Programming Interview Questions

The previous materials on the topic can be found at the following links:

- https://bit.ly/2YIdr02
- https://bit.ly/30lfdF7
- https://bit.ly/3Hf5W1Q

Q1. What do you know about palindromes? Can you implement one in Python?

A palindrome is a phrase, a word, or a sequence that reads the same forward and backward.

Palindrome implementation:

```
n = input()
n = int(n)
copy_n=n
result = 0

while(n!=0):
    digit = n%10
    result = result*10 + digit
    n=int(n/10)

print("Result is: ", result)
if(result==copy_n):
    print("Palindrome!")
else:
    print("Not a Palindrome!")
```

Input n : `123321`

Output :

```
Result is:  123321
Palindrome!
```

*Q2. What do you mean by *args and **kwargs?*

When we don't know how many arguments will be passed to a function, like when we want to pass a list or a tuple of values, we use *args.

```
>>> def func(*args):
   for i in args:
       print(i)
>>> func(3,2,1,4,7)
```

Output:

```
3
2
1
4
7
```

On the other hand **kwargs takes keyword arguments when we don't know how many there will be.

```
>>> def func(**kwargs):
   for i in kwargs:
       print(i,kwargs[i])
>>> func(a=1,b=2,c=7)
```

Output:

```
a.1
b.2
c.7
```

Note! The words args and kwargs are a convention, and we can use anything in their place.

Q3. What is a closure in Python? Can you give us an example?

Closures are these inner functions that are enclosed within the outer function. Closures can access variables present in the outer function scope. It can access these variables even after the outer function has completed its execution.

```
def outer(name):
    # this is the enclosing function

    def inner():
        # this is the enclosed function
        # the inner function accessing the outer function's variable 'name'
        print(name)

    return inner

# call the enclosing function
myFunction = outer('Python')
myFunction()
```

The call to outer function returns the inner function. This then gets assigned to 'myFunction'. When we call 'myFunction', it prints 'Python'. The power of the closure is that even after 'outer' finishes its execution and all its variables go out of scope, the value passed to its argument is still remembered.

Q4. When we have a close in Python?

We have a closure in Python in the following cases:

- We have a nested function, i.e. function within a function.
- The nested function refers to a variable of the outer function.
- The enclosing function returns the enclosed function.

Q5. Why do you need to use closures in Python?

- To replace the unnecessary use of class: Suppose you have a class that contains just one method besides the __init__ method. In such cases, it is often more elegant to use a closure instead of a class.
- To avoid the use of the global scope: If you have global variables which only one function in your program will use, think closure. Define the variables in the outer function and use them in the inner function.
- To implement data hiding: The only way to access the enclosed function is by calling the enclosing function. There is no way to access the inner function directly.
- To remember a function environment even after it completes its execution: You can then access the variables of this environment later in your program

Q6. What is the iterator protocol?

An iterator protocol is nothing but a specific class in Python which further has the __next()__ method. Which means every time you ask for the next value, an iterator knows how to compute it. It keeps information about the current state of the iterable it is working on. The iterator calls the next value when you call next() on it. An object that uses the __next__() method is ultimately an iterator.

Iterators help to produce cleaner looking code because they allows us to work with infinite sequences without having to reallocate resources for every possible sequence, thus also saving resource space. Python has several built-in objects, which implement the iterator protocol and you must have seen some of these before: lists, tuples, strings, dictionaries and even files.

```
iter()- To create an iterator
next()- To iterate to the next element

>>> a=iter([2,4,6,8,10])
>>> next(a)
2
>>> next(a)
4
>>> next(a)
6
>>> next(a)
8
>>> next(a)
10
```

Q7. What is tuple unpacking?

Unpacking tuples means assigning individual elements of a tuple to multiple variables. Use the * operator to assign remaining elements of an unpacking assignment into a list and assign it to a variable.

```
>>> nums=(1,2,3)
>>> a,b,c=nums
>>> a
1
>>> b
2
>>> c
3
```

Q8. What is a frozen set in Python?

A set is a collection of items, where there cannot be any duplicates. A set is also unordered.

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

Set is mutable. A frozen set is immutable. This means we cannot change its values. This also makes it eligible to be used as a key for a dictionary.

```
>>> myset=frozenset([1,3,2,2])
>>> myset
frozenset({1,3,2,2})
```

Q9. When you exit Python, is all memory deallocated?

Exiting Python deallocates everything except:

- modules with circular references.
- Objects referenced from global namespaces.
- Parts of memory reserved by the C library.

Q10. What is the Dogpile effect?

The Dogpile effect occurs when cache expires and websites are hit by numerous requests the same time. It is triggered because we allowed more than one request to execute the expensive query.

Q11. Explain garbage collection with Python.

- Python maintains a count of how many references there are to each object in memory.
- When a reference count drops to zero, it means the object is dead and Python can free the memory it allocated to that object.
- The garbage collector looks for reference cycles and cleans them up.
- Python uses heuristics to speed up garbage collection.
- Recently created objects might as well be dead.
- The garbage collector assigns generations to each object as it is created.
- It deals with the younger generations first.

Q12. Describe in brief how you'd convert JSON data into Python data?

Python supports JSON parsers. In fact, JSON-based data is internally represented as a dictionary in Python. To convert JSON data into Python data, we use the load() function from the JSON module.

Q13. Differentiate between split(), sub(), and subn() methods of the remodule.

The re module is what we have for processing regular expressions with Python.

- **split()** - This makes use of a regex pattern to split a string into a list.
- **sub()** - This looks for all substrings where the regex pattern matches, and replaces them with a different string.
- **subn()** - Like sub(), this returns the new string and the number of replacements made.

Q14. Whenever you exit Python, is all memory de-allocated?

The answer here is no. The modules with circular references to other objects, or to objects referenced from global namespaces, aren't always freed on exiting Python. Also, it is impossible to de-allocate portions of memory reserved by the C library.

Q15. Can I dynamically load a module in Python?

Dynamic loading is where we do not load a module till we need it. This is slow, but lets us utilize the memory more efficiently. In Python, you can use the importlib module for this:

```
import importlib
module = importlib.import_module('my_package.my_module')
```

Q16. How would you generate a random number in Python?

To generate a random number, we import the function random() from the module random.

```
>>> from random import random
>>> random()
```

Q17. Optionally, what statements can you put under a try-except block?

else - To run a piece of code when the try-block doesn't create an exception.

finally - To execute some piece of code regardless of whether there is an exception.

```
>>> try:
print("Hello")
except:
print("Sorry")
else:
print("Oh then")
finally:
print("Bye")
```

Output:

```
Hello
Oh then
Bye
```

Q18. Can you explain the filter(), map(), and reduce() functions?

filter() - This function lets us keep the values that satisfy some conditional logic.

```
>>> set(filter(lambda x:x>4, range(7)))
{5,6}
```

Filter the elements from 0 to 6 greater than the number 4.

map() - This function applies a function to each element in the iterable.

```
>>> set(map(lambda x:x**3, range(7)))
{0,1,64,8,216,27,125}
```

Calculates the cube for each element in the range 0 to 6 and stores them in a set.

reduce() - This function reduces a sequence pair-wise, repeatedly until we arrive at a single value.

```
>>> reduce(lambda x,y:y-x, [1,2,3,4,5])
3

Why 3 is the output?
2 - 1 = 1
3 - 1 = 2
4 - 2 = 2
5 - 2 = 3
```

Q19. How will you share global variables across modules?

To do this for modules within a single program, we create a special module and then import the config module in all modules of our application. This lets the module be global to all modules.

Q20. Write a regular expression that will accept an email id. Use the re module.

```
>>> import re
>>> e=re.search(r'[0-9a-zA-Z.]+@[a-zA-Z]+\.(com|co\.in)$','re@gmail.com')
>>> e.group()
```

Q21. What is pickling and unpickling?

"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 (from a binary file or bytes-like object) is converted back into an object hierarchy.

Q22. What is the MRO in Python?

MRO stands for Method Resolution Order. MRO is a concept used in inheritance. It is the order in which a method is searched for in a classes hierarchy and is especially useful in Python because Python supports multiple inheritance. In this case, the MRO would be C -> B -> A.

Q23. What are accessors, mutators, and @property?

What we call getters and setters in languages like Java, we term accessors and mutators in Python. In Java, if we have a user-defined class with a property 'x', we have methods like getX() and setX(). In Python, we have @property, which is syntactic sugar for property(). This lets us get and set variables without compromising on the conventions. For a detailed explanation on property, refer to Python property.

Q24. What do you mean by overriding methods?

Suppose class Second inherits from class First. Both have the method sayhello() - to each, their own version. Second overrides the sayhello() of class First. So, when we create an object of class Second, it calls the version that class Second has.

```
>>> class First:
   def sayhello(self):
       print("Hello, I'm First.")
>>> class Second(First):
  def sayhello(self):
      print("Hello, I'm Second.")
>>> a=First()
>>> b=Second()
>>> a.sayhello()
```

Output: `Hello, I'm First.`

Q25. How is memory managed in Python?

Python has a private heap space to hold all objects and data structures. Being programmers, we cannot access it; it is the interpreter that manages it. But with the core API, we can access some tools. The Python memory manager controls the allocation. Additionally, an inbuilt garbage collector recycles all unused memory so it can make it available to the heap space.

Q26. Have you heard of the yield keyword in Python?

Yes. This keyword bears the ability to turn any function into a generator. Much like the standard return keyword, but returns a generator object. It is also true that one function may observe multiple yields.

```
>>> def odds(n):
     odd=[i for i in range(n+1) if i%2!=0]
     for i in odd:
              yield i
>>> for i in odds(8):
     print(i)
```

Output:

```
1
3
5
7
```

Q27. If a function does not have a return statement, is it valid?

Very conveniently. A function that doesn’t return anything returns a None object. Not necessarily does the return keyword mark the end of a function; it merely ends it when present in the function. Normally, a block of code marks a function and where it ends, the function body ends.

Q28. What is a Thread?

A thread is a component of any process managed by the operating system. The OS achieves parallelism or multitasking by dividing the process among threads. It is a lightweight process that ensures a separate flow of execution.

Q29. How is multithreading achieved in Python?

A thread is a component of any process managed by the operating system. The OS achieves parallelism or multitasking by dividing the process among threads. It is a lightweight process that ensures a separate flow of execution.

1. Import the threading module

```
import threading
```

2. Starting the Thread

```
t1.start()
```

3. Join method of Thread

```
def print_hi(num):
    print("Hi, you are customer ",num)

t1 = threading.Thread(target = print_hi, args=(10,))
t1.start()
t1.join()
print("End")
```

Output:

```
Hi, you are customer 10
End
```

The Thread class is used to create an object and it has been named as t1. The start() method is invoked on the thread object t1 which marks the beginning of the thread activity. The join() method is then called. By doing so, we ensure that the main program halts execution of the main thread and waits until the completion of thread t1. Once t1 has completed its activity, the main thread (main program) can continue its execution. Hence, the line print("End") is executed only after the completion of the thread activity.

Q30. What are the Benefits of Multithreading in Python?

Some of the benefits are:

- Effective utilization of resources.
- More responsive.
- Resource sharing makes it more economical.
- Effective use of Multiprocessor architecture due to parallelism
- Saves time.

- Threads (since part of the same process) communicate with each other more easily than if they were separate processes.
- They do not require much memory overhead.
- Multi-threaded servers and interactive GUIs use multithreading exclusively.

Q31. What are the disadvantages of multithreading?

- Increases the complexity of the program.
- Synchronization of shared resources (objects, data) is necessary.
- Difficult to debug unpredictable results.
- Constructing and synchronizing threads is CPU/memory intensive.

Resources: 1. https://bit.ly/3ophO8P

2. https://bit.ly/30iYphL

3. https://github.com/rusevrosen/codehub.github.io