_numbers)
```

21. What is list and dictionary comprehension? Explain 1 example?

List comprehension:

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

Examples:

Ex-1:

```
data = [i**2 for i in range (1,6)]

print(data)
```

Ex-2:

```
list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

result = [i for i in list1 if i % 2 == 0]

print(result)
```

Dictionary Comprehension:

Dictionary comprehension method is used to create efficient dictionaries in a single line of code.

Examples:

Ex-1:

```
data = {'Alice': 75, 'Bob': 50, 'Charlie': 80, 'Dave': 65, 'Eve': 90}

result = {i:j for i, j in data.items() if j >= 60}

print(result)
```

Ex-2:

```
list1 = [2, 4, 6, 8, 10]

result = {i: i**2 for i in list1}

print(result)
```

22. what is Class and Object? Explain 1 example?

Class:

A class is a **Plan** (or) **Blueprint** (or) **Model** of an **object.**

- Class can be defined as a **collection of objects.**
- Class is a **logical entity.**

Object:

Object is an **instance** of a **class.**

- Object is an entity that existing in the **real world.**
- Object is a **physical entity.**

Attributes:

Attributes means to store the data values.

(or)

Attributes are the properties that belongs to an **Object** to be created.

Methods:

Methods refer to the functionality of an **Object** to be created.

Instance:

Instance is an event when an **Object** was created out of its class.

Ex:

```
class Bag:                                          (class Name)
    def __init__(self,pen,book,pencil):          (Method-1)
        self.Pen_Name=pen                          (Public attributes)
        self. Book_Name=book                       (Public attributes)
        self._Pencil_Name=pencil                   (Private attributes)
    def test1(self):                               (Method-2)
        print("Bag")
```

result = Bag("apsara","maths”) **(**creating an "**instance”** of **Bag** called **result)**

print(result.Pen_Name) **(**creating an **"Object”** of **Bag** called **result)**

print(result.Book_Name)

23. What is Object-oriented programming (OOP) language and advantages of it?

OOP:

Object-oriented programming (OOP) is a way of structuring a program into **class, object, method, or constructor.**

Advantages of OOPs:

1. Easy to understand.
2. Reusability
3. Through inheritance, we can eliminate code and extend the use of existing classes.
4. Models the real world well.
5. The principle of data hiding helps the programmer to build secure programs.
6. Software complexity can be easily managed.

For advantages visit link: **https://youtu.be/9qfdgORmAxE**

Visit link: **https://prepinsta.com/python/oops-concept-in-python/**

Visit link: **https://www.edureka.co/blog/object-oriented-programming-python/#:~:text=OOPs%20concepts%20in%20Python),objects%20or%20several%20mini%2Dprograms.**

24. What is Instance method, Class method, and Static method? Explain with one example?

Instance Method:

- Whenever you want to perform the operations on both static variables and instance variables then we have to define **"instance method".**
- Instance method must be defined by **"self"** as first argument.

Ex:

```
class Car:
    def __init__(self, brand, model):
        self.company = brand
        self.model = model

c1 = Car("TATA", "Nexon")
print(c1.__dict__)

c2 = Car("AUDI", "Sedan")
print(c2.__dict__)

```

(or)

```
class Car:
    def __init__(self):
        self.brand = "TATA"

    def display(self):
        self.model = "Nexon"

c1 = Car()
c1.display()
c1.year = "2017"

print(c1.brand, c1.model, c1.year)


```

(or)

```
class Car:
    def __init__(self):
        self.brand = "TATA"
        self.model = "Nexon"

c1 = Car()
print(c1.__dict__)


```

(or)

```
class Car:
    def display(self):
        self.brand = "TATA"
        self.model = "Nexon"
        self.year = "year"

c1 = Car()
c1.display()

print(c1.__dict__)


```

(or)

```
class Car:
    def display(self):
        print("Welcome to instance variables")

c1 = Car()
c1.brand = "TATA"
c1.model = "Nexon"
c1.year = "2017"

print(c1.__dict__)


```

Class Method:

- Whenever you want to perform the operations only on static variables then we have to define **"class method".**
- Class method must be defined by using predefined decorator **"@classmethod"** and **"cls"** as the first argument.

Ex:

```
class Student():
    grade = 4
    def __init__(self,name, age):
        self.name = name
        self.age = age
    def get_data(self):
        print({"name":self.name, "age":self.age, "grade":self.grade})
    @classmethod
    def update_grade(cls,grade):
        cls.grade = grade

s1 = Student("Mary", "25")
s2 = Student("Kavya", "26")

Student.update_grade(20)

s1.get_data()
s2.get_data()


```

Static Method (or) Utility Method:

- Whenever you don't want to perform any operations on static variable (or) instance variable then we have to define **"static method (or) utility method".**
- Static method must be defined by using predefined decorator **"@staticmethod"** and without **"self" (or) "cls"** as first arguments.

Ex:

```
class Student():
    grade = 4
    def __init__(self,name, age):
        self.name = name
        self.age = age
    def get_data(self):
        print({"name":self.name, "age":self.age, "grade":self.grade})
    @classmethod
    def update_grade(cls,grade):
        cls.grade = grade
    @staticmethod
    def check_age(age):
        if age>18:
            print("Above 18")
        else:
            print("Below 18")
s1 = Student("Mary", "17")
s2 = Student("Kavya", "24")
s1.update_grade(20)
s1.check_age(17)
s1.get_data()
s2.update_grade(27)
s2.check_age(26)
s2.get_data()
```

```
class MyMaths:
    @staticmethod
    def add(self,cls):  # here "self" and "cls" are formal parameters.
        print("self is:", self)
        print("cls is:", cls)
        print("sum is:", self+cls)
        # formal parameters are act as "local variables"

#calling
MyMaths.add(20,40)
```

25. what are the features of OOPs? Explain one (1) example?

Features of OOPs in python are,

- **Encapsulation**
- **Inheritance**
- **Polymorphism**
- **Abstraction**

Encapsulation:

Encapsulation means wrapping up of data and functions into a single unit.

(or)

Encapsulation is used to binding the data and functions into a single class.

(or)

Encapsulation is a programming technique that binds data and corresponding methods together into a single unit.

Inheritance:

The process of creating a new class from already existing class.

(or)

The process of creating a new class /child class /sub class/derived class from the parent class /base class /super class.

Polymorphism:

(Poly means **"Many"** and morph means **"forms")**

Implementing same method in different content.

Abstraction / Data Abstraction:

Hiding the implementation details and showing essential details.

Ex: ATM machine

26. Different types of Inheritance? With examples?

Inheritance:

Inheritance means accessing the properties of parent class in the child class is called **"inheritance".**

(or)

The process of creating a child class from the parent class.

Child class: It is a class which is ready to give resources to another class.

Parent class: It is a class which is ready to take resources from another class.

There are several types of inheritances,

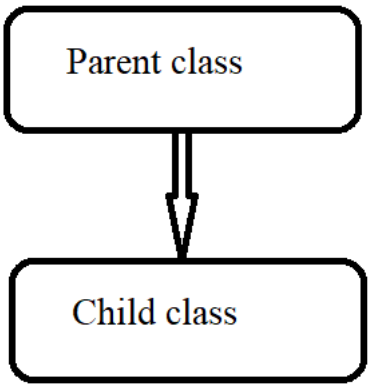
- Single inheritance
- Multiple inheritance
- Multi-level inheritance
- Hierarchical inheritance

1. Single inheritance:

Single inheritance is the process of deriving a single child class from a single parent class.

(or)

It is a process of creating only **one child class** from only **one parent class**.

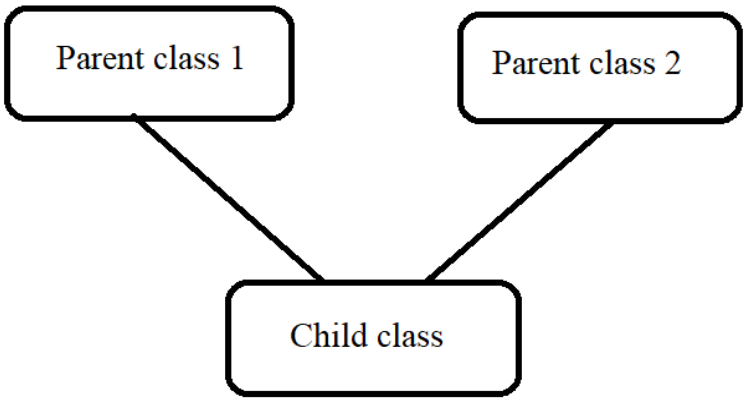


2. Multiple inheritance:

It is the process of creating a child class from more than one parent classes.

(or)

If a child class is derived from more than one parent class, then it is called "Multiple inheritance".

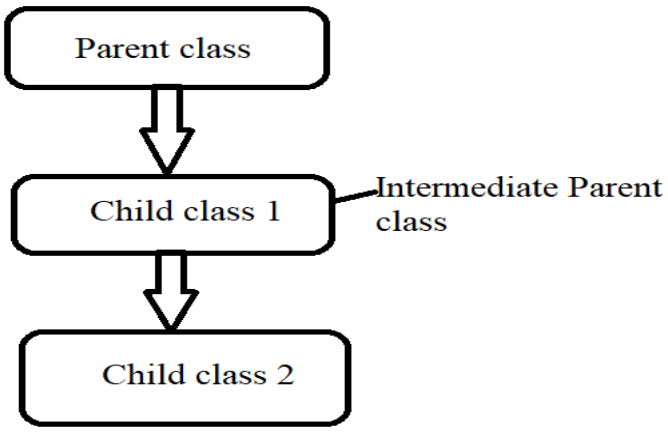


3. Multi-level inheritance:

It is the process of deriving a **child class** from another **derived child class**.

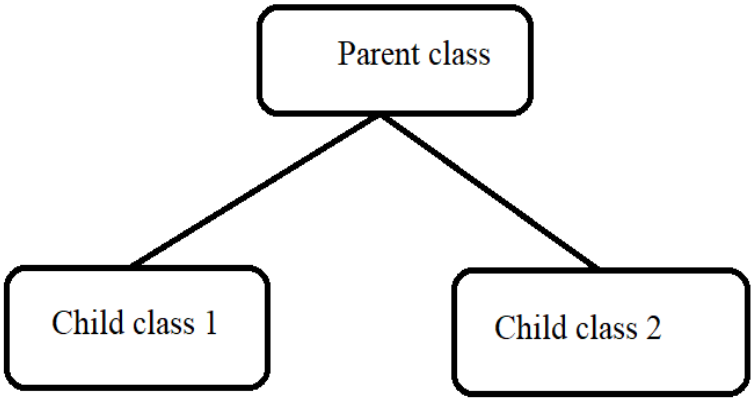
(or)

It is the process of deriving a **child class** from **already derived child class**.



4. Hierarchical inheritance:

It is the process of creating **more than one child classes** from only **one parent classes**.



27. What is Method Resolution order (MRO)?

MRO:

Method Resolution Order is the order in which the interpreter looks for and executes methods in a class hierarchy. Especially useful in python because python supports multiple inheritance.

(or)

Method Resolution Order is the order in which the interpreter looks for a method in a class hierarchy. It determines which method to call when there are multiple methods with the same name in a class hierarchy.

28. What is method overloading and method overriding?

Method overloading:

In the method overloading, methods or functions must have the same name and different signatures/parameters/arguments in a class.

- In method overloading, there is no need for more than one class.

Ex:

```
class Mathematics:
    def add(self, a, b):
        print(a + b)
    def add(self, a, b, c):
        print(a + b + c)

obj = Mathematics()
obj.add(2, 3, 4)

```

Method overriding:

In the method overriding, methods or functions must have the same name and same parameters/arguments/signatures in a parent class and child class. \

- In method overriding, there is need for at least two classes.

Ex:

```
class Employee:
    def message(self):
        print('This message is from Employee')

class Department(Employee):
    def message(self):
        print('This Department is inherited from Employee')

emp = Employee()
emp.message()
print('------------')
dept = Department()
dept.message()

```

29. What is the use of __init__?

__init__:

In Object-oriented programming, it is known as a **"constructor" or "special method".**

Use of __init__:

When we create a new object of a class, Python automatically calls the __init__() method to initialize the object's attributes.

30. What is Module in python?

Module:

In python, collection of any objects is called **"Module".**

(or)

In Python, a module is a file containing Python codes, definitions of functions and statements which can be imported and used in other Python Programs.

(or)

In python, modules are simple files with the **".py"** extension containing the Python code that can be imported inside another Python Program.

Ex:

(my_module.py) script page name. In this page,

```
1 def greet(name):
2     print("Hello, " + name + "!")
3 def add(a, b):
4     return a + b
5
6
```

(test1.py) script page name. In this page,

```
import my_module

my_module.greet("Alice")   # Output: Hello, Alice!
result = my_module.add(2, 3)
print(result)              # Output: 5


```

31. What is Package in python?

Package:

In python, collection of modules is called **"Package".**

Ex:

Let's say we have a directory called **my_package** containing two python modules, **module1.py & module2.py** and an empty file called **__init__.py**

Structure like this, Inside directory

my_module/

__init__.py

module1.py

module2.py

In **module1.py** page,

```
def foo():
    print("Hello from module1")


```

In **module2.py** page,

```
def too():
    print("Hello from module2")


```

In **check.py** page,

```
from my_package import module1, module2

module1.foo()   # Output: Hello from module1
module2.too()   # Output: Hello from module2

```

list all regex methods:

1) re.search
2) re.findall
3) re.compile
4) re.match
5) re.fullmatch
6) re.finditer
7) re.sub
8) re.subn
9) re.split
10) re.purge
11) re.escape
12) re.template