Interview Questions

Introduction to Python:

Python was developed by Guido van Rossum and was released first on February 20, 1991. It is one of the most widely-used and loved programming languages and is interpreted in nature thereby providing flexibility of incorporating dynamic semantics. It is also a free and open-source language with very simple and clean syntax. This makes it easy for developers to learn python. Python also supports object-oriented programming and is most commonly used to perform general-purpose programming

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

3. What is an Interpreted language?

An Interpreted language executes its statements line by line.

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.

5. What are lists and tuples? What is the key difference between the two?

Lists and Tuples are both sequence data types that can store a collection of objects in Python. The objects stored in both sequences can have different data types. Lists are represented with square brackets `['sara', 6, 0.19]`, while tuples are represented with parantheses `('ansh', 5, 0.97)`.

But what is the real difference between the two? The key difference between the two is that while lists are mutable, tuples on the other hand are immutable objects. This means that lists can be modified, appended or sliced on the go but tuples remain constant and cannot be modified in any manner.

What is break, continue and pass in Python?

Break	The break statement terminates the loop immediately and the control flows to the statement after the body of the loop.
Continue	The continue statement terminates the current iteration of the statement, skips the rest of the code in the current iteration and the control flows to the next iteration of the loop.
Pass	As explained above, the pass keyword in Python is generally used to fill up empty blocks and is similar to an empty statement represented by a semi-colon in languages such as Java, C++, Javascript, etc.

```
pat = [1, 3, 2, 1, 2, 3, 1, 0, 1, 3]
for p in pat:
   pass
   if (p == 0):
       current = p
       break
   elif (p % 2 == 0):
       continue
   print(p)    # output => 1 3 1 3 1
print(current)    # output => 0
```

What are decorators in Python?

Decorators in Python are essentially functions that add functionality to an existing function in Python without changing the structure of the function itself. They are represented the `@decorator_name` in Python and are called in a bottom-up fashion. For example

```
# decorator function to capitalize names
def names_decorator(function):
   def wrapper(arg1, arg2):
       arg1 = arg1.capitalize()
       arg2 = arg2.capitalize()
       string_hello = function(arg1, arg2)
       return string_hello
   return wrapper
@names_decorator
def say_hello(name1, name2):
   return 'Hello ' + name1 + '! Hello ' + name2 + '!'
say_hello('sara', 'ansh')   # output => 'Hello Sara! Hello Ansh!'
```

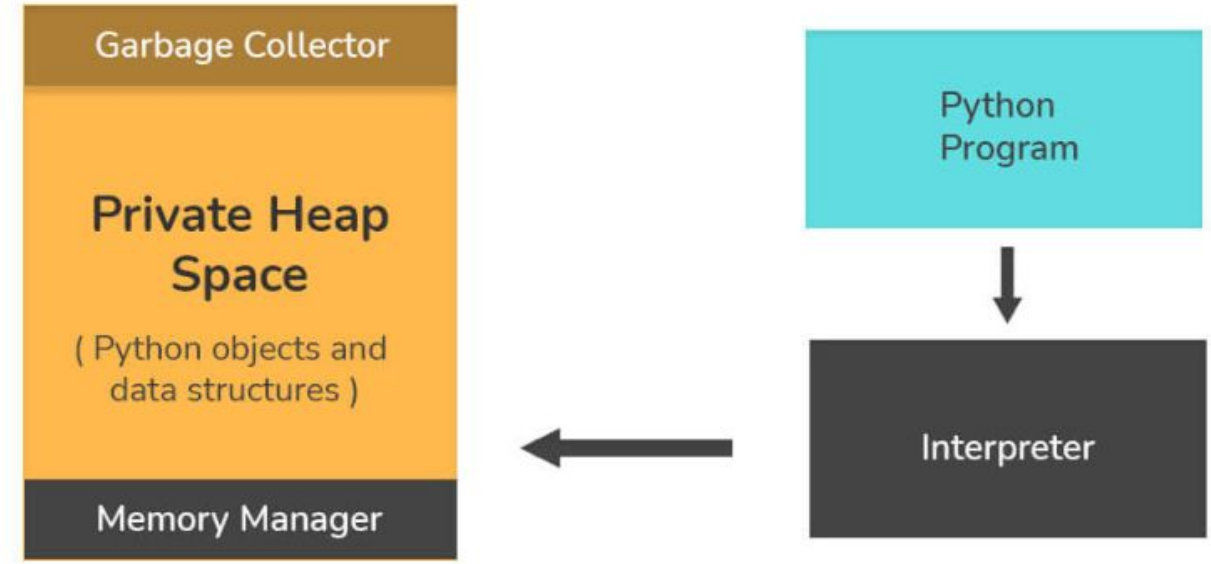
What is lambda in Python? Why is it used?

Lambda is an anonymous function in Python, that can accept any number of arguments, but can only have a single expression. It is generally used in situations requiring an anonymous function for a short time period.

```
mul = lambda a, b : a * b
print(mul(2, 5))    # output => 10
```

How is memory managed in Python?

- Memory management in Python is handled by the Python Memory Manager. The memory allocated by the manager is in form of a private heap space dedicated to Python. All Python objects are stored in this heap and being private, it is inaccessible to the programmer. Though, python does provide some core API functions to work upon the private heap space.
- Additionally, Python has an in-built garbage collection to recycle the unused memory for the private heap space.



What is pickling and unpickling?

Python library offers a feature - serialization out of the box. Serializing an object refers to transforming it into a format that can be stored, so as to be able to deserialize it, later on, to obtain the original object. Here, the pickle module comes into play.

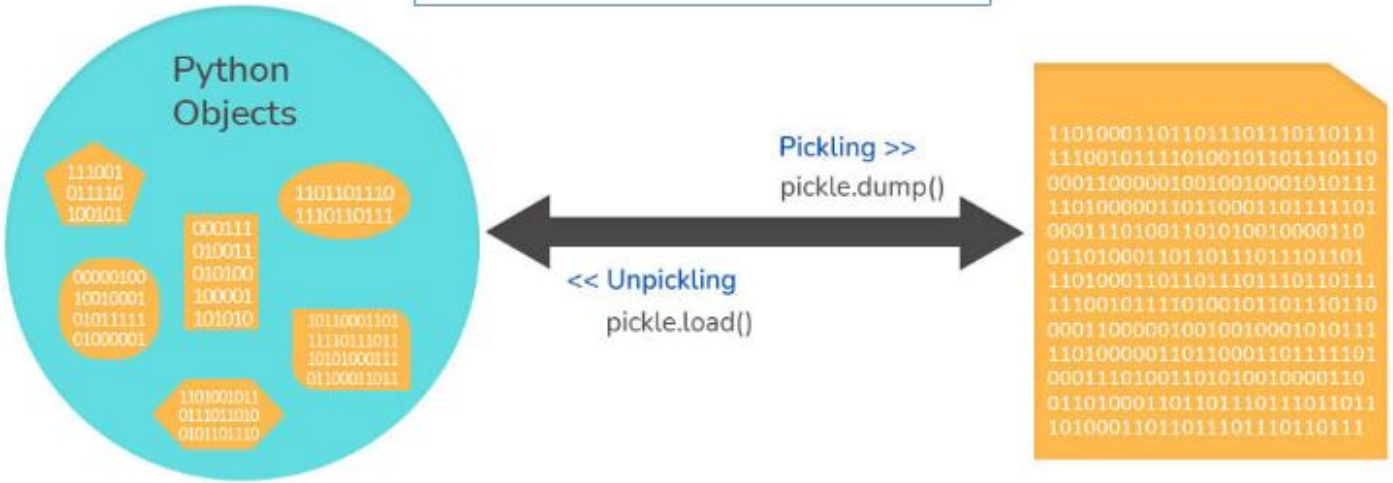
Pickling:

- Pickling is the name of the serialization process in Python. Any object in Python can be serialized into a byte stream and dumped as a file in the memory. The process of pickling is compact but pickle objects can be compressed further. Moreover, pickle keeps track of the objects it has serialized and the serialization is portable across versions.
- The function used for the above process is `pickle.dump()`.

Unpickling:

- Unpickling is the complete inverse of pickling. It deserializes the byte stream to recreate the objects stored in the file and loads the object to memory.
- The function used for the above process is `pickle.load()`

The Pickle Module



What are generators in Python?

Generators are functions that return an iterable collection of items, one at a time, in a set manner. Generators, in general, are used to create iterators with a different approach. They employ the use of `yield` keyword rather than `return` to return a generator object. Let's try and build a generator for fibonacci numbers -

```
## generate fibonacci numbers upto n
def fib(n):
   p, q = 0, 1
   while(p < n):
       yield p
       p, q = q, p + q
x = fib(10)    # create generator object

## iterating using __next__(), for Python2, use next()
x.__next__()    # output => 0
x.__next__()    # output => 1
x.__next__()    # output => 1
x.__next__()    # output => 2
x.__next__()    # output => 3
x.__next__()    # output => 5
x.__next__()    # output => 8
x.__next__()    # error

## iterating using loop
for i in fib(10):
   print(i)    # output => 0 1 1 2 3 5 8
```

What are iterators in Python?

- An iterator is an object.
- It remembers its state i.e., where it is during iteration (see code below to see how)
- __iter__() method initializes an iterator.
- It has a __next__() method which returns the next item in iteration and points to the next element.

  Upon reaching the end of iterable object __next__() must return StopIteration exception.
- It is also self-iterable.
- Iterators are objects with which we can iterate over iterable objects like lists, strings, etc.