# **Memory Management in Python**

* Python optimizes memory utilization by allocating same reference to a new variable if object already exist
* Objects and instance variables are created in heap memory whereas its references variables and methods are created in stack memory
* A new stack frame is created for handling methods and variables during execution and stacks frames are destroyed automatically whenever functions/methods returns.
* Python has mechanism called Garbage collector, it runs as soon as reference counter for the object becomes zero, it removed dead object from memory
* The algorithm used for Garbage collection is called reference counting
* But, Java uses mark and swiped algorithm for garbage collection, in which dead objects marked and swiped periodically not immediately

**Deep and Shallow Copy in Python**

* A deep copy copies an object into another. A shallow copy, however, copies one object’s reference to another.
* Deep copy doesn't reflect changes made to the new/copied object in the original object; whereas, shallow copy does
* Shallow copies allow faster execution whereas Deep copies make execution of program slower
* Deep copy: l2 = copy.deepcopy(l1), Shallow Copy: l2 = copy.copy(l1)

**Difference between lists and tuples**

* Lists are mutable, tuples are immutable

**Multi-Threading – when should we use?**

* Threading in python is used to run multiple threads (tasks, function calls) at the same time. Note that this does not mean that they are executed on different CPUs. Python threads will NOT make your program faster if it already uses 100 % CPU time. In that case, you probably want to look into parallel programming.
* Python threads are used in cases where the execution of a task involves some waiting. One example would be interaction with a service hosted on another computer, such as a webserver. Threading allows python to execute other code while waiting; this is easily simulated with the sleep function.

**Turnery Operators**

* The ternary operator is a way of writing conditional statements in Python. As the name ternary suggests, this Python operator consists of three operands. For ex, big = x if x>y else y

**Monkey Patching**

* In Python, the term monkey patch refers to dynamic (or runtime) modifications of a class or module
* It is used when developes needs to modify or extend behaviour of a third-party or in-bilt modules and does not wish to maintain a private copy of the source code
* In other words, we want current instance to be patched; we want to build somethin on top of the existing method, not to replace entriely
* Import m
* Def monkey\_f(self):
* Print “monky function”
* Obj = m.MyClass()
* m.MyClass.f = monkey\_f
* Obj.f()
* o/p --> monkey function

**RE module functions**

|  |  |
| --- | --- |
| [findall](https://www.w3schools.com/python/python_regex.asp#findall) | Returns a list containing all matches |
| [search](https://www.w3schools.com/python/python_regex.asp#search) | Returns a Match object if there is a match anywhere in the string |
| [split](https://www.w3schools.com/python/python_regex.asp#split) | Returns a list where the string has been split at each match |
| [sub](https://www.w3schools.com/python/python_regex.asp#sub) | Replaces one or many matches with a string |

The Match object has properties and methods used to retrieve information about the search, and the result:

.span() returns a tuple containing the start-, and end positions of the match.  
.string returns the string passed into the function  
.group() returns a part of the string where there was a match

**Difference between findall() and finditer()**

difference between finditer and findall is that the former returns regex match objects whereas the other returns a list of the matched capturing groups (or the entire match if there are no capturing groups).

**Errors & Exceptions**

* It is used when you want to run rest of the script even if a particular block of code raises an error.

4 keywords:

Try: A main logic that might cause an error

Except: How you want to handle this error

Else: only executes if there is no error (else for except)

Finally: executes regardless there is an error or not.

**What is the PYTHONPATH variable?**

* PYTHONPATH is the variable that tells the interpreter where to locate the module files imported into a program. Hence, it must include the Python source library directory and the directories containing Python source code. You can manually set PYTHONPATH, but usually, the Python installer will preset it.

**Map, filter and reduce functions**

**a. filter()**

Filter lets us filter in some values based on conditional logic.

1. >>> **list**(**filter**(lambda x:x>5,**range**(8)))

**[6, 7]**

**b. map()**

Map applies a function to every element in an iterable.

1. >>> **list**(**map**(lambda x:x\*\*2,**range**(8)))
2. [0, 1, 4, 9, 16, 25, 36, 49]

**c. reduce()**

Reduce repeatedly reduces a sequence pair-wise until we reach a single value.

1. >>> from functools import reduce
2. >>> **reduce**(lambda x,y:x-y,[1,2,3,4,5])

**Is del the same as remove()? What are they?**

del and remove() are methods on lists/ ways to eliminate elements.

>>> list=[3,4,5,6,7]

>>> del list[3]

>>> list

o/p: [3, 4, 5, 7]

>>> list.**remove**(5)

>>> list

o/p: [3, 4, 7]

**What is the output of the following piece of code?**

>>> tuple=(123,'John')

>>> tuple\*=2

>>> tuple

o/p: (123, ‘John’, 123, ‘John’)

**Differentiate between the append() and extend() methods of a list.**

The methods append() and extend() work on lists. While append() adds an element to the end of the list, extend adds another list to the end of a list.

Let’s take two lists.

>>> list1,list2=[1,2,3],[5,6,7,8]

This is how append() works:

>>> list1.**append**(4)

>>> list1

o/p: [1, 2, 3, 4]

And this is how extend() works:

>>> list1.**extend**(list2)

>>> list1

o/p: [1, 2, 3, 4, 5, 6, 7, 8]

**What are the different file-processing modes with Python?**

* read-only – ‘r’
* write-only – ‘w’
* read-write – ‘rw’
* append – ‘a’

**Raise keyword in python**

* The raise keyword is used to raise an exception.

You can define what kind of error to raise, and the text to print to the user.

Raise a TypeError if x is not an integer:

x = "hello"  
  
 if not type(x) is int:  
 raise TypeError("Only integers are allowed")

**Is there a way to remove the last object from a list?**

Yes, there is. Try running the following piece of code-

>>> list=[1,2,3,4,5]

>>> list.**pop**()

**How will you convert an integer to a Unicode character?**

All we need is the chr(x) built-in function to convert

**So, does recursion cause any trouble?**

Sure does:

Needs more function calls.

Each function call stores a state variable to the program stack- consumes memory, can cause memory overflow.

Calling a function consumes time.

**What good is recursion?**

With recursion, we observe the following:

Need to put in less efforts.

Smaller code than that by loops.

Easier-to-understand code.

**What does the following code give us?**

>>> b=(1)

Not a tuple. This gives us a plain integer.

>>> **type**(b)

To let it be a tuple, we can declare so explicitly with a comma after 1:

>>> b=(1,)

>>> **type**(b)

**Why are identifier names with a leading underscore disparaged?**

Since Python does not have a concept of private variables, it is a convention to use leading underscores to declare a variable private. This is why we mustn’t do that to variables we do not want to make private.

**Can you remove the whitespaces from the string “aaa bbb ccc ddd eee”?**

I can think of three ways to do this.

Using join-

>>> s='aaa bbb ccc ddd eee'

>>> s1=''.join(s.split())

Using a list comprehension–

>>> s='aaa bbb ccc ddd eee'

>>> s1=str(''.join(([i for i in s if i!=' '])))

Using replace()-

>>> s='aaa bbb ccc ddd eee'

>>> s1 = s.replace(' ','')

**How do you get the current working directory using Python?**

>>> import os

>>> os.**getcwd**()

**How would you randomize the contents of a list in-place?**

For this, we’ll import the function shuffle() from the module random.

>>> from random import shuffle

>>> shuffle(mylist)

**How do you remove the leading whitespace in a string?**

Leading whitespace in a string is the whitespace in a string before the first non-whitespace character. To remove it from a string, we use the method lstrip().

>>> ' Ayushi '.lstrip()

**‘Ayushi ‘**

As you can see, this string had both leading and trailing whitespaces. lstrip() stripped the string of the leading whitespace. If we want to strip the trailing whitespace instead, we use rstrip().

>>> ' Ayushi '.rstrip()

**‘ Ayushi’**

**How will you create the following pattern using Python?**  
**\***  
**\*\***  
**\*\*\***  
**\*\*\*\***  
**\*\*\*\*\***

We will use two for-loops for this.

>>> for i in range(1,6):

for j in range(1,i+1):

print('\*',end='')

print()

**Difference between generator, iterator and iterable**

An Iterable is an object which has \_\_iter\_\_ method to return iterator object and Iterator is an object which has \_\_iter\_\_ as well as \_\_next\_\_ method which accesses it’s elements in the container one at a time

For a generator, we create a function. For an iterator, we use in-built functions iter() and next().

For a generator, we use the keyword ‘yield’ to yield/return an object at a time.

A generator may have as many ‘yield’ statements as you want.

A generator will save the states of the local variables every time ‘yield’ will pause the loop. An iterator does not use local variables; it only needs an iterable to iterate on.

Using a class, you can implement your own iterator, but not a generator.

Generators are fast, compact, and simpler.

Iterators are more memory-efficient.

**What functions or methods will you use to delete a file in Python?**

For this, we may use remove() or unlink().

>>> import os

>>> os.**chdir**('C:\\Users\\Desktop')

>>> os.**remove**('try.py')

>>>

When we go and check our Desktop, the file is gone. Let’s go make it again so we can delete it again using unlink().

>>> os.**unlink**('try.py')

>>>

Both functions are the same, but unlink is the traditional Unix name for it.

**How would you make a Python script executable on Unix?**

You need to do two things: the script file’s mode must be executable and the first line must begin with #! followed by the path of the Python interpreter.

The first is done by executing “chmod +x scriptfile” or perhaps “chmod 755 scriptfile.

$ chmod +x myscript.py

The second can be done in a number of ways. The most straightforward way is to put

#!/usr/local/bin/python

as the very first line of your file, using the pathname for where the Python interpreter is installed on your platform.

**Write Python logic to count the number of capital letters in a file.**

import os

os.**chdir**('C:\\Users\\lifei\\Desktop')

with **open**('Today.txt') as today:

count=0

for i in today.**read**():

if i.**isupper**():

count+=1

**print**(count)

**Explain inheritance in Python.**

When one class inherits from another, it is said to be the child/derived/sub class inheriting from the parent/base/super class. It inherits/gains all members (attributes and methods).

Inheritance lets us reuse our code, and also makes it easier to create and maintain applications. Python supports the following kinds of inheritance:

Single Inheritance- A class inherits from a single base class.

Multiple Inheritance- A class inherits from multiple base classes.

Multilevel Inheritance- A class inherits from a base class, which, in turn, inherits from another base class.

Hierarchical Inheritance- Multiple classes inherit from a single base class.

Hybrid Inheritance- Hybrid inheritance is a combination of two or more types of inheritance.

**Explain, in brief, the uses of the modules sqlite3, ctypes, pickle, traceback, and itertools.**

sqlite3- Helps with handling databases of type SQLite

ctypes- Lets create and manipulate C data types in Python

pickle- Lets put any data structure to external files

traceback- Allows extraction, formatting, and printing of stack traces

itertools– Supports working with permutations, combinations, and other useful iterables.

**What are the file-related modules we have in Python?**

We have the following libraries and modules that let us manipulate text and binary files on our file systems-

**os**  
**os.path**  
**shutil**

**What is a Python module?**

A module is a script in Python that defines import statements, functions, classes, and variables. It also holds runnable Python code. ZIP files and DLL files can be modules too. The module holds its name as a string that is in a global variable.

**Are methods and constructors the same thing?**

No, there are subtle but considerable differences-

We must name a constructor in the name of the class; a method name can be anything.

Whenever we create an object, it calls a constructor by defalt; for a method we need to call it

For one object, a constructor executes only once; a method can execute any number of times for one object.

We use constructors to define and initialize non-static variables; we use methods to represent business logic to perform operations.

**Which methods/functions do we use to determine the type of instance and inheritance?**

Here, we talk about three methods/functions- type(), isinstance(), and issubclass().

**a. type()**

This tells us the type of object we’re working with.

**type**(3)

<class ‘int’>

**b. isinstance()**

This takes in two arguments- a value and a type. If the value is of the kind of the specified type, it returns True. Else, it returns False.

**isinstance**(3,int)

True

**c. issubclass()**

This takes two classes as [***arguments***](https://data-flair.training/blogs/python-function-arguments/). If the first one inherits from the second, it returns True. Else, it returns False.

class A: pass

class **B**(A): pass

**issubclass**(B,A)

True

**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')

**How do you debug a program in Python? Answer in brief.**

To debug a Python program, we use the pdb module. This is the Python debugger. If we start a program using pdb, it will let us step through the code.

**How can you keep track of different versions of code?**

To make this happen, we implement version control. For this, one tool you can use is Git.

**If while installing a package with pip, you get the error No matching installation found, what can you do?**

In such a situation, one thing I can do is to download the binaries for that package from the following location:

[**https://www.lfd.uci.edu/~gohlke/pythonlibs/**](https://www.lfd.uci.edu/~gohlke/pythonlibs/)

Then, I can install the wheel using pip.

**If you installed a module with pip but it doesn’t import in your IDLE, what could it possibly be?**

Well, for one, it could be that I installed two versions of Python on my system- possibly, both 32-bit and 64-bit.

The Path variable in my system’s environment variables is probably set to both, but one of them prior to the other- say, the 32-bit.

This made the command prompt use the 32-bit version of pip to install the module I chose.

When I ran the IDLE, I ran the 64-bit version.

As this sequence of events unlapped, I couldn’t import the module I just installed.

**Based on your previous answer, how will you solve this issue?**

I could do two things.

**The temporary solution-** I will add the path to sys manually every time I work on a new session of the interpreter.

1. >>> sys.path.**append**('C:\\Users\\Ayushi\\AppData\\Local\\Programs\\Python\\Python37\\Scripts')

**The permanent solution-** I will update the value of Path in my environment variables to hold the location of the Scripts folder for the 64-bit version first.

**Is a NumPy array better than a list?**

NumPy arrays have 3 benefits over lists:

They are faster

They require less memory

They are more convenient to work with

**Can you make a local variable’s name begin with an underscore? (developer)**

You can, but you should not. This is because:

Local variables indicate private variables of a class, and so, they confuse the interpreter.

**Where will you use while rather than for?**

Although we can do with for all that we can do with while, there are some places where a while loop will make things easier-

For simple repetitive looping

When we don’t need to iterate through a list of items- like database records and characters in a string.

In languages like C++, we have something like this:

**switch**(name)

{

case ‘Ayushi’:

cout<<”Monday”;

break;

case ‘Megha’:

cout<<”Tuesday”;

break;

default:

cout<<”Hi, user”;

}

But in Python, we do not have a [switch-case statement](https://data-flair.training/blogs/python-switch-case/). Here, you may write a switch function to use. Else, you may use a set of if-elif-else statements. To implement a function for this, we may use a dictionary.

>>> def **switch**(choice):

switcher={

'Ayushi':'Monday',

'Megha':'Tuesday',

**print**(switcher.**get**(choice,'Hi, user'))

**Can you make a local variable’s name begin with an underscore?**

You can, but you should not. This is because:

Local variables indicate private variables of a class, and so, they confuse the interpreter.

* Template functions
* GST and timezone
* Pickels
* Hash value
* Re.findall
* How to write millions of row
* Built in decorators
* Purpose of serialization in django