150+ Python Interview Questions and Answers for Freshers [Latest]

DataFlair is devoted to help Python learners become successful in their Python careers. That's why we are publishing an interesting and helpful series of Python Interview Questions and Answers. In this series, you will get 150+ Python Interview Questions and Answers in 3 different parts, that covers

- Python Interview Questions and Answers for Beginners
- Python Interview Questions and Answers for Intermediates
- Python Interview Questions and Answers for Experienced

Starting with our first part that is Python Interview Questions and Answers for freshers. I have divided this blog into different categories.

- Common Python Interview Questions and Answers
- Frequently asked Python Interview Questions and Answers
- Basic Python Programming Interview Questions and Answers
- Top Python Interview Questions and Answers
- Technical Python Interview Questions and Answers
- Python OOPS Interview Questions and Answers
- Open-ended Python Interview Questions for Beginners

Common Python Interview Questions and Answers

The reason I am sharing these interview questions is that you can revise all your basic concepts. As you are a beginner, the interviewer will surely check your understanding of Python Fundamentals. Let’s start exploring the basic Python Interview Questions and Answers –

Q.1. What are the key features of Python?

If it makes for an introductory language to programming, Python must mean something. These are its qualities:

- Interpreted
- Dynamically-typed
- Object-oriented
- Concise and simple
- Free
- Has a large community

Q.2. Differentiate between lists and tuples.

The major difference is that a list is mutable, but a tuple is immutable. Examples:

```
>>> mylist=[1,3,3]
```

```
>>> mylist[1]=2
>>> mytuple=(1,3,3)
```

```
>>> mytuple[1]=2
```

Traceback (most recent call last):

File “<pyshell#97>”, line 1, in <module>

mytuple[1]=2

TypeError: ‘tuple’ object does not support item assignment

Python Tuples vs Lists – A Detailed Comparison

Q.3. Explain the ternary operator in Python.

Unlike C++, we don’t have ?: in Python, but we have this:

[on true] if [expression] else [on false]

If the expression is True, the statement under [on true] is executed. Else, that under [on false] is executed.

Below is how you would use it:

```
>>> a,b=2,3
```

```
>>> min=a if a<b else b
```

```
>>> min
```

2

```
>>> print("Hi") if a<b else print("Bye")
```

Hi

Q.4. What are negative indices?

Let’s take a list for this.

```
>>> mylist=[0,1,2,3,4,5,6,7,8]
```

A negative index, unlike a positive one, begins searching from the right.

```
>>> mylist[-3]
```

6

This also helps with slicing from the back:

```
>>> mylist[-6:-1]
```

[3, 4, 5, 6, 7]

Q.5. Is Python case-sensitive?

A language is case-sensitive if it distinguishes between identifiers like myname and Myname. In other words, it cares about case- lowercase or uppercase. Let’s try this with Python.

```
>>> myname='Ayushi'
```

```
>>> Myname
```

Traceback (most recent call last):

File “<pyshell#3>”, line 1, in <module>

Myname

NameError: name ‘Myname’ is not defined

As you can see, this raised a NameError. This means that Python is indeed case-sensitive.

Q.6. How long can an identifier be in Python?

According to the official Python documentation, an identifier can be of any length. However, PEP 8 suggests that you should limit all lines to a maximum of 79 characters. Also, PEP 20 says ‘readability counts’. So, a very long identifier will violate PEP-8 and PEP-20.

Apart from that, there are certain rules we must follow to name one:

- It can only begin with an underscore or a character from A-Z or a-z.
- The rest of it can contain anything from the following: A-Z/a-z/_/0-9.
- Python is case-sensitive, as we discussed in the previous question.
- Keywords cannot be used as identifiers. Python has the following keywords:

and	def	False	import	not	True
as	del	finally	in	or	try
assert	elif	for	is	pass	while
break	else	from	lambda	print	with
class	except	global	None	raise	yield

continue	exec	if	nonlocal	return

Learn everything about Python Identifiers

Q.7. How would you convert a string into lowercase?

We use the lower() method for this.

```
>>> 'AyuShi'.lower()
```

‘ayushi’

To convert it into uppercase, then, we use upper().

```
>>> 'AyuShi'.upper()
```

‘AYUSHI’

Also, to check if a string is in all uppercase or all lowercase, we use the methods isupper() and islower().

```
>>> 'AyuShi'.isupper()
```

False

```
>>> 'AYUSHI'.isupper()
```

True

```
>>> 'ayushi'.islower()
```

True

```
>>> '@yu$hi'.islower()
```

True

```
>>> '@YU$HI'.isupper()
```

True

So, characters like @ and $ will suffice for both cases

Also, istitle() will tell us if a string is in title case.

```
>>> 'The Corpse Bride'.istitle()
```

True

Q.8. What is the pass statement in Python?

There may be times in our code when we haven’t decided what to do yet, but we must type something for it to be syntactically correct. In such a case, we use the pass statement.

```
>>> def func(*args):
```

```
pass
```

```
>>>
```

Similarly, the break statement breaks out of a loop.

```
>>> for i in range(7):

if i==3: break

print(i)
```

1

2

Finally, the continue statement skips to the next iteration.

```
>>> for i in range(7):

if i==3: continue

print(i)
```

1

2

4

5

6

Hope you have read all the basic Python Interview Questions and Answers. Now, let's move towards the second part of the blog – Most asked Python Interview Questions and Answers for freshers

Frequently Asked Python Interview Questions and Answers for Freshers

While solving or answering these questions, if you feel any difficulty, comment us. DataFlair is always ready to help you.

Q.9. Explain help() and dir() functions in Python.

The help() function displays the documentation string and help for its argument.

```
>>> import copy
```

```
>>> help(copy.copy)
```

Help on function copy in module copy:

copy(x)

Shallow copy operation on arbitrary Python objects.

See the module's __doc__ string for more info.

The dir() function displays all the members of an object(any kind).

```
>>> dir(copy.copy)
```

[‘__annotations__’, ‘__call__’, ‘__class__’, ‘__closure__’, ‘__code__’, ‘__defaults__’, ‘__delattr__’, ‘__dict__’, ‘__dir__’, ‘__doc__’, ‘__eq__’, ‘__format__’, ‘__ge__’, ‘__get__’, ‘__getattribute__’, ‘__globals__’, ‘__gt__’, ‘__hash__’, ‘__init__’, ‘__init_subclass__’, ‘__kwdefaults__’, ‘__le__’, ‘__lt__’, ‘__module__’, ‘__name__’, ‘__ne__’, ‘__new__’, ‘__qualname__’, ‘__reduce__’, ‘__reduce_ex__’, ‘__repr__’, ‘__setattr__’, ‘__sizeof__’, ‘__str__’, ‘__subclasshook__’]

Q.10. How do you get a list of all the keys in a dictionary?

Be specific in these type of Python Interview Questions and Answers.

For this, we use the function keys().

```
>>> mydict={'a':1,'b':2,'c':3,'e':5}
```

```
>>> mydict.keys()
```

dict_keys([‘a’, ‘b’, ‘c’, ‘e’])

Wait!! Have you developed any Python Project yet?

Practice with Top Python Projects with source code and become job-ready

Q.11. What is slicing?

Slicing is a technique that allows us to retrieve only a part of a list, tuple, or string. For this, we use the slicing operator [].

```
>>> (1,2,3,4,5)[2:4]
```

(3, 4)

```
>>> [7,6,8,5,9][2:]
```

[8, 5, 9]

```
>>> 'Hello'[:-1]
```

‘Hell’

Q.12. How would you declare a comment in Python?

Unlike languages like C++, Python does not have multiline comments. All it has is octothorpe (#). Anything following a hash is considered a comment, and the interpreter ignores it.

```
>>> #line 1 of comment
```

```
>>> #line 2 of comment
```

In fact, you can place a comment anywhere in your code. You can use it to explain your code.

Q.13. How will you check if all characters in a string are alphanumeric?

For this, we use the method isalnum().

Q.14. How will you capitalize the first letter of a string?

Simply using the method capitalize().

```
>>> 'ayushi'.capitalize()
```

‘Ayushi’

```
>>> type(str.capitalize)
```

<class ‘method_descriptor’>

However, it will let other characters be.

```
>>> '@yushi'.capitalize()
```

‘@yushi’

```
>>> 'Ayushi123'.isalnum()
```

True

```
>>> 'Ayushi123!'.isalnum()
```

False

Other methods that we have include:

```
>>> '123.3'.isdigit()
```

False

```
>>> '123'.isnumeric()
```

True

```
>>> 'ayushi'.islower()
```

True

```
>>> 'Ayushi'.isupper()
```

False

```
>>> 'Ayushi'.istitle()
```

True

```
>>> ' '.isspace()
```

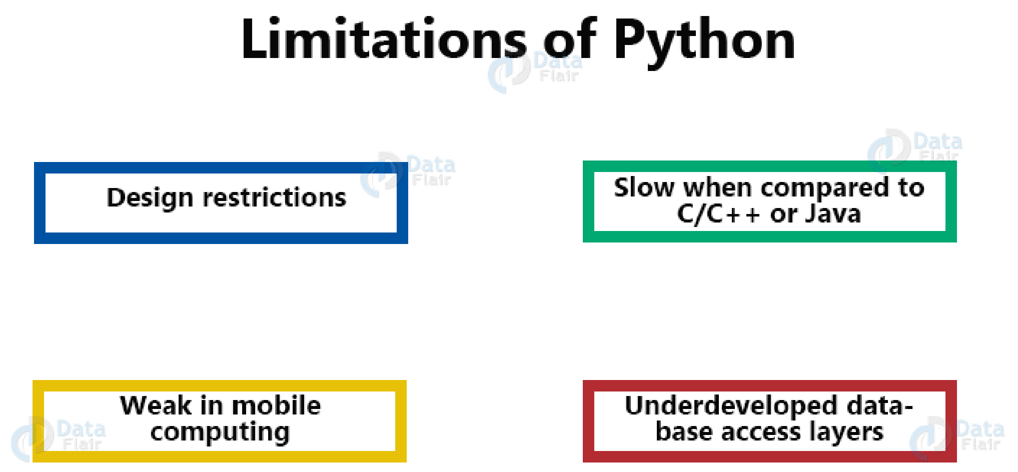
True

```
>>> '123F'.isdecimal()
```

False

Q.15. We know Python is all the rage these days. But to be truly accepting of a great technology, you must know its pitfalls as well. Would you like to talk about this?

Of course. To be truly yourself, you must be accepting of your flaws. Only then can you move forward to work on them. Python has its flaws too:


Limitations of Python
Design restrictions
Slow when compared to C/C++ or Java
Weak in mobile computing
Underdeveloped data-base access layers

- Python's interpreted nature imposes a speed penalty on it.
- While Python is great for a lot of things, it is weak in mobile computing, and in browsers.
- Being dynamically-typed, Python uses duck-typing (If it looks like a duck, it must be a duck). This can raise runtime errors.
- Python has underdeveloped database access layers. This renders it a less-than-perfect choice for huge database applications.
- And then, well, of course. Being easy makes it addictive. Once a Python-coder, always a Python coder.

Q.16. With Python, how do you find out which directory you are currently in?

To find this, we use the function/method getcwd(). We import it from the module os.

```
>>> import os
```

```
>>> os.getcwd()
```

‘C:\\Users\\lifei\\AppData\\Local\\Programs\\Python\\Python36-32’

```
>>> type(os.getcwd)
```

<class ‘builtin_function_or_method’>

We can also change the current working directory with chdir().

```
>>> os.chdir('C:\\Users\\lifei\\Desktop')
```

```
>>> os.getcwd()
```

‘C:\\Users\\lifei\\Desktop’

Q.17. How do you insert an object at a given index in Python?

Let’s build a list first.

```
>>> a=[1,2,4]
```

Now, we use the method insert. The first argument is the index at which to insert, the second is the value to insert.

```
>>> a.insert(2,3)
```

```
>>> a
```

[1, 2, 3, 4]

Q.18. And how do you reverse a list?

Using the reverse() method.

```
>>> a.reverse()
```

```
>>> a
```

[4, 3, 2, 1]

You can also do it via slicing from right to left:

```
>>> a[::-1]
```

```
>>> a
```

[1, 2, 3, 4]

This gives us the original list because we already reversed it once. However, this does not modify the original list to reverse it.

Q.19. What is the Python interpreter prompt?

This is the following sign for *Python Interpreter*:

```
>>>
```

If you have worked with the IDLE, you will see this prompt.

Q.20. How does a function return values?

A function uses the ‘return’ keyword to return a value. Take a look:

```
>>> def add(a,b):
```

```
return a+b
```

Q.21. How would you define a block in Python?

For any kind of statements, we possibly need to define a block of code under them. However, Python does not support curly braces. This means we must end such statements with colons and then indent the blocks under those with the same amount.

```
>>> if 3>1:
```

```
print("Hello")
```

```
print("Goodbye")
```

Hello

Goodbye

Q.22. Why do we need break and continue in Python?

Both break and continue are statements that control flow in *Python loops*. break stops the current loop from executing further and transfers the control to the next block. continue jumps to the next iteration of the loop without exhausting it.

Q.23. Will the do-while loop work if you don’t end it with a semicolon?

Trick question! Python does not support an intrinsic do-while loop. Secondly, to terminate do-while loops is a necessity for languages like C++.

Q.24. In one line, show us how you’ll get the max alphabetical character from a string.

For this, we’ll simply use the max function.

```
>>> max('flyiNg')
```

‘y’

The following are the ASCII values for all the letters of this string-

f- 102

l- 108

y- 121

i- 105

N- 78

g- 103

By this logic, try to explain the following line of code-

```
>>> max('fly{}iNg')
```

‘}’

(Bonus: } – 125)

Q.25. What is Python good for?

Python is a jack of many trades, check out Applications of Python to find out more.

Meanwhile, we’ll say we can use it for:

- Web and Internet Development
- Desktop GUI
- Scientific and Numeric Applications
- Software Development Applications
- Applications in Education
- Applications in Business
- Database Access
- Network Programming
- Games, 3D Graphics
- Other Python Applications

Q.26. Can you name ten built-in functions in Python and explain each in brief?

Ten Built-in Functions, you say? Okay, here you go.

complex()- Creates a complex number.

```
>>> complex(3.5,4)
```

(3.5+4j)

eval()- Parses a string as an expression.

```
>>> eval('print(max(22,22.0)-min(2,3))')
```

20

filter()- Filters in items for which the condition is true.

```
>>> list(filter(lambda x:x%2==0,[1,2,0,False]))
```

[2, 0, False]

format()- Lets us format a string.

```
>>> print("a={0} but b={1}".format(a,b))
```

a=2 but b=3

hash()- Returns the hash value of an object.

```
>>> hash(3.7)
```

644245917

hex()- Converts an integer to a hexadecimal.

```
>>> hex(14)
```

‘0xe’

input()- Reads and returns a line of string.

```
>>> input('Enter a number')
```

Enter a number7

```
'7'
```

len()- Returns the length of an object.

```
>>> len('Ayushi')
```

6

locals()- Returns a dictionary of the current local symbol table.

```
>>> locals()
```

{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <class '_frozen_importlib.BuiltinImporter'>, '__spec__':

None, ‘__annotations__’: {}, ‘__builtins__’: <module ‘builtins’ (built-in)>, ‘a’: 2, ‘b’: 3}

open()- Opens a file.

```
>>> file=open('tabs.txt')
```

Q.27. What will the following code output?

```
>>> word=’abcdefghij’
```

```
>>> word[:3]+word[3:]
```

The output is ‘abcdefghij’. The first slice gives us ‘abc’, the next gives us ‘defghij’.

Q.28. How will you convert a list into a string?

We will use the join() method for this.

```
>>> nums=['one','two','three','four','five','six','seven']
```

```
>>> s=' '.join(nums)
```

```
>>> s
```

‘one two three four five six seven’

Q.29. How will you remove a duplicate element from a list?

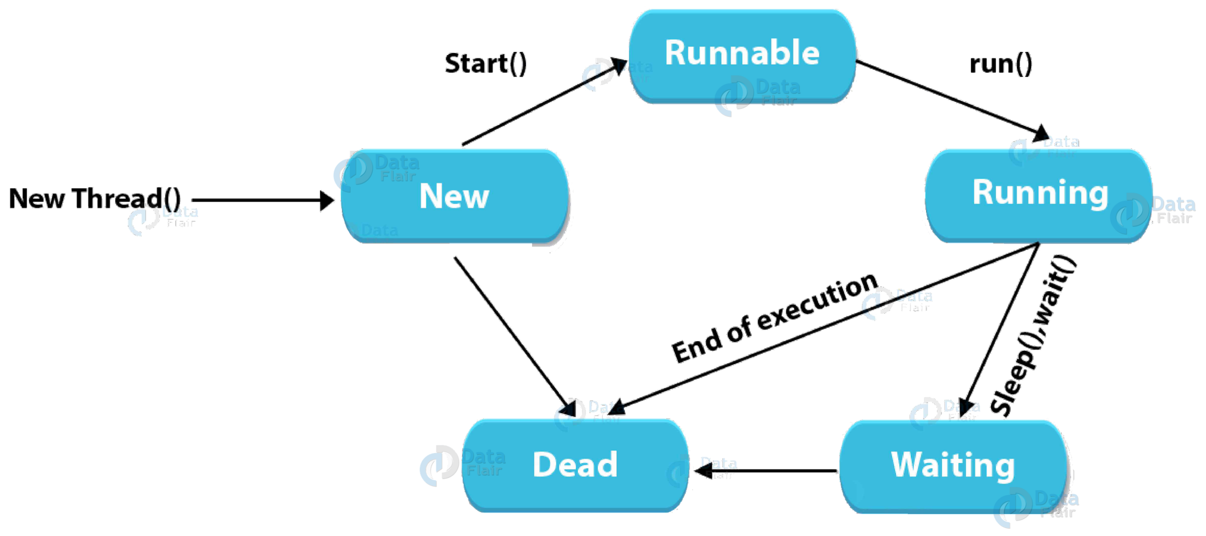
We can turn it into a set to do that.

```
>>> list=[1,2,1,3,4,2]
```

```
>>> set(list)
```

{1, 2, 3, 4}

Q.30. Can you explain the life cycle of a thread?



- To create a thread, we create a class that we make override the run method of the thread class. Then, we instantiate it.
- A thread that we just created is in the new state. When we make a call to start() on it, it forwards the threads for scheduling. These are in the ready state.
- When execution begins, the thread is in the running state.
- Calls to methods like sleep() and join() make a thread wait. Such a thread is in the waiting/blocked state.
- When a thread is done waiting or executing, other waiting threads are sent for scheduling.
- A running thread that is done executing terminates and is in the dead state.

Basic Python Program Interview Questions and Answers

Q.31. What is a dictionary in Python?

A *python dictionary* is something I have never seen in other languages like C++ or Java programming. It holds key-value pairs.

```
>>> roots={25:5,16:4,9:3,4:2,1:1}
>>> type(roots)
```

<class ‘dict’>

```
>>> roots[9]
```

3

A dictionary is mutable, and we can also use a comprehension to create it.

```
>>> roots={x**2😡 for x in range(5,0,-1)}
>>> roots
```

{25: 5, 16: 4, 9: 3, 4: 2, 1: 1}

Q.32. Explain the //, %, and ** operators in Python.

The // operator performs floor division. It will return the integer part of the result on division.

```
>>> 7//2
```

3

Normal division would return 3.5 here.

Similarly, ** performs exponentiation. a**b returns the value of a raised to the power b.

```
>>> 2**10
```

1024

Finally, % is for modulus. This gives us the value left after the highest achievable division.

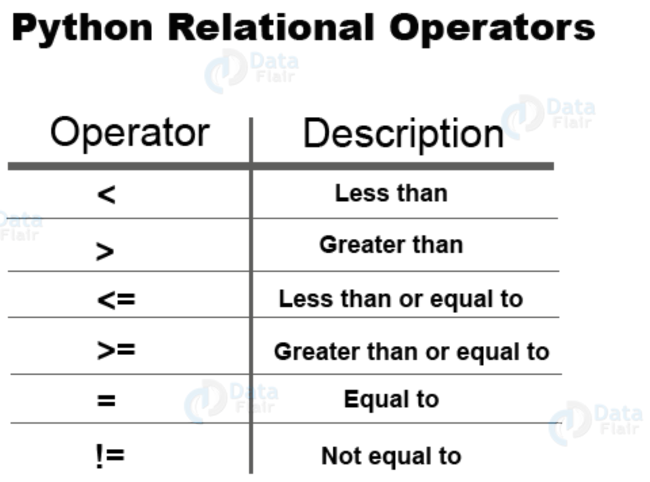
```
>>> 13%7
```

6

```
>>> 3.5%1.5
```

0.5

Q.33. What do you know about relational operators in Python.



Python Relational Operators

Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
=	Equal to
!=	Not equal to

Relational operators compare values.

Less than (<) If the value on the left is lesser, it returns True.

```
>>> 'hi'<'Hi'
```

False
Greater than (>) If the value on the left is greater, it returns True.

```
>>> 1.1+2.2>3.3
```

True

This is because of the flawed floating-point arithmetic in Python, due to hardware dependencies.

Less than or equal to (<=) If the value on the left is lesser than or equal to, it returns True.

```
>>> 3.0<=3
```

True

Greater than or equal to (>=) If the value on the left is greater than or equal to, it returns True.

```
>>> True>=False
```

True

Equal to (==) If the two values are equal, it returns True.

```
>>> {1,3,2,2}=={1,2,3}
```

True

Not equal to (!=) If the two values are unequal, it returns True.

```
>>> True!=0.1
```

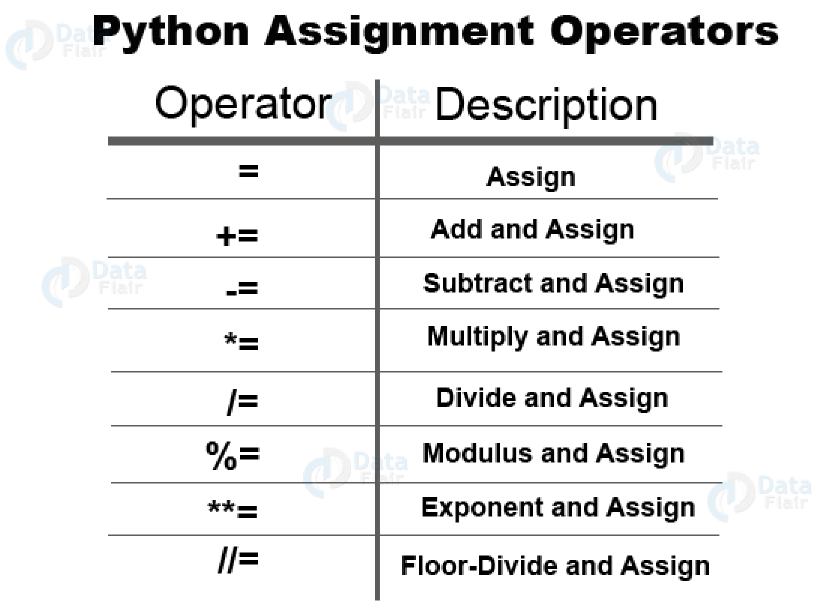
True

```
>>> False!=0.1
```

True

You will surely face a question from Python Operators. There are chances that question may be in an indirect way. Prepare yourself for it with the best guide – Python Operators

Q.34. What are assignment operators in Python?



Python Assignment Operators

Operator	Description
=	Assign
+=	Add and Assign
-=	Subtract and Assign
*=	Multiply and Assign
/=	Divide and Assign
%=	Modulus and Assign
**=	Exponent and Assign
//=	Floor-Divide and Assign

We can combine all arithmetic operators with the assignment symbol.

```
>>> a=7

>>> a+=1

>>> a

8

>>> a-=1

>>> a

7

>>> a*=2

>>> a

14

>>> a/=2
>>> a
```

7.0

```
>>> a**=2
```

```
>>> a
```

49.0

```
>>> a//=3
```

```
>>> a
```

16.0

```
>>> a%=4
```

```
>>> a
```

0.0

Q.35. Explain logical operators in Python.

We have three logical operators- and, or, not.

```
>>> False and True
```

False

```
>>> 7<7 or True
```

True

```
>>> not 2==2
```

False

Q.36. What are membership operators?

With the operators ‘in’ and ‘not in’, we can confirm if a value is a member in another.

```
>>> 'me' in 'disappointment'
```

True

```
>>> 'us' not in 'disappointment'
```

True

Q.37. Explain identity operators in Python.

The operators ‘is’ and ‘is not’ tell us if two values have the same identity.

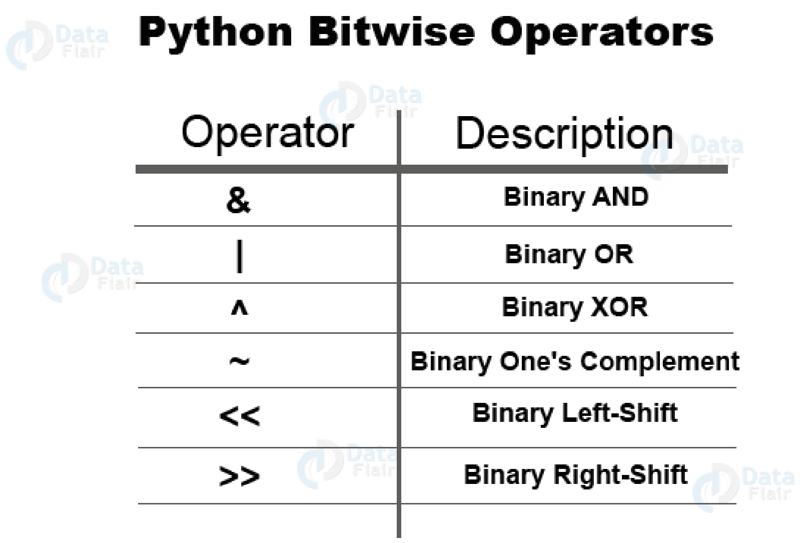
```
>>> 10 is '10'
```

False

```
>>> True is not False
```

True

Q.38. Finally, tell us about bitwise operators in Python.



Python Bitwise Operators

Operator	Description
&	Binary AND
|	Binary OR
^	Binary XOR
~	Binary One's Complement
<<	Binary Left-Shift
>>	Binary Right-Shift

These operate on values bit by bit.

AND (&) This performs & on each bit pair.

```
>>> 0b110 & 0b010
```

2

OR (|) This performs | on each bit pair.

```
>>> 3|2
```

3

XOR (^) This performs an exclusive-OR operation on each bit pair.

```
>>> 3^2
```

1

Binary One’s Complement (~) This returns the one’s complement of a value.

```
>>> ~2
```

-3

Binary Left-Shift (<<) This shifts the bits to the left by the specified amount.

```
>>> 1<<2
```

4

Here, 001 was shifted to the left by two places to get 100, which is binary for 4.

Binary Right-Shift (>>)

```
>>> 4>>2
```

1

Q.39. What data types does Python support?

Python provides us with five kinds of data types:

Numbers – Numbers use to hold numerical values.

```
>>> a=7.0

>>>
```

Strings – A string is a sequence of characters. We declare it using single or double quotes.

```
>>> title="Ayushi's Book"
```

Lists – A list is an ordered collection of values, and we declare it using square brackets.

```
>>> colors=['red','green','blue']
```

```
>>> type(colors)
```

<class 'list'>

Tuples – A tuple, like a list, is an ordered collection of values. The difference. However, is that a tuple is immutable. This means that we cannot change a value in it.

```
>>> name=('Ayushi','Sharma')
```

```
>>> name[0]='Avery'
```

```
Traceback (most recent call last):

File “<pyshell#129>”, line 1, in <module>

name[0]=’Avery’

TypeError: ‘tuple’ object does not support item assignment
```

Dictionary – A dictionary is a data structure that holds key-value pairs. We declare it using curly braces.

```
>>> squares={1:1,2:4,3:9,4:16,5:25}

>>> type(squares)

<class ‘dict’>

>>> type({})

<class ‘dict’>
```

We can also use a dictionary comprehension:

```
>>> squares={x:x**2 for x in range(1,6)}
```

```
>>> squares
```

{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

Don’t miss the complete guide for Python Data Types and Variables

Q.40. What is a docstring?

A docstring is a documentation string that we use to explain what a construct does. We place it as the first thing under a function, class, or a method, to describe what it does. We declare a docstring using three sets of single or double-quotes.

```
>>> def sayhi():
```

```
"""
```

```
The function prints Hi

"""

print("Hi")

>>> sayhi()
```

Hi

To get a function's docstring, we use its __doc__ attribute.

```
>>> sayhi.__doc__
```

'\n\tThis function prints Hi\n\t'

A docstring, unlike a comment, is retained at runtime.

Q.41. How would you convert a string into an int in Python?

If a string contains only numerical characters, you can convert it into an integer using the int() function.

```
>>> int('227')
```

227

Let's check the types:

```
>>> type('227')
```

<class 'str'>

```
>>> type(int('227'))
```

<class 'int'>

Q.42. How do you take input in Python?

For taking input from the user, we have the function input(). In Python 2, we had another function raw_input().

The input() function takes, as an argument, the text to be displayed for the task:

```
>>> a=input('Enter a number')
```

Enter a number7

But if you have paid attention, you know that it takes input in the form of a string.

```
>>> type(a)
```

<class ‘str’>

Multiplying this by 2 gives us this:

```
>>> a*=2
>>> a
```

’77’

So, what if we need to work on an integer instead?

We use the int() function for this.

```
>>> a=int(input('Enter a number'))
```

Enter a number7

Now when we multiply it by 2, we get this:

```
>>> a*=2
```

```
>>> a
```

14

Q.43. What is a function?

When we want to execute a sequence of statements, we can give it a name. Let's define a function to take two numbers and return the greater number.

```
>>> def greater(a,b):
```

```
return a is a>b else b
```

```
>>> greater(3,3.5)
```

3.5

Q.44. What is recursion?

When a function makes a call to itself, it is termed *recursion*. But then, in order for it to avoid forming an infinite loop, we must have a base condition.

Let's take an example.

```
>>> def facto(n):
```

```
if n==1: return 1
```

```
return n*facto(n-1)
```

```
>>> facto(4)
```

24

Q.45. What does the function zip() do?

One of the less common functions with beginners, zip() returns an iterator of tuples.

```
>>> list(zip(['a','b','c'],[1,2,3]))
```

[(‘a’, 1), (‘b’, 2), (‘c’, 3)]

Here, it pairs items from the two lists and creates tuples with those. But it doesn’t have to be lists.

```
>>> list(zip(('a','b','c'),(1,2,3)))
```

[(‘a’, 1), (‘b’, 2), (‘c’, 3)]

Q.46. How do you calculate the length of a string?

This is simple. We call the function len() on the string we want to calculate the length of.

```
>>> len('Ayushi Sharma')
```

13

Q.47. Explain Python List Comprehension.

The *list comprehension in python* is a way to declare a list in one line of code. Let’s take a look at one such example.

```
>>> [i for i in range(1,11,2)]
```

[1, 3, 5, 7, 9]

```
>>> [i*2 for i in range(1,11,2)]
```

[2, 6, 10, 14, 18]

Q.48. How do you get all values from a Python dictionary?

We saw previously, to get all keys from a dictionary, we make a call to the keys() method. Similarly, for values, we use the method values().

```
>>> 'd' in {'a':1,'b':2,'c':3,'d':4}.values()
```

False

```
>>> 4 in {'a':1,'b':2,'c':3,'d':4}.values()
```

True

Q.49. What if you want to toggle case for a Python string?

We have the swapcase() method from the str class to do just that.

```
>>> 'AyuShi'.swapcase()
```

‘aYUsHI’

Let’s apply some concepts now, shall we? Questions 50 through 52 assume the string ‘I love Python’. You need to do the needful.

Q.50. Write code to print only upto the letter t.

```
>>> i=0

>>> while s[i]!='t':

print(s[i],end=’’)

i+=1
```

I love Py

Q.51. Write code to print everything in the string except the spaces.

```
>>> for i in s:

if i==' ': continue

print(i,end='')
```

IlovePython

Q.52. Now, print this string five times in a row.

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

print(s)
```

I love Python

I love Python

I love Python

I love Python

I love Python

I love Python

Okay, moving on to more domains to conquer.

Q.53. What is the purpose of bytes() in Python?

bytes() is a built-in function in Python that returns an immutable bytes object. Let’s take an example.

```
>>> bytes([2,4,8])
```

b’\x02\x04\x08′

```
>>> bytes(5)
```

```
b’\x00\x00\x00\x00\x00′
```

```
>>> bytes('world','utf-8')
```

```
b’world’
```

Q.54. What is a control flow statement?

A Python program usually starts to execute from the first line. From there, it moves through each statement just once and as soon as it’s done with the last statement, it transactions the program. However, sometimes, we may want to take a more twisted path through the code. Control flow statements let us disturb the normal execution flow of a program and bend it to our will.

Q.55. Create a new list to convert the following list of number strings to a list of numbers.

```
nums=[‘22’,’68’,’110’,’89’,’31’,’12’]
```

We will use the int() function with a list comprehension to convert these strings into integers and put them in a list.

```
>>> [int(i) for i in nums]
```

[22, 68, 110, 89, 31, 12]

Q.56. Given the first and last names of all employees in your firm, what data type will you use to store it?

I can use a dictionary to store that. It would be something like this-

{‘first_name’:’Ayushi’,’second_name’:’Sharma’

Top Python Interview Questions and Answers

Q.57. How would you work with numbers other than those in the decimal number system?

With Python, it is possible to type numbers in binary, octal, and hexadecimal.

Binary numbers are made of 0 and 1. To type in binary, we use the prefix 0b or 0B.

```
>>> int(0b1010)
```

10

To convert a number into its binary form, we use bin().

```
>>> bin(0xf)
```

‘0b1111’

Octal numbers may have digits from 0 to 7. We use the prefix 0o or 0O.

```
>>> oct(8)
```

‘0o10’

Hexadecimal numbers may have digits from 0 to 15. We use the prefix 0x or 0X.

```
>>> hex(16)
```

'0x10'

```
>>> hex(15)
```

'0xf'

DataFlair's latest article on [Python Numbers]() with Examples

Q.58. What does the following code output?

```
>>> def extendList(val, list=[]):

list.append(val)

return list
```

```
>>> list1 = extendList(10)
```

```
>>> list2 = extendList(123,[])
```

```
>>> list3 = extendList('a')
```

```
>>> list1,list2,list3
```

([10, ‘a’], [123], [10, ‘a’])

You’d expect the output to be something like this:

([10],[123],[‘a’])

Well, this is because the list argument does not initialize to its default value ([]) every time we make a call to the function. Once we define the function, it creates a new list. Then, whenever we call it again without a list argument, it uses the same list. This is because it calculates the expressions in the default arguments when we define the function, not when we call it.

Q.59. How many arguments can the range() function take?

The range() function in Python can take up to 3 arguments. Let’s see this one by one.

a. One argument

When we pass only one argument, it takes it as the stop value. Here, the start value is 0, and the step value is +1.

```
>>> list(range(5))
```

[0, 1, 2, 3, 4]

```
>>> list(range(-5))
```

[]

```
>>> list(range(0))
```

[]

b. Two arguments

When we pass two arguments, the first one is the start value, and the second is the stop value.

```
>>> list(range(2,7))
```

[2, 3, 4, 5, 6]

```
>>> list(range(7,2))
```

[]

```
>>> list(range(-3,4))
```

[-3, -2, -1, 0, 1, 2, 3]

c. Three arguments

Here, the first argument is the start value, the second is the stop value, and the third is the step value.

```
>>> list(range(2,9,2))
```

[2, 4, 6, 8]

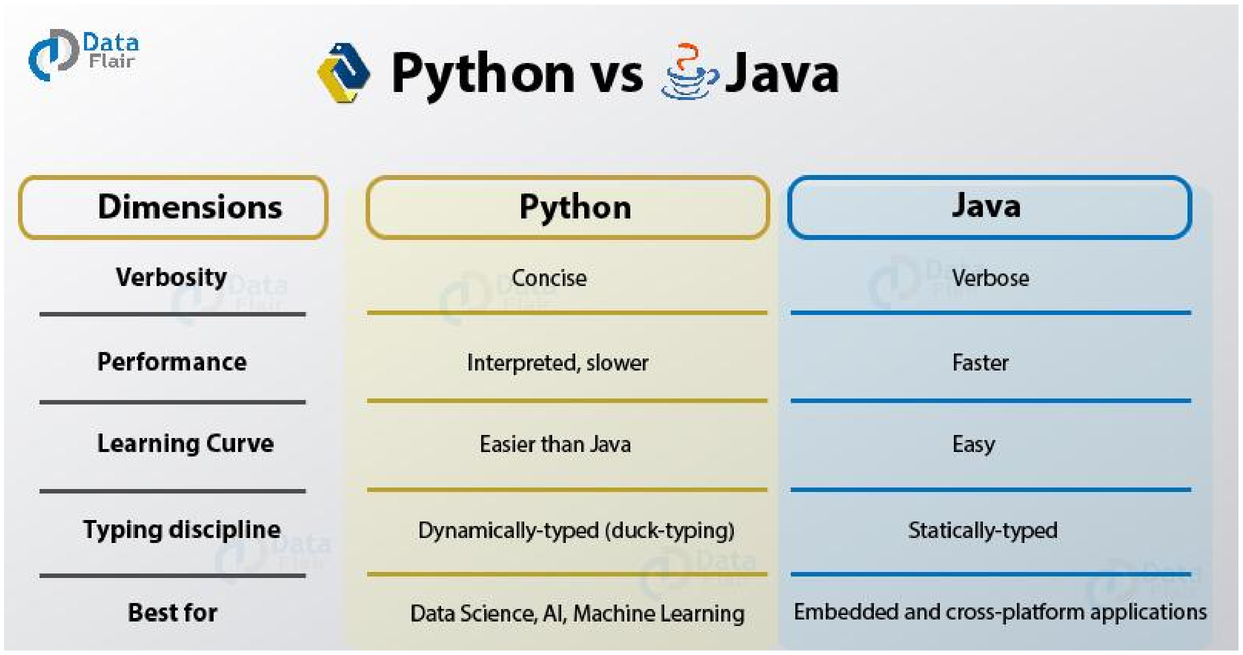
```
>>> list(range(9,2,-1))
```

[9, 8, 7, 6, 5, 4, 3]

Q.60. What is PEP 8?

PEP 8 is a coding convention that lets us write more readable code. In other words, it is a set of recommendations.

Q.61. How is Python different from Java?


Python vs Java

Dimensions	Python	Java
Verbosity	Concise	Verbose
Performance	Interpreted, slower	Faster
Learning Curve	Easier than Java	Easy
Typing discipline	Dynamically-typed (duck-typing)	Statically-typed
Best for	Data Science, AI, Machine Learning	Embedded and cross-platform applications

Following is the comparison of Python vs Java –

- Java is faster than Python
- Python mandates indentation. Java needs braces.
- Python is dynamically-typed; Java is statically typed.
- Python is simple and concise; Java is verbose
- Python is interpreted
- Java is platform-independent
- Java has stronger database-access with JDBC

Python vs Java, the most commonly asked python interview question for freshers.

Learn it in detail – Python vs Java for Interview

Q.62. What is the best code you can write to swap two numbers?

I can perform the swapping in one statement.

```
>>> a,b=b,a
```

Here’s the entire code, though-

```
>>> a,b=2,3
```

```
>>> a,b=b,a
```

```
>>> a,b
```

(3, 2)

Q.63. How can you declare multiple assignments in one statement?

This is one of the most asked interview questions for Python freshers –

There are two ways to do this:

First –

```
>>> a,b,c=3,4,5 #This assigns 3, 4, and 5 to a, b, and c respectively
```

Second –

```
>>> a=b=c=3 #This assigns 3 to a, b, and c
```

Q.64. If you are ever stuck in an infinite loop, how will you break out of it?

For this, we press Ctrl+C. This interrupts the execution. Let's create an infinite loop to demonstrate this.

```
>>> def counterfunc(n):
```

```
while(n==7):print(n)
```

```
>>> counterfunc(7)
```

7

7

7

7

7

7

7

7

7

7

```
7

7

7

7

7

7

7

Traceback (most recent call last):

File "<pyshell#332>", line 1, in <module>

counterfunc(7)
```

```
File "<pyshell#331>", line 2, in counterfunc

while(n==7):print(n)

KeyboardInterrupt
```

Technical Python Interview Questions and Answers

Q.65. How do we execute Python?

Python files first compile to bytecode. Then, the host executes them.

Revise the concept of [Python Compiler]()

Q.66. Explain Python's parameter-passing mechanism.

To pass its parameters to a function, Python uses pass-by-reference. If you change a parameter within a function, the change reflects in the calling function. This is its default behavior. However, when we pass literal arguments like strings, numbers, or tuples, they pass by value. This is because they are immutable.

Q.67. What is the with statement in Python?

The with statement in Python ensures that cleanup code is executed when working with unmanaged resources by encapsulating common preparation and cleanup tasks. It may be used to open a file, do something, and then automatically close the file at the end. It may be used to open a database connection, do some processing, then automatically close the connection to ensure resources are closed and available for others. with will cleanup the resources even if an exception is thrown. This statement is like the using statement in C#.
Consider you put some code in a try block, then in the finally block, you close any resources used. The with statement is like syntactic sugar for that.

The syntax of this control-flow structure is:

```
with expression [as variable]:
....with-block
```

```
>>> with open('data.txt') as data:

#processing statements
```

Q.68. How is a .pyc file different from a .py file?

While both files hold bytecode, .pyc is the compiled version of a Python file. It has platform-independent bytecode. Hence, we can execute it on any platform that supports the .pyc format. Python automatically generates it to improve performance(in terms of load time, not speed).

Python OOPS Interview Questions and Answers

Q.69. What makes Python object-oriented?

Again the frequently asked Python Interview Question

Python is object-oriented because it follows the Object-Oriented programming paradigm. This is a paradigm that revolves around classes and their instances (objects). With this kind of programming, we have the following features:

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism
- Data hiding

Q.70. How many types of objects does Python support?

Objects in Python are mutable and immutable. Let’s talk about these.

Immutable objects- Those which do not let us modify their contents. Examples of these will be tuples, booleans, strings, integers, floats, and complexes. Iterations on such objects are faster.

```
>>> tuple=(1,2,4)
```

```
>>> tuple
```

(1, 2, 4)

```
>>> 2+4j
```

(2+4j)

Mutable objects – Those that let you modify their contents. Examples of these are lists, sets, and dicts. Iterations on such objects are slower.

```
>>> [2,4,9]
```

[2, 4, 9]

```
>>> dict1={1:1,2:2}
```

```
>>> dict1
```

{1: 1, 2: 2}

While two equal immutable objects' reference variables share the same address, it is possible to create two mutable objects with the same content.

Open-ended Python Interview Questions

Q.71 Why do you want to work for this company?

Q.72 Where do you see yourself in 10 years?

Q.73 What will you bring to the table if we hire you?

Q.74 Tell me about your best personal project. What challenges did you face, and how did it change the way you work?

Q.75 Would you have a problem with menial tasks?

Q.76 What makes you like Python over other languages? (The most commonly asked Python interview questions)

So, these were some of the important Python Interview Questions and Answers. If you practiced all the above questions then you are ready to move towards the next part of DataFlair's *Python Interview Questions and Answers Series – Part 2* for Python Interviews. I advise you not to miss a single part of this series. All these questions are specifically designed by experienced individuals to provide you with complete help for cracking your next interview.