iNeuron

Python

Interview Questions

Q1. How are identity operators different from membership operators?
Ans- Unlike membership operators, identity operators compare the values to find out if they have the same value or not.

Q2. What are Python decorators?
Ans- A specific change made in Python syntax to alter the functions easily is termed a Python decorator.

Q3. Differentiate between list and tuple.
Ans- Tuple is not mutable. It can be hashed e.g. key for dictionaries. On the other hand, lists are mutable.

Q4. Describe multithreading in Python.
Ans- Using Multithreading to speed up the code is not the go-to option, even though Python comes with a multi-threading package.

The package has the GIL or Global Interpreter Lock, which is a construct. It ensures that only one of the threads executes at any given time. A thread acquires the GIL and then performs work before passing it to the next thread.

This happens so fast that to a user, it seems that threads are executing in parallel. Obviously, this is not the case, as they are just taking turns while using the same CPU core. GIL passing adds to the overall overhead of the execution.

As such, if you intend to use the threading package to speed up the execution, using the package is not recommended.

Q5. Draw a comparison between the range and xrange in Python.

Ans- In terms of functionality, both range and xrange are identical. Both allow for generating a list of integers. The main difference between the two is that while range returns a Python list object, xrange returns an xrange object.

Xrange is not able to generate a static list at runtime the way range does. On the contrary, it creates values along with the requirements via a special technique called yielding. It is used with a type of object known as a generator.

If you have an enormous range for which you need to generate a list, then xrange is the function to opt for. This is especially relevant for scenarios dealing with a memory-sensitive system, such as a smartphone.

Q6. Explain Inheritance.
Ans- Inheritance enables a class to acquire all members of another class. These members can be attributes, methods, or both. By providing reusability, inheritance makes it easier to create as well as maintain an application.

The class which acquires is known as the child class or the derived class. The one that it acquires from is known as the superclass, base class, or parent class. There are 4 forms of inheritance supported by Python:

Single inheritance: A single derived class acquires members from one superclass.
Multi-Level inheritance: At least 2 different derived classes acquire members from two distinct base classes.

Hierarchical inheritance: A number of child classes acquire members from one superclass
Multiple inheritance: A derived class acquires members from several super classes.

Q7. What is the difference between deep copy and shallow copy?
Ans- We use a shallow copy when a new instance type gets created. It keeps the values that are copied in the new instance. Just like it copies the values, the shallow copy also copies the reference pointers.

Reference points copied in the shallow copy reference to the original objects. Any changes made in any member of the class affect the original copy of the same. Shallow copy enables faster execution of the program.

Deep copy is used for storing values that are already copied. Unlike shallow copy, it doesn't copy the reference pointers to the objects. Deep copy makes the reference to an object in addition to storing the new object that is pointed by some other object.

Changes made to the original copy will not affect any other copy that makes use of the referenced or stored object. Contrary to the shallow copy, deep copy makes the execution of a program slower. This is due to the fact that it makes some copies for each object that is called.

Q8. How do you distinguish between NumPy and SciPy?
Ans- Typically, NumPy contains nothing but the array data type and the most basic operations, such as basic element-wise functions, indexing, reshaping, and sorting. All the numerical code resides in SciPy.

As one of NumPy's most important goals is compatibility, the library tries to retain all features supported by either of its predecessors. Hence, NumPy contains a few linear algebra functions despite the fact that these more appropriately belong to the SciPy library.

SciPy contains fully-featured versions of the linear algebra modules available to NumPy in addition to several other numerical algorithms.

Q9. What is the output of the following code?
A0 = dict(zip(('a', 'b', 'c', 'd', 'e'),(1,2,3,4,5)))
A1 = range(10)A2 = sorted([i for i in A1 if i in A0])
A3 = sorted([A0[s] for s in A0])
A4 = [i for i in A1 if i in A3]
A5 =
A6 = [[i, i*i] for i in A1]
print(A0,A1,A2,A3,A4,A5,A6)
A0 = {'a': 1, 'c': 3, 'b': 2, 'e': 5, 'd': 4} # The order may vary

A1 = range(0, 10)

A2 = []

A3 = [1, 2, 3, 4, 5]

A4 = [1, 2, 3, 4, 5]

A5 =

A6 = [[0, 0], [1, 1], [2, 4], [3, 9], [4, 16], [5, 25], [6, 36], [7, 49], [8, 64], [9, 81]]

Q10. Explain dictionaries with an example.
A dictionary in the Python programming language is an unordered collection of data values such as a map. The dictionary holds the key: value pair. This helps define a one-to-one relationship between keys and values. Indexed by keys, a typical dictionary contains a pair of keys and corresponding values.

Let us take an example with three keys, namely website, language, and offering. Their corresponding values are hackr.io, Python, and Tutorials. The code for would be:

```
dict={‘Website’:‘hackr.io’,‘Language’:‘Python’:‘Offering’:‘Tutorials’}
print dict[Website] #Prints hackr.io
print dict[Language] #Prints Python
print dict[Offering] #Prints Tutorials
```

Q11. Python supports negative indexes. What are they and why are they used?
The sequences in Python are indexed. It consists of positive and negative numbers. Positive numbers use 0 as the first index, 1 as the second index, and so on. Hence, any index for a positive number n is n-1.

Unlike positive numbers, index numbering for negative numbers starts from -1, and it represents the last index in the sequence. Likewise, -2 represents the penultimate index. These are known as negative indexes. Negative indexes are used for:

Removing any new-line spaces from the string, thus allowing the string to except the last character, represented as S[:-1]
Showing the index to represent the string in the correct order.

Q12. What is the output of the following code?
Ans- try: if '1' != 1:
raise "someError"
else: print("someError has not occurred")
except "someError": pr
int ("someError has occurred")
The output of the program will be “invalid code.” This is because a new exception class must inherit from a Base Exception.

Q13. Explain the process of compilation and linking.
Ans- In order to compile new extensions without any error, compiling and linking is used in Python. Linking initiates only and only when the compilation is complete.

In the case of dynamic loading, the process of compilation and linking depends on the style that is provided with the concerned system. In order to provide dynamic loading of the configuration setup files and rebuild the interpreter, the Python interpreter is used.

Q14. What is Flask and what are the benefits of using it?
Ans- Flask is a web microframework for Python with Jinja2 and Werkzeug as its dependencies. As such, it has some notable advantages:

Flask has little to no dependencies on external libraries.
Because there is a little external dependency to update and fewer security bugs, the web microframework is lightweight.
It has an inbuilt development server and a fast debugger.

Q15. What is the map() function used for?
Ans- The map() function applies a given function to each item of an iterable. It then returns a list of the results. The value returned from the map() function can then be passed on to functions to the likes of the list() and set().

Typically, the given function is the first argument, and the iterable is available as the second argument to a map() function. Several tables are given if the function takes in more than one argument.

Q16. Whenever Python exits, not all the memory is deallocated. Why is it so?
Ans- Upon exiting, Python's built-in effective cleanup mechanism comes into play and tries to deallocate or destroy every other object. However, Python modules that have circular references to other objects, or the objects that are referenced from the global namespaces, aren't always deallocated or destroyed.

This is because it is not possible to deallocate those portions of the memory that are reserved by the C library.

Q17. Write a program in Python for getting indices of N maximum values in a NumPy array.

```
import numpy as np
arr = np.array([1, 3, 2, 4, 5])
print(arr.argsort()[-3:][::-1])
```

Output:
[4 3 1]

Q18. How is memory managed in Python?
Ans- Python private heap space takes the place of memory management in Python. It contains all Python objects and data structures. The interpreter is responsible to take care of this private heap, and the programmer does not have access to it. The Python memory manager is responsible for the allocation of Python heap space for Python objects. The programmer may access some tools for the code with the help of the core API. Python also provides an inbuilt garbage collector, which recycles all the unused memory and frees the memory, and makes it available to heap space.

Q19. What is the lambda function?
Ans- A lambda function is an anonymous function. This function can have only one statement but can have any number of parameters.

```
a = lambda x,y : x+y
print(a(5, 6))
```

Q20. How are arguments passed in Python? By value or by reference?
Everything in Python is an object, and all variables hold references to the object. The reference values are according to the functions; as a result, the value of the reference cannot be changed.