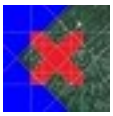

CampusX Mentorship Program

Python interview questions & answers for lecture 1, lecture 2 and lecture 3:

Question No.	Question
1	What is Python? What are the benefits of using Python?
2	What is a dynamically typed language?
3	What is an Interpreted language?
4	What is PEP 8 and why is it important?
5	What are the common built-in data types in Python?
6	Explain the ternary operator in Python.
7	What Does the 'is' Operator Do?
8	Disadvantages of Python.
9	How strings are stored in Python?
10	What is Zen of Python?
11	Identity operator (is) vs ==?
12	What does _ variables represent in Python?
13	Modules vs packages vs Library
14	Why 0.3 – 0.2 is not equal to 0.1 in Python?
15	Python Docstrings

1. What is Python? What are the benefits of using Python?

Python is a high-level, interpreted, general-purpose programming language. Being a general-purpose language, it can be used to build almost any type of application with the right tools/libraries. Additionally, python supports objects, modules, threads, exception-handling, and automatic memory management which help in modelling real-world problems and building applications to solve these problems.

Benefits of using Python:

- Python is a general-purpose programming language that has a simple, easy-to-learn syntax that emphasizes readability and therefore reduces the cost of program maintenance. Moreover, the language is capable of scripting, is completely open-source, and supports third-party packages encouraging modularity and code reuse.
- Its high-level data structures, combined with dynamic typing and dynamic binding, attract a huge community of developers for Rapid Application Development and deployment.

`2. What is a dynamically typed language?`

Before we understand a dynamically typed language, we should learn about what typing is. **Typing** refers to type-checking in programming languages. In a **strongly-typed** language, such as Python, **"1" + 2** will result in a type error since these languages don't allow for "type-coercion" (implicit conversion of data types). On the other hand, a **weakly-typed** language, such as Javascript, will simply output **"12"** as result.

Type-checking can be done at two stages:

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

Read more - https://www.techtarget.com/whatis/definition/strongly-typed#:~:text=In%20computer%20programming%2C%20a%20programming,types%20of%20objects%20and%20variables.

`3. What is an Interpreted language?`

An Interpreted language executes its statements line by line. Languages such as Python, Javascript, R, PHP, and Ruby are prime examples of Interpreted languages. Programs written in an interpreted language runs directly from the source code, with no intermediary compilation step.

`4. What is PEP 8 and why is it important?`

PEP stands for **Python Enhancement Proposal**. A PEP is an official design document providing information to the Python community, or describing a new feature for Python or its processes. **PEP 8** is especially important since it documents the style guidelines for Python Code. Apparently contributing to the Python open-source community requires you to follow these style guidelines sincerely and strictly.

Read more - https://realpython.com/python-pep8/#:~:text=PEP%208%2C%20sometimes%20spelled%20PEP8,and%20consistency%20of%20Python%20code.

`5. What are the common built-in data types in Python?`

There are several built-in data types in Python. Although, Python doesn't require data types to be defined explicitly during variable declarations type errors are likely to occur if the knowledge of data types and their compatibility with each other are neglected. Python provides `type()` and `isinstance()` functions to check the type of these variables. These data types can be grouped into the following categories-

1. **None Type: None** keywork represents the null values in Python. Boolean equality operation can be performed using these NoneType objects.

2. **Numeric Type:** There are three distinct numeric types - `integers`, `floating-point numbers` and `complex numbers`. Additionally, `booleans` are a sub-type of integers.

3. **Sequence Types:** According to Python Docs, there are three basic Sequence Types - `lists`, `tuples`, and `range` objects. Sequence types have the `in` and `not in` operators defined for their traversing their elements. These operators share the same priority as the comparison operations.

4. **Mapping Types:** A mapping object can map hashable values to random objects in Python. Mappings objects are mutable and there is currently only one standard mapping type, the `dictionary`.

5. **Set Types:** Currently, Python has two built-in set types - `set` and `frozenset`. `set` type is mutable and supports methods like *add()* and *remove()*. `frozenset` type is immutable and can't be modified after creation.

6. **Callable Types:** Callable types are the types to which function call can be applied. They can be **user-defined functions, instance methods, generator functions**, and some other **built-in functions, methods** and **classes**. Refer to the documentation at docs.python.org for a detailed view of the callable types.

`###Q.6. Operator Precedence.`
https://www.programiz.com/python-programming/precedence-associativity

`###Q.7. 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
print(min)
```

Above will print 2.

```
# Run this cell to see result.
a,b=14,12
print("Hi") if a<b else print("Bye")
```

```
Bye
```

Q 8. What Does the ‘is’ Operator Do?

Identity operators In Python, is and is not are used to check if two values are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

```
a = 1
id(a)
```

```
11126688
```

```
a = 2
id(a)
```

```
11126720
```

```
a = 1
b = 1

a is b
```

```
True
```

```
id(a)
```

```
11126688
```

```
id(b)
```

```
11126688
```

```
a = 257
b = 257

a is b
```

```
False
```

```
id(a)
```

```
140582907232784
```

```
a == b
```

```
True
```

```
id(b)
```

```
140582907232464
```

```
# -5 to 256

a = -14
b = -14
```

```
a is b
```

```
False
```

`Q 9: Disadvantages of Python.`

https://www.geeksforgeeks.org/disadvantages-of-python

`Q10 How strings are stored in Python?`

- https://stackoverflow.com/questions/19224059/how-strings-are-stored-in-python-memory-model
- https://www.quora.com/How-are-strings-stored-internally-in-Python-3
- https://betterprogramming.pub/an-interviewers-favorite-question-how-are-python-strings-stored-in-internal-memory-ac0eaef9d9c2

`Q10 What is Zen of Python?`

The Zen of Python is a collection of 19 "guiding principles" for writing computer programs that influence the design of the Python programming language.
https://en.wikipedia.org/wiki/Zen_of_Python

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- Errors should never pass silently.
- Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.
- There should be one-- and preferably only one --obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than *right* now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea -- let's do more of those!

Explained in detail here - https://inventwithpython.com/blog/2018/08/17/the-zen-of-python-explained/

`Q11 Identity operator (is) vs ==?`

->> Here’s the main difference between python “==” vs “is:”

Identity operators: The “is” and “is not” keywords are called identity operators that compare objects based on their identity. Equality operator: The “==” and “!=” are called equality operators that compare the objects based on their values.

```
# Case 4:
# Here variable s is assigned a list,
# and q assigned a list values same as s but on slicing of list a new
list is generated
s=[1,2,3]
p=s
# cloning
q=s[:]
print("id of p", id(p))
print("Id of s", id(s))
print("id of q", id(q))
print("Comapare- s == q", s==q)
print("Identity- s is q", s is q)
print("Identity- s is p", s is p)
print("Comapare- s == p", s==p)
```

```
id of p 140582906466864
Id of s 140582906466864
id of q 140582906466944
Comapare- s == q True
Identity- s is q False
Identity- s is p True
Comapare- s == p True
```

```
a = [1,2,3]
b = a[:]

a.append(4)
print(a)
print(b)
```

```
[1, 2, 3, 4]
[1, 2, 3]
```

`Q12 What does _ variables represent in Python?`

GssksForGeeks Article

Underscore _ is considered as "I don't Care" or "Throwaway" variable in Python

- The underscore _ is used for ignoring the specific values. If you don’t need the specific values or the values are not used, just assign the values to underscore.

** Ignore a value when unpacking

** Ignore the index

```
# Ignore a value when unpacking
x, _, y = (1, 2, 3)

print("x-",x)
print("y-", y)
```

```
x- 1
y- 3
_ 2
```

```
#Ignore the index

# Say we want to print hello 5 times, we don't need index value

for _ in range(5):
    print('hello')
```

```
hello
hello
hello
hello
hello
```

Q13 Modules vs packages vs Library

Python uses some terms that you may not be familiar with if you're coming from a different language. Among these are modules, packages, and libraries.

- A **module** is a Python file that's intended to be imported into scripts or other modules. It often defines members like classes, functions, and variables intended to be used in other files that import it.
- A **package** is a collection of related modules that work together to provide certain functionality. These modules are contained within a folder and can be imported just like any other modules. This folder will often contain a special `__init__` file that tells Python it's a package, potentially containing more modules nested within subfolders
- A **library** is an umbrella term that loosely means "a bundle of code." These can have tens or even hundreds of individual modules that can provide a wide range of functionality. Matplotlib is a plotting library. The Python Standard Library contains hundreds of modules for performing common tasks, like sending emails or reading JSON data. What's special about the Standard Library is that it comes bundled with your installation of Python, so you can use its modules without having to download them from anywhere.

These are not strict definitions. Many people feel these terms are somewhat open to interpretation. Script and module are terms that you may hear used interchangeably.

https://stackoverflow.com/questions/19198166/whats-the-difference-between-a-module-and-a-library-in-python

https://www.geeksforgeeks.org/what-is-the-difference-between-pythons-module-package-and-library/

`Q14 Why 0.3 – 0.2 is not equal to 0.1 in Python?`

The reason behind it is called "*precision*", and it's due to the fact that computers do not compute in Decimal, but in Binary. Computers do not use a base 10 system, they use a base 2 system (also called Binary code).

https://www.geeksforgeeks.org/why-0-3-0-2-is-not-equal-to-0-1-in-python/

```
# code
print(0.3 - 0.2)
print(0.3 - 0.2 == 0.1)
```

```
0.09999999999999998
False
```

`Q 15 - Python Docstrings`

https://www.geeksforgeeks.org/python-docstrings

```
print('hello')
```

```
hello
```

```
type(3)
```

```
int
```

```
print(input.__doc__)
```

```
Forward raw_input to frontends

        Raises
        ------
        StdinNotImplentedError if active frontend doesn't support
stdin.
```

```
print(type.__doc__)
```

```
type(object_or_name, bases, dict)
type(object) -> the object's type
type(name, bases, dict) -> a new type
```

```
s = 'have'
print(id(s))
s = s.capitalize()
print(id(s))
```

```
140582962705648
140582906473392

a,b = print('hello'),print('world')

hello
world

for i in range(1, 5):
    i = 3
    print(i, end=' ')

3 3 3 3
```