

# Python Interview Questions

## Python test

Top 100 Python Interview Questions & Answers For 2021 | Edureka

is the most sought-after skill in programming domain. In this Python Certification Python Interview Questions blog, I will introduce you to the most frequently asked questions in Python interviews for the year 2021.

[e! https://www.edureka.co/blog/interview-questions/python-interview-questions/](https://www.edureka.co/blog/interview-questions/python-interview-questions/)



What does if `__name__ == "__main__":` do?

Whenever the Python interpreter reads a source file, it does two things: Let's see how this works and how it relates to your question about the `__name__` checks we always see in Python scripts. Let's

[https://stackoverflow.com/questions/419163/what-does-if-name\\_\\_main\\_\\_-do](https://stackoverflow.com/questions/419163/what-does-if-name__main__-do)



## Python Interview Questions

### Python Program to Calculate the Length of a String Without Using a Library Function

```
string=raw_input("Enter string:")
count=0
for i in string:
    count=count+1
print("Length of the string is:")
print(count)
```

**Write a function called FooBar that takes input integer n and prints all the numbers from 1 up to n in a new line. If the number is divisible by 3 then print "Foo", if the number is divisible by 5 then print "Bar" and if the number is divisible by both 3 and 5, print "FooBar". Otherwise, just print the number.**

```

values = ((3, "Foo"), (5, "Bar"))
for n in range(1, 101):
    res = ''.join(v for (k, v) in values if not n % k)
    print(res if res else n)

```

## Find smallest subset prefixes

```

minimal = []
words = ['foo', 'foog', 'food', 'asdf']
words.sort(key=lambda x: (len(x), x))
while words:
    word = words[0]
    minimal.append(word)
    words = [ x for x in words[1:] if not x.startswith(word) ]
print (minimal)

```

## 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))

```

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

- Basically \*args is applied when you are not sure about the number of arguments that are going to be passed to a function, or if there arises a need to pass a stored list or tuple of arguments to a function.

- Whereas `**kwargs` is used when you are unsure about the number of keyword arguments that will be passed to a function, or it can be also used to pass the values of a dictionary as keyword arguments. The identifier `args` and `kwargs` are a convention, you could also use `*bob` and `**billy` but that would not be wise.

### **What do you understand by the term Deep copy and Shallow copy?**

- To store the values, which have already copied, a deep copy is used. The reference pointers to the object are not copied by the deep copy.
- It simply helps in making reference to the object and the new object that is pointed by some other object gets stored. The changes are made in the original copy that will not affect any other copy while using the object. The Deep copy makes the execution of the program slower due to making certain copies for each object that is been called.
- 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.

For immutable objects, there is no need for copying because the data will never change, so Python uses the same data; ids are always the same. For mutable objects, since they can potentially change, `[shallow]` copy creates a new object.

**Deep copy is related to nested structures. If you have a list of lists, then `deepcopy` `copies` the nested lists also, so it is a recursive copy. With just `copy`, you have a new outer list, but inner lists are references.**

Assignment does not copy. It simply sets the reference to the old data. So you need `copy` to create a new list with the same contents.

Normal assignment operations will simply point the new variable towards the existing object. The [docs](#) explain the difference between shallow and deep copies:

The difference between shallow and deep copying is only relevant for compound objects (objects that contain other objects, like lists or class instances): A shallow copy constructs a new compound

object and then (to the extent possible) inserts references into it to the objects found in the original. A deep copy constructs a new compound object and then, recursively, inserts copies into it of the objects found in the original.

Here's a little demonstration:

```
import copy

a = [1, 2, 3]
b = [4, 5, 6]
c = [a, b]
```

Using normal assignment operations to copy:

```
d = c

print id(c) == id(d)      # True - d is the same object as c
print id(c[0]) == id(d[0]) # True - d[0] is the same object as c[0]
```

Using a shallow copy:

```
d = copy.copy(c)

print id(c) == id(d)      # False - d is now a new object
print id(c[0]) == id(d[0]) # True - d[0] is the same object as c[0]
```

Using a deep copy:

```
d = copy.deepcopy(c)

print id(c) == id(d)      # False - d is now a new object
print id(c[0]) == id(d[0]) # False - d[0] is now a new object
```

### What is the use of 'self' keyword?

- In python 'self' references the instance of the class.

# Decorators

- Decorators are a significant part of Python. In simple words: they are functions which modify the functionality of other functions.

The shortest explanation that I can give is that decorators wrap your function in another function that returns a function.

This code, for example:

```
@decorate
def foo(a):
    print a
```

would be equivalent to this code if you remove the decorator syntax:

```
def bar(a):
    print a

foo = decorate(bar)
```

Decorators sometimes take parameters, which are passed to the dynamically generated functions to alter their output.

## 7.5.1. Use-cases:

Now let's take a look at the areas where decorators really shine and their usage makes something really easy to manage.

## 7.5.2. Authorization

Decorators can help to check whether someone is authorized to use an endpoint in a web application. They are extensively used in Flask web framework and Django. Here is an example to employ decorator based authentication:

**Example :**

```
from functools import wraps

def requires_auth(f):
```

```

@wraps(f)
def decorated(*args, **kwargs):
    auth = request.authorization
    if not auth or not check_auth(auth.username, auth.password):
        authenticate()
    return f(*args, **kwargs)
return decorated

```

### 7.5.3. Logging

Logging is another area where the decorators shine. Here is an example:

```

from functools import wraps

def logit(func):
    @wraps(func)
    def with_logging(*args, **kwargs):
        print(func.__name__ + " was called")
        return func(*args, **kwargs)
    return with_logging

@logit
def addition_func(x):
    """Do some math."""
    return x + x

result = addition_func(4)
# Output: addition_func was called

```

### How is the memory managed in Python?

- Python memory 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 and the interpreter takes care of this Python private heap.
- The allocation of Python heap space for Python objects is done by Python memory manager. The core API gives access to some tools for the programmer to code.
- Python also has an inbuilt **garbage collector**, which recycle all the unused memory and frees the memory and makes it available to the heap space.

### What is pickling and unpickling?

**Ans:** Pickle module accepts any Python object and converts it into a string representation and dumps it into a file by using dump function, this process is called

pickling.

While the process of retrieving original Python objects from the stored string representation is called unpickling.

- The pickle module implements a fundamental, but powerful algorithm for serializing and de-serializing a Python object structure.
- **Pickling** - is the process whereby a Python object hierarchy is converted into a byte stream, and **Unpickling** - is the inverse operation, whereby a byte stream is converted back into an object hierarchy.

Pickling (and unpickling) is alternatively known as **serialization**, **marshalling**, or **flattening**.

```
import pickle

data1 = {'a': [1, 2.0, 3, 4+6j],
        'b': ('string', u'Unicode string'),
        'c': None}

selfref_list = [1, 2, 3]
selfref_list.append(selfref_list)

output = open('data.pkl', 'wb')

# Pickle dictionary using protocol 0.
pickle.dump(data1, output)

# Pickle the list using the highest protocol available.
pickle.dump(selfref_list, output, -1)

output.close()
```

To read from a pickled file -

```
import pprint, pickle

pkl_file = open('data.pkl', 'rb')

data1 = pickle.load(pkl_file)
pprint.pprint(data1)

data2 = pickle.load(pkl_file)
pprint.pprint(data2)

pkl_file.close()
```

Short for Pretty Printer, **pprint** is a native Python library that allows you to customize the formatting of your output.

## 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.

1. 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.
2. Dir() function: The dir() function is used to display the defined symbols.

## 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.

## 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)
```

## What is the difference between Python's list methods append and extend?

**append**: Appends object at the end.



```
x = [1, 2, 3]
x.append([4, 5])
print (x)
```

gives you: `[1, 2, 3, [4, 5]]`

extend: Extends list by appending elements from the iterable.

```
x = [1, 2, 3]
x.extend([4, 5])
print (x)
```

gives you: `[1, 2, 3, 4, 5]`

## What is monkey patching in Python?

In Python, the term monkey patch only refers to dynamic modifications of a class or module at run-time.

Consider the below example:

```
# m.py
class MyClass:
    def f(self):
        print "f()"

import m
def monkey_f(self):
    print "monkey_f()"

m.MyClass.f = monkey_f
obj = m.MyClass()
obj.f()

#The output will be as below:

monkey_f()
```

## 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, unlike Java.

## What is Polymorphism in Python?

**Ans:** 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.

### **Define encapsulation in Python?**

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

### **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.

### **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 behavior of protected and private access specifiers.

### **Matplotlib vs Bokeh**

matplotlib is for basic plotting -- bars, pies, lines, scatter plots, etc. Bokeh is for interactive visualization

if your data is so complex (or you haven't yet found the "message" in your data), then use Bokeh to create interactive visualizations that will allow your viewers to explore the data themselves.

### **How is memory managed in Python?**

**Ans:** Memory is managed in Python in the following ways:

1. 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.
2. 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.

3. 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.

## Q7. What is namespace in Python?

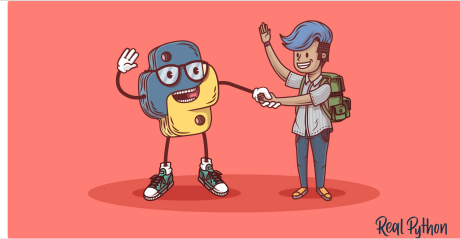
**Ans:** A namespace is a naming system used to make sure that names are unique to avoid naming conflicts.

global, local, scope of variable and functions

### Namespaces and Scope in Python - Real Python



In this tutorial, you'll learn about Python namespaces, the structures used to store and organize the symbolic names created during execution of a Python program. You'll learn when namespaces are

 <https://realpython.com/python-namespaces-scope/>



## What are 'signals'?


Django consists of a signal dispatcher that helps allow decoupled applications to get notified when actions occur elsewhere in the framework. Django provides a set of built-in signals that basically allow senders to notify a set of receivers when some action is executed. Some of the signals are as follows:

 Signal	 Description
<u><a href="#">django.db.models.signals.pre_save</a></u> <u><a href="#">django.db.models.signals.post_save</a></u>	Sent before or after a model's save() method is called
<u><a href="#">django.db.models.signals.pre_deleted</a></u> <u><a href="#">django.db.models.signals.post_delete</a></u>	Sent before or after a model's delete() method or queryset's delete() method is called
<u><a href="#">django.db.models.signals.m2m_changed</a></u>	Sent when Django starts or finishes an HTTP request

# Create an empty dataframe

## Create an empty data.frame

If you want to declare such a data.frame with many columns, it'll probably be a pain to type all the column classes out by hand.

 <https://stackoverflow.com/questions/10689055/create-an-empty-data-frame/31193730#31193730>



If you **already have an existent data frame**, let's say `df` that has the columns you want, then you can just create an empty data frame by removing all the rows:

```
empty_df = df[FALSE,]
```


Notice that `df` still contains the data, but `empty_df` doesn't.

I found this question looking for how to create a new instance with empty rows, so I think it might be helpful for some people.

# Dictionary with duplicate keys

## python3 dictionary with duplicate keys but different values


Thanks for contributing an answer to Stack Overflow! Please be sure to answer the question. Provide details and share your research! Asking for help, clarification, or responding to other answers. Making

 <https://stackoverflow.com/questions/49587961/python3-dictionary-with-duplicate-keys-but-different-values>



## Adding duplicate keys to JSON with python

Asked Is there a way to add duplicate keys to json with python? From my understanding, you can't have duplicate keys in python dictionaries. Usually, how I go about creating json is to create the

 <https://stackoverflow.com/questions/29519858/adding-duplicate-keys-to-json-with-python>



```
$> python -c "import json; print json.dumps({1: 'a', '1': 'b'})"
{"1": "b", "1": "a"}
```

Python dicts have unique keys. There is no getting around that.

One approach might be to make a `defaultdict` of lists in Python followed by jinja `for` loops in the form code to iterate the values of the dict.

Alternatively, if you are able to send a `json` string, [this workaround for handling duplicate keys](#) may help:

### Given

```
data = [
    ("evt", "2001"),
    ("evt", "1024001"),
    ("src", "mstrWeb.my.fbb.fb.2001"),
    ("src", "mstrWeb.my.fbb.1024001")
]
```

### Code

```
class Container(dict):
    """Overload the items method to retain duplicate keys."""

    def __init__(self, items):
        self[""] = ""
        self._items = items

    def items(self):
        return self._items

json.dumps(Container(data))
# '{"evt": "2001", "evt": "1024001", "src": "mstrWeb.my.fbb.fb.2001", "src": "mstrWeb.my.fbb.1024001"}'
```