**1) What are the key features of Python?**

**Ans:** Python is an **interpreted** language. That means that, unlike languages like C and its variants, Python does not need to be compiled before it is run. Other interpreted languages include Ruby.

* Python is **dynamically typed**, this means that you don’t need to state the types of variables when you declare them or anything like that. You can do things like x=111 and then x="I'm a string" without error
* Python is well suited to **object orientated programming** in that it allows the definition of classes along with composition and inheritance. Python does not have access specifiers (like C++’s public, private).
* In Python, **functions** are **first-class objects**. This means that they can be assigned to variables, returned from other functions and passed into functions. Classes are also first class objects
* **Writing Python code is quick** but running it is often slower than compiled languages. Fortunately，Python allows the inclusion of C based extensions so bottlenecks can be optimized away and often are. The numpy package is a good example of this, it’s really quite quick because a lot of the number crunching it does isn’t actually done by Python
* Python finds **use in many spheres** – web applications, automation, scientific modeling, big data applications and many more. It’s also often used as “glue” code to get other languages and components to play nice.

**2) How is Python an interpreted language?**

**Ans:** An interpreted language is any programming language which is not in machine level code before runtime. Therefore, Python is an interpreted language.

**3) How is memory managed in Python?**

**Ans:** 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.
* 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.

**4) What is PYTHONPATH?**

**Ans:** It is an environment variable which is used when a module is imported. Whenever a module is imported, PYTHONPATH is also looked up to check for the presence of the imported modules in various directories. The interpreter uses it to determine which module to load.

**5) Is python case sensitive?**

**Ans:** Yes. Python is a case sensitive language.

**6) What is type conversion in Python?**

**Ans:** Type conversion refers to the conversion of one data type iinto another.

**Int()** – converts any data type into integer type

**float()** – converts any data type into float type

**ord()** – converts characters into integer

**hex(**) – converts integers to hexadecimal

**oct()** – converts integer to octal

**tuple() –** This function is used to convert to a tuple.

**set() –** This function returns the type after converting to set.

**list() –** This function is used to convert any data type to a list type.

**dict() –** This function is used to convert a tuple of order (key,value) into a dictionary.

**str() –** Used to convert integer into a string.

**complex(real,imag) –** This functionconverts real numbers to complex(real,imag) number.

**7) Is indentation required in python?**

**Ans:** 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.

**8) What is the difference between Python Arrays and lists?**

**Ans:** Arrays and lists, in Python, have the same way of storing data. But, arrays can hold only a single data type elements whereas lists can hold any data type elements.

**Example:**

**import** array **as** arr

My\_Array=arr.array(**'i'**,[1,2,3,4])

My\_list=[1,**'abc'**,1.20]

**print**(My\_Array)

**print**(My\_list)

**9) What are functions in Python?**

**Ans:** A function is a block of code which is executed only when it is called. To define a Python function, the **def** keyword is used.

**Example:**

**def** func():

**print**(**"Hi, Welcome to functional example"**)

func() *#calling the function*

**10) What are local variables and global variables in Python? Explain with an example.**

**Global Variables:**

Variables declared outside a function or in global space are called global variables. These variables can be accessed by any function in the program.

**Local Variables:**

Any variable declared inside a function is known as a local variable. This variable is present in the local space and not in the global space.

**Example:**

a = 2

**def** add():

b = 3

c = a+b

**print**(c)

add()

**Output:** 5

When you try to access the local variable outside the function add(), it will throw an error.

**11) What is self in Python?**

**Ans:**

Self is an instance or an object of a class. In Python, this is explicitly included as the first parameter. However, this is not the case in Java where it’s optional. It helps to differentiate between the methods and attributes of a class with local variables.

The self variable in the init method refers to the newly created object while in other methods, it refers to the object whose method was called.

**12) What are the built-in types of python?**

**Ans:**

Built-in types in Python are as follows –

* Integers
* Floating-point
* Complex numbers
* Strings
* Boolean
* Built-in functions

**13) What is a lambda function?**

**Ans:** An anonymous function is known as a lambda function. This function can have any number of parameters but, can have just one statement.

**Example:**

a = **lambda** x, y : x+y

**print**(a(5, 6))

Output: 11

**14) How does break, continue and pass work?**

**Ans:**

**break:** Allows loop termination when some condition is met and the control is transferred to the next statement.

**continue:** Allows skipping some part of a loop when some specific condition is met and the control is transferred to the beginning of the loop

**pass:** Used when you need some block of code syntactically, but you want to skip its execution. This is basically a null operation. Nothing happens when this is executed.

**15) What is the usage of help() and dir() function in Python?**

**Ans:** help() and dir() both functions are accessible from the Python interpreter and used for viewing a consolidated dump of built-in functions.

help() function: The help() function is used to display the documentation string and also facilitates you to see the help related to modules, keywords, attributes, etc.

dir() function: The dir() function is used to display the defined symbols.

**16) What is a dictionary in Python?**

**Ans:** The built-in datatypes in Python is called dictionary. It defines one-to-one relationship between keys and values. Dictionaries contain pair of keys and their corresponding values. Dictionaries are indexed by keys.

Let’s take an example:

The following example contains some keys. Country, Capital & PM. Their corresponding values are India, Delhi and Modi respectively.

**Example:**

dict = {**'Country'**: **'India'**, **'Capital'**: **'Delhi'**, **'PM'**: **'Modi'**}

**print** dict[**'Country'**] *# output: India*

**print** dict[**'Capital'**] *# Output: Delhi*

**print** dict[**'PM'**] *# Output: Modi*

**17) What does this mean: \*args, \*\*kwargs? And why would we use it?**

**Ans:** We use \*args when we aren’t sure how many arguments are going to be passed to a function, or if we want to pass a stored list or tuple of arguments to a function. \*\*kwargs is used when we don’t know how many keyword arguments will be passed to a function, or it can be used to pass the values of a dictionary as keyword arguments. The identifiers args and kwargs are a convention, you could also use \*bob and \*\*billy but that would not be wise.

**18) Explain split(), sub(), subn() methods of “re” module in Python.**

**Ans:** To modify the strings, Python’s “re” module is providing 3 methods. They are:

* split() – uses a regex pattern to “split” a given string into a list.
* sub() – finds all substrings where the regex pattern matches and then replace them with a different string
* subn() – it is similar to sub() and also returns the new string along with the no. of replacements.

**19) How can files be deleted in Python?**

**Ans:** To delete a file in Python, you need to import the OS Module. After that, you need to use the os.remove() function.

**import** os

os.remove("xyz.txt")

**20) How to add values to a python array?**

**Ans:** Elements can be added to an array using the **append()**, **extend()** and the **insert (i,x)** functions.

**Example:**

a**=**arr.array('d', [1.1 , 2.1 ,3.1] )

a.append(3.4)

print(a)

a.extend([4.5,6.3,6.8])

print(a)

a.insert(2,3.8)

print(a)

**Output:**

array(‘d’, [1.1, 2.1, 3.1, 3.4])

array(‘d’, [1.1, 2.1, 3.1, 3.4, 4.5, 6.3, 6.8])

array(‘d’, [1.1, 2.1, 3.8, 3.1, 3.4, 4.5, 6.3, 6.8])

**21) How to remove values to a python array?**

**Ans:** Array elements can be removed using **pop()** or **remove()** method. The difference between these two functions is that the former returns the deleted value whereas the latter does not.

**Example:**

a **=** arr.array('d', [1.1, 2.2, 3.8, 3.1, 3.7, 1.2, 4.6])

print(a.pop())

print(a.pop(3))

a.remove(1.1)

print(a)

**Output:**

4.6

3.1

array(‘d’, [2.2, 3.8, 3.7, 1.2])

**22) What is the difference between deep and shallow copy?**

**Ans:** Shallow copy is used when a new instance type gets created and it keeps the values that are copied in the new instance. Shallow copy is used to copy the reference pointers just like it copies the values. These references point to the original objects and the changes made in any member of the class will also affect the original copy of it. Shallow copy allows faster execution of the program and it depends on the size of the data that is used.

Deep copy is used to store the values that are already copied. Deep copy doesn’t copy the reference pointers to the objects. It makes the reference to an object and the new object that is pointed by some other object gets stored. The changes made in the original copy won’t affect any other copy that uses the object. Deep copy makes execution of the program slower due to making certain copies for each object that is been called.

**Example:**

**import** copy

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

**print**(a)

b = a *# Shallow copy*

c = copy.copy(a) *# deep copy*

b.pop()

**print**(**"After pop\n"**)

**print**(a)

**print**(b)

**print**(c)

**Output:**

[1, 2, 3, 4, 5]

After pop

[1, 2, 3, 4]

[1, 2, 3, 4]

[1, 2, 3, 4, 5]

**23) How can you randomize the items of a list in place in Python?**

**Example:**

**from** random **import** shuffle

x = [**'Keep'**, **'The'**, **'Blue'**, **'Flag'**, **'Flying'**, **'High'**]

shuffle(x)

print(x)

**Output:**

['Flying', 'Keep', 'Blue', 'High', 'The', 'Flag'

**24) What are python iterators?**

**Ans:** Iterators are objects which can be traversed though or iterated upon.

**Example:**

# Sample built-in iterators

*# Iterating over a list*

print(**"List Iteration"**)

l = [**"python"**, **"is"**, **"best"**]

**for** i **in** l:

print(i)

*# Iterating over a tuple (immutable)*

print(**"\nTuple Iteration"**)

t = (**"python"**, **"is"**, **"best"**)

**for** i **in** t:

print(i)

*# Iterating over a String*

print(**"\nString Iteration"**)

s = **"Python"**

**for** i **in** s:

print(i)

*# Iterating over dictionary*

print(**"\nDictionary Iteration"**)

d = dict()

d[**'xyz'**] = 123

d[**'abc'**] = 345

**for** i **in** d:

print(**"%s %d"** % (i, d[i]))

**Output:**

List Iteration

python

is

best

Tuple Iteration

python

is

best

String Iteration

P

y

t

h

o

n

Dictionary Iteration

xyz 123

abc 345

**25) What are python generators?**

Ans: A generator-function is defined like a normal function, but whenever it needs to generate a value, it does so with the yield keyword rather than return. If the body of a def contains yield, the function automatically becomes a generator function.

**Example:**

# A generator function that yields 1 for first time,

# 2 second time and 3 third time

**def** simpleGeneratorFun():

**yield** 1

**yield** 2

**yield** 3

# Driver code to check above generator function

**for** value **in** simpleGeneratorFun():

**print**(value)

**Output:**

1

2

3

As another example, below is a generator for Fibonacci Numbers.

# A simple generator for Fibonacci Numbers

**def** fib(limit):

    # Initialize first two Fibonacci Numbers

    a, b **=** 0, 1

    # One by one yield next Fibonacci Number

**while** a < limit:

**yield** a

        a, b **=** b, a **+** b

# Create a generator object

x **=** fib(5)

# Iterating over the generator object using next

print(x.next()); # In Python 3, \_\_next\_\_()

**print**(x.next());

print(x.next());

**print**(x.next());

**print**(x.next());

# Iterating over the generator object using for

# in loop.

print("\nUsing for in loop")

**for** i **in** fib(5):

    print(i)

**Output:**

0

1

1

2

3

Using for in loop

0

1

1

2

3

**26) Is Python Object Oriented or Procedural?**

**An**s: Python support both **Object Oriented** and **Procedural Programming language** as it is a high level programming language designed for general purpose programming. Python are multi-paradigm, you can write programs or libraries that are largely procedural, object-oriented, or functional in all of these languages. It depends on what you mean by functional. Python does have some features of a functional language.

OOP's concepts like, Classes, Encapsulation, Polymorphism, Inheritance etc.. in Python makes it as an object oriented programming language.

In Similar way we can created procedural program through python using loops, for, while etc.. and control structure.

**27) What are Python libraries? Name a few of them.**

**Ans:** Python libraries are a collection of Python packages. Some of the majorly used python libraries are – numpy, scipy, matlpotlib, socket, paramiko etc.,

**28) How to import modules in python?**

**Ans:** Modules can be imported using the **import** keyword. You can import modules in three ways-

**import** array *#importing using the original module name*

**import** array **as** arr *# importing using an alias name*

**from** array **import** \* *#imports everything present in the array module*

**29) How are classes created in Python?**

**Ans:** Class in Python is created using the **class** keyword.

**class** Employee:

**def** \_\_init\_\_(self, name):

self.name = name

E1=Employee(**"abc"**)

print(E1.name)

**Output:** abc

**30) What is \_\_init\_\_?**

**Ans:**

\_init\_\_ is a method or constructor in Python. This method is automatically called to allocate memory when a new object/ instance of a class is created. All classes have the \_\_init\_\_ method.

Example:

**class** Employee:

**def** \_\_init\_\_(self, name, age,salary):

self.name = name

self.age = age

self.salary = 20000

e1 = Employee(**"XYZ"**, 23, 20000)

*# E1 is the instance of class Employee.*

*# \_\_init\_\_ allocates memory for E1.*

**print**(e1.name)

**print**(e1.age)

**print**(e1.salary)

Output:

XYZ

23

20000

**31) Does python support multiple inheritance?**

Ans: Multiple inheritance means that a class can be derived from more than one parent classes. Python does support multiple inheritance.

**32)** **What is Polymorphism in Python?**

Polymorphism means the ability to take multiple forms. So, for instance, if the parent class has a method named ABC then the child class also can have a method with the same name ABC having its own parameters and variables. Python allows polymorphism.

**Example:**

**class Parrot:**

**def** fly(self):

print(**"Parrot can fly"**)

**def** swim(self):

print(**"Parrot can't swim"**)

**class** Penguin:

**def** fly(self):

print(**"Penguin can't fly"**)

**def** swim(self):

print(**"Penguin can swim"**)

*# common interface*

**def** flying\_test(bird):

bird.fly()

*# instantiate objects*

blu = Parrot()

peggy = Penguin()

*# passing the object*

flying\_test(blu)

flying\_test(peggy)

**Output:**

Parrot can fly

Penguin can't fly

**33) Define encapsulation in Python?**

Ans: Encapsulation means binding the code and the data together. A Python class in an example of encapsulation.

**Example:**

**class** Car:

**def** \_\_init\_\_(self):

self.\_\_updateSoftware()

**def** drive(self):

print(**'driving'**)

**def** \_\_updateSoftware(self):

print(**'updating software'**)

redcar = Car()

redcar.drive()

*#redcar.\_\_updateSoftware() not accesible from object.*

Encapsulation prevents from accessing accidentally, but not intentionally.

The private attributes and methods are not really hidden, they’re renamed adding \_Car in the beginning of their name.

The method can actually be called using redcar.\_Car\_\_updateSoftware()

**34) How do you do data abstraction in Python?**

Ans: Data Abstraction is providing only the required details and hiding the implementation from the world. It can be achieved in Python by using interfaces and abstract classes.

**Example:**

***# Python program showing***

*# abstract base class work*

**from** abc **import** ABC, abstractmethod

**class** Animal(ABC):

**def** move(self):

**pass**

**class** Human(Animal):

**def** move(self):

print(**"I can walk and run"**)

**class** Snake(Animal):

**def** move(self):

print(**"I can crawl"**)

**class** Dog(Animal):

**def** move(self):

print(**"I can bark"**)

**class** Lion(Animal):

**def** move(self):

print(**"I can roar"**)

R = Human()

R.move()

K = Snake()

K.move()

R = Dog()

R.move()

K = Lion()

K.move()

**Output:**

I can walk and run

I can crawl

I can bark

I can roar

**35) Does python make use of access specifiers?**

Ans: Python does not deprive access to an instance variable or function. Python lays down the concept of prefixing the name of the variable, function or method with a single or double underscore to imitate the behaviour of protected and private access specifiers.

**36) How to create an empty class in Python?**

Ans: An empty class is a class that does not have any code defined within its block. It can be created using the *pass* keyword. However, you can create objects of this class outside the class itself. IN PYTHON THE PASS command does nothing when its executed. it’s a null statement.

Example:

**class** Employee:

**pass**

E1=Employee()

E1.name = **"hi"**

print(E1.name)

Output:

hi

**37) Decorators in Python**

Decorators are very powerful and useful tool in Python since it allows programmers to modify the behavior of function or class. Decorators allow us to wrap another function in order to extend the behavior of wrapped function, without permanently modifying it.

In Decorators, functions are taken as the argument into another function and then called inside the wrapper function.

Example:

**Function decorator**

*# defining a decorator*

**def** hello\_decorator(func):

*# inner1 is a Wrapper function in*

*# which the argument is called*

*# inner function can access the outer local*

*# functions like in this case "func"*

**def** inner1():

print(**"Hello, this is before function execution"**)

*# calling the actual function now*

*# inside the wrapper function.*

func()

print(**"This is after function execution"**)

**return** inner1

*# defining a function, to be called inside wrapper*

**def** function\_to\_be\_used():

print(**"This is inside the function !!"**)

*# passing 'function\_to\_be\_used' inside the*

*# decorator to control its behavior*

function\_to\_be\_used = hello\_decorator(function\_to\_be\_used)

*# calling the function*

function\_to\_be\_used()

**Output:**

Hello, this is before function execution

This is inside the function !!

This is after function execution

**Function Decorator with args:**

*# Python code to illustrate*

*# Decorators with parameters in Python*

**def** decorator(\*args, \*\*kwargs):

print(**"Inside decorator"**)

**def** inner(func):

print(**"Inside inner function"**)

print(**"I like"**, kwargs[**'like'**])

**return** func

**return** inner

@decorator(like=**"python"**)

**def** func():

print(**"Inside actual function"**)

**Output:**

Inside decorator

Inside inner function

I like python

**Class Decorator:**

**class** my\_decorator(object):

**def** \_\_init\_\_(self, f):

print(**"inside my\_decorator.\_\_init\_\_()"**)

f() *# Prove that function definition has completed*

**def** \_\_call\_\_(self):

print(**"inside my\_decorator.\_\_call\_\_()"**)

@my\_decorator

**def** aFunction():

print(**"inside aFunction()"**)

print(**"Finished decorating aFunction()"**)

aFunction()

**Output:**

inside my\_decorator.\_\_init\_\_()

inside aFunction()

Finished decorating aFunction()

inside my\_decorator.\_\_call\_\_()

Programmatic problems:

* Write a program to accept inputs from command line and use them in the script.



* Write a program to read a log/text file and find out whether particular keyword is present or not.



* Write a program to run other programming language executable from Python script.

import subprocess

subprocess.run("C:\\path\to\some\executable.exe")

or



* Write a script to automate camera feature testing on mobile device.