**Python** is an [interpreted](https://en.wikipedia.org/wiki/Interpreted_language), [high-level](https://en.wikipedia.org/wiki/High-level_programming_language), [general-purpose](https://en.wikipedia.org/wiki/General-purpose_programming_language) [programming language](https://en.wikipedia.org/wiki/Programming_language). It is is [dynamically typed](https://en.wikipedia.org/wiki/Dynamic_programming_language) and [garbage-collected](https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)).

Python is **Case Sensitive.**

**Zen of python – import this**

#!(shebang) In Python, a shebang (also known as a hashbang) is the first line of a script, which specifies the interpreter that should be used to run the script.

The Python Enhancement Proposal, also known as PEP 8, is a document that provides instructions on how to write Python code. In essence, it is a set of guidelines for formatting Python code for maximum readability.

Python is an interpreted language and not a compiled one, although compilation is a step. Python code, written in **.py** file is first compiled to what is called bytecode (discussed in detail further) which is stored with a **.pyc** or **.pyo** format.

Instead of translating source code to machine code like C++, Python code it translated to bytecode (.pyc). This bytecode is a low-level set of instructions that can be executed by an **interpreter (PVM)**. In most PCs, Python interpreter is installed at /usr/local/bin/python3.8.

Instead of executing the instructions on CPU, bytecode instructions are executed on a Virtual Machine.

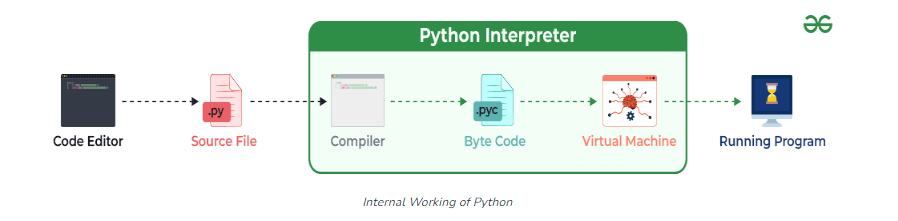
**What is \_\_pycache\_\_ ?**

Many times in your personal project or on GitHub, you might have seen a folder named \_\_pycache\_\_ being created automatically.

/folder - \_\_pycache\_\_ - preprocess.cpython-36.pyc - preprocess.py

As you can see, the filename is the same as the one outside \_\_pycache\_\_ folder. The .pyc extension tells us that the file contains bytecode for preprocess.py. The names **cpython**denotes the type of interpreter. CPython means that the interpreter was implemented in C language. Similarly, JPython is a Python interpreter implemented in Java.

But why is the folder created in the first place? Well, it slightly increases the speed of the Python program. Unless you change your Python code, recompilation to bytecode is avoided, thereby saving time.



**The Python Package Index** (PyPI) is a repository of software for the Python programming language.

 pip is a package manager for installing and managing Python packages.

A virtual environment is (amongst other things):

* Used to contain a specific Python interpreter and software libraries and binaries which are needed to support a project (library or application). These are by default isolated from software in other virtual environments and Python interpreters and libraries installed in the operating system.
* Contained in a directory, conventionally either named venv or .venv in the project directory, or under a container directory for lots of virtual environments, such as ~/.virtualenvs.
* Not checked into source control systems such as Git.
* Considered as disposable – it should be simple to delete and recreate it from scratch. You don’t place any project code in the environment
* Not considered as movable or copyable – you just recreate the same environment in the target location.

Tuple is immutable

Set cannot contain duplicate elements

False 0

True 1

Docstrings and Multi-line comments may look the same but they aren’t.

* Docstrings are written in the functions and classes to show how to use the program.
* Multi-line comments are used to show how a block of code works.

In Python, **with statement** is used in exception handling to make the code cleaner and much more readable. It simplifies the management of common resources like file streams.

### The contextlib module

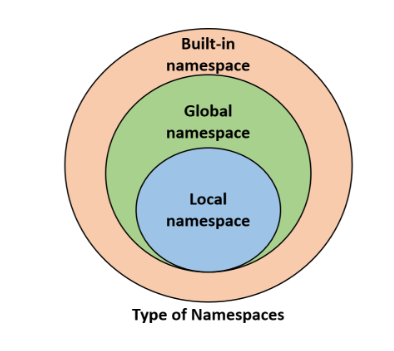
A class based context manager as shown above is not the only way to support the with statement in user defined objects. The [contextlib](https://docs.python.org/2/library/contextlib.html" \t "_blank) module provides a few more abstractions built upon the basic context manager interface. Here is how we can rewrite the context manager for the MessageWriter object using the contextlib module.

Input() – Takes any type of values

Raw\_input() – Takes value and convert it to string(Python 2.x)

### **Types of namespaces :**

When Python interpreter runs solely without any user-defined modules, methods, classes, etc. Some functions like print(), id() are always present, these are built-in namespaces. When a user creates a module, a global namespace gets created, later the creation of local functions creates the local namespace. The **built-in namespace** encompasses the **global namespace** and the global namespace encompasses the **local namespace**.



### The lifetime**of a namespace :**

A lifetime of a namespace depends upon the scope of objects, if the scope of an object ends, the lifetime of that namespace comes to an end. Hence, it is not possible to access the inner namespace’s objects from an outer namespace.

**Python Global variables** are those which are not defined inside any function and have a global scope whereas Python **local variables** are those which are defined inside a function and their scope is limited to that function only.

Converting **Strings to byte** objects is termed as encoding. This is necessary so that the text can be stored on disk using mapping using **ASCII** or **UTF-8** encoding techniques.

Similarly, Decoding is process to convert a**Byte object to String**. It is implemented using [**decode()**](https://www.geeksforgeeks.org/python-strings-decode-method/) . A byte string can be decoded back into a character string, if you know which encoding was used to encode it. Encoding and Decoding are **inverse** processes.

Since there is no main() function in Python, when the command to run a python program is given to the interpreter, the code that is at level 0 indentation is to be executed. However, before doing that, it will define a few special variables. \_\_name\_\_ is one such special variable. If the source file is executed as the main program, the interpreter sets the \_\_name\_\_ variable to have a value “\_\_main\_\_”. If this file is being imported from another module, \_\_name\_\_ will be set to the module’s name.  
**\_\_name\_\_ is a built-in variable which evaluates to the name of the current module.** Thus it can be used to check whether the current script is being run on its own or being imported somewhere else by combining it with if statement, as shown below.

**Python Lists**are just like dynamically sized arrays, declared in other languages (vector in C++ and ArrayList in Java). In simple language, a list is a collection of things, enclosed in [ ] and separated by commas.

**deque is implemented as a doubly linked list, making it efficient for appending and popping from both ends. Operations such as append, appendleft, pop, and popleft are all O(1) on average. However, indexing and slicing operations are O(n) because deque doesn’t support fast random access.**

**Use Cases**

* **Queue: When you need a FIFO (First In, First Out) queue, deque is a perfect choice.**
* **Stack: For LIFO (Last In, First Out) behavior, deque can also serve as a stack.**
* **Sliding Window: deque can efficiently manage a sliding window of fixed size due to its fast append and pop operations from both ends.**

**Tuple**is a collection of Python objects much like a list. The sequence of values stored in a tuple can be of any type, and they are indexed by integers.

Values of a tuple are syntactically separated by ‘commas’. Although it is not necessary, it is more common to define a tuple by closing the sequence of values in parentheses. This helps in understanding the Python tuples more easily.

## Accessing of Tuples

**Tuples** are **immutable**, and usually, they contain a sequence of heterogeneous elements that are accessed via [unpacking](https://www.geeksforgeeks.org/unpacking-a-tuple-in-python/)or indexing (or even by attribute in the case of named tuples)

In Python, a **Set**is an unordered collection of data types that is iterable, mutable and has **no duplicate** elements. The order of elements in a set is undefined though it may consist of various elements. The major advantage of using a set, as opposed to a list, is that **it has a highly optimized method for checking whether a specific element is contained in the set**.

A set contains only unique elements but at the time of set creation, multiple duplicate values can also be passed. Order of elements in a set is undefined and is unchangeable. Type of elements in a set need not be the same, various mixed-up data type values can also be passed to the set.

Set **items cannot be accessed by referring to an index, since sets are unordered the items has no index**. But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

Elements can be removed from the Set by using the built-**in remove() function but a KeyError arises if the element doesn’t exist in the set**. To remove elements from a **set without KeyError, use discard(),** if the element doesn’t exist in the set, it remains unchanged.

### Using pop() method:

Pop() function can also be used to remove and return an element from the set, but it removes only the last element of the set.

### Using clear() method:

To remove all the elements from the set, clear() function is used.

**Frozen sets** in Python are **immutable objects that only support methods and operators that produce a result without affecting the frozen set** or sets to which they are applied. While elements of a set can be modified at any time, elements of the frozen set remain the same after creation.

If no parameters are passed, it returns an empty frozenset.

**A dictionary in Python** is a data structure that stores the value in value:key pairs.

In Python, generics is a mechanism with which you to define functions, classes, or methods that can operate on multiple types while maintaining type safety. With the implementation of Generics enable it is possible to write reusable code that can be used with different data types. It ensures promoting code flexibility and type correctness.

Generics in Python are implemented using type hints. This feature was introduced in Python with version 3.5 onwards.

Normally, you don't need to declare a variable type. The type is determined dynamically by the value assigned to it. Python's interpreter doesn't perform type checks and hence it may raise runtime exceptions.

Python's new type hinting feature helps in prompting the user with the expected type of the parameters to be passed.

Type hints allow you to specify the expected types of variables, function arguments, and return values. Generics extend this capability by introducing type variables, which represent generic types that can be replaced with specific types when using the generic function or class.

## Creating an Array

Array in Python can be created by importing an array module. **array(*data\_type*, *value\_list*)** is used to create an array with data type and value list specified in its arguments.

## ****Generator Function in Python****

A generator function in Python is defined like a normal function, but whenever it needs to generate a value, it does so with the [yield keyword](https://www.geeksforgeeks.org/python-yield-keyword/) rather than return. If the body of a def contains yield, the function automatically becomes a Python generator function.

**in operator:**The ‘in’ operator is used to check if a character/ substring/ element exists in a sequence or not. Evaluate to True if it finds the specified element in a sequence otherwise False.

dict1={1:'Geeks',2:'For',3:'Geeks'} # Checking 3 in keys of dictionary

3 in dict1

True

**‘is’ operator –** Evaluates to True if the variables on either side of the operator point to the same object and false otherwise.

# Python program to illustrate the use

# of 'is' identity operator

x **=** 5

y **=** 5

print(x **is** y)

id(x)

id(y)

==(equality) check for value and is (identity)check for values and memory

Python’s Itertool is a module that provides various functions that work on iterators to produce complex iterators. This module works as a fast, memory-efficient tool that is used either by themselves or in combination to form **iterator algebra**.

Python iterators implement the [iterator design pattern](https://en.wikipedia.org/wiki/Iterator_pattern), which allows you to traverse a container and access its elements.

## Python \_\_iter\_\_()

The [\_\_iter\_\_()](https://www.geeksforgeeks.org/iterators-in-python/) function in Python returns an iterator for the given object (array, set, tuple, etc., or custom objects). It creates an object that can be accessed one element at a time using **\_\_next\_\_()** in Python, which generally comes in handy when dealing with loops.

## Python \_\_next\_\_()

Python \_\_next\_\_() is responsible for returning the next element of the iteration**. If there are no more elements then it raises the StopIteration exception**. It is part of the iterable and iterator interface, which allows us to create custom iterable objects, such as generators, and control how elements are retrieved one at a time from those iterables.

**Iterable:** An iterable is anything you can loop over. Imagine it as a container or a thing that holds a bunch of items, and you can go through each item one by one. Think of it like a list of items, where you can look at each item individually.

For example, a shopping list is an iterable. You can go through each item on the list one after the other. The list itself is iterable because you can iterate (go through) its items.

**Iterator:** An iterator, on the other hand, is like a pointer or a cursor that helps you move through an iterable. It keeps track of where you are in the iterable and gives you the next item when you ask for it.

Continuing with the shopping list analogy, an iterator is like your finger pointing to the current item on the list. You start at the first item, check it off, move your finger to the next item, and so on. The iterator keeps track of your position in the iterable (shopping list) and helps you get the next item.

## Python Class

A class is a collection of objects. A class contains the blueprints or the prototype from which the objects are being created. It is a logical entity that contains some attributes and methods.

## Python Objects

**An object consists of:**

* **State:** It is represented by the attributes of an object. It also reflects the properties of an object.
* **Behavior:** It is represented by the methods of an object. It also reflects the response of an object to other objects.
* **Identity:** It gives a unique name to an object and enables one object to interact with other objects.

## Abstract Base Class

Python has five abstract base classes. They are as follows:

* Callable (\_\_call\_\_)
* Container (\_\_contains\_\_)
* Hashable (\_\_hash\_\_)
* Iterable (\_\_iter\_\_)
* Sized (\_\_len\_\_)

These abstract base classes contain one abstract method each.

**Static Variables**

In Python, a static variable is a variable that is shared among all instances of a class, rather than being unique to each instance. It is also sometimes referred to as a class variable, because it belongs to the class itself rather than any particular instance of the class.

Static variables are defined inside the class definition, but outside of any method definitions. They are typically initialized with a value, just like an instance variable, but they can be accessed and modified through the class itself, rather than through an instance.

**Class method vs Static Method**

The difference between the Class method and the static method is:

* A class method takes cls as the first parameter while a static method needs no specific parameters.
* A class method can access or modify the class state while a static method can’t access or modify it.
* In general, static methods know nothing about the class state. They are utility-type methods that take some parameters and work upon those parameters. On the other hand class methods must have class as a parameter.
* We use @classmethod decorator in python to create a class method and we use @staticmethod decorator to create a static method in python.

**When to use the class or static method?**

* We generally use the class method to create factory methods. Factory methods return class objects ( similar to a constructor ) for different use cases.
* We generally use static methods to create utility functions.

**The counter** is a **container** included in the [collections](https://www.geeksforgeeks.org/python-collections-module/) module. Now you all must be wondering what is a container. Don’t worry first let’s discuss the container. In Python, the Counter class from the collections module is a handy tool for counting hashable objects. It’s essentially a dictionary subclass designed for counting elements.

## What is Container?

Containers are objects that hold objects. They provide a way to access the contained objects and iterate over them. Examples of built-in containers are Tuples, lists, and dictionaries. Others are included in the [Collections](https://www.geeksforgeeks.org/python-collections-module/) module.  
A Counter is a subclass of dict. Therefore **it is an unordered collection where elements and their respective count are stored as a dictionary**.

An **OrderedDict** is a dictionary subclass that remembers the order in which keys were first inserted. The only difference between [dict()](https://www.geeksforgeeks.org/python-set-4-dictionary-keywords-python/) and OrderedDict() lies in their handling of key order in [Python](https://www.geeksforgeeks.org/python-programming-language/).

### OrderedDict vs dict in Python

**`OrderedDict` maintains the sequence in which keys are added, ensuring that the order is preserved during iteration. In contrast, a standard dictionary does not guarantee any specific order when iterated, providing values in an arbitrary sequence.** `OrderedDict` distinguishes itself by retaining the original insertion order of items.

**Key Differences Between OrderedDict and dict**

* [**Insertion Order: OrderedDict maintains the insertion order of keys, whereas the standard dict did not guarantee this before Python 3.71**](https://www.geeksforgeeks.org/ordereddict-in-python/)**.**
* **Methods: OrderedDict provides additional methods like move\_to\_end() and popitem(last=True) which are not available in a regular dict.**

**Defaultdict** is a container like [dictionaries](https://www.geeksforgeeks.org/python-dictionary/) present in the module **collections**. Defaultdict is a sub-class of the dictionary class that returns a dictionary-like object. The functionality of both dictionaries and defaultdict are almost same except for the fact that **defaultdict never raises a KeyError. It provides a default value for the key that does not exists**.

**Advantages Over Regular**dict

* **No KeyError**: Automatically handles missing keys by providing a default value.
* **Simplifies Code**: Reduces the need for explicit checks for key existence.
* **Customizable**: Allows for flexible default values through the default\_factory function.

**When to Use**defaultdict

* **Handling Missing Keys**: When you frequently access keys that might not exist.
* **Counting and Grouping**: When you need to count occurrences or group items by a key.

UserDict:

**When to Use**UserDict

* **Custom Dictionaries**: When you need a dictionary with custom behavior or additional methods.
* **Subclassing**: When you want to create a subclass of a dictionary without modifying the built-in dict class.

**When to Use**UserList

* **Custom Lists**: When you need a list with custom behavior or additional methods.
* **Subclassing**: When you want to create a subclass of a list without modifying the built-in list class.

**When to Use**UserString

* **Custom Strings**: When you need a string with custom behavior or additional methods.
* **Subclassing**: When you want to create a subclass of a string without modifying the built-in str class.

Python contains a container called “**ChainMap**” which encapsulates many [dictionaries](https://www.geeksforgeeks.org/python-set-4-dictionary-keywords-python/) into one unit. ChainMap is member of module “[collections](https://www.geeksforgeeks.org/python-collections-module/)“.

## What is NamedTuple in Python?

In [Python](https://www.geeksforgeeks.org/python-programming-language/), NamedTuple is present inside the [collections module](https://www.geeksforgeeks.org/python-collections-module/). It provides a way to create simple, lightweight data structures similar to a class, but without the overhead of defining a full class. Like dictionaries, they contain keys that are hashed to a particular value. On the contrary, it supports both access from key-value and iteration, the functionality that [dictionaries](https://www.geeksforgeeks.org/python-dictionary/)lack.

[Garbage Collection in Python - GeeksforGeeks](https://www.geeksforgeeks.org/garbage-collection-python/)

### Method resolution order:

In Python, every class whether built-in or user-defined is derived from the object class and all the objects are instances of the class object. Hence, the object class is the base class for all the other classes.  
In the case of multiple inheritance, a given attribute is first searched in the current class if it’s not found then it’s searched in the parent classes. The parent classes are searched in a left-right fashion and each class is searched once.  
If we see the above example then the order of search for the attributes will be Derived, Base1, Base2, object. The order that is followed is known as a linearization of the class Derived and this order is found out using a set of rules **called Method Resolution Order (MRO)**

***Python Magic methods*** are the methods starting and ending with double underscores ‘\_\_’. They are defined by built-in classes in Python and commonly used for operator overloading.

They are also called **Dunder methods,** Dunder here means “Double Under (Underscores)”.

\_\_\*\_\_  
System-defined names. These names are defined by the interpreter and its implementation (including the standard library). Current system names are discussed in the Special method names section and elsewhere. More will likely be defined in future versions of Python. Any use of \_\_\*\_\_ names, in any context, that does not follow explicitly documented use, is subject to breakage without warning.

In Python, the term monkey patch refers to dynamic (or run-time) modifications of a class or module. In Python, we can actually change the behavior of code at run-time.

Usage of Monkey patching:  
For instance, consider a class that has a method get\_data. This method does an external lookup (on a database or web API, for example), and various other methods in the class call it. However, in a unit test, you don't want to depend on the external data source - so you dynamically replace the get\_data method with a stub that returns some fixed data.

PYTHONPATH: It is an environment variable which is used when a module is imported. Whenever a module is imported, PYTHONPATH is also looked up to check for the presence of the imported modules in various directories. The interpreter uses it to determine which module to load.

**Python pickle is one that transforms any Python object into a string representation. Utilizing the dump function, it copies the Python object to a file; this cycle is called Pickling.**

**Unpickling is the procedure of obtaining the original Python objects from the stored string representation.**

## ****Python File Open****

Before performing any operation on the file like reading or writing, first, we have to open that file. For this, we should use Python’s inbuilt function [open()](https://www.geeksforgeeks.org/python-open-function/) but at the time of opening, we have to specify the mode, which represents the purpose of the opening file.

Where the following mode is supported:

1. **r:**open an existing file for a read operation.
2. **w:** open an existing file for a write operation. If the file already contains some data, then it will be overridden but if the file is not present then it creates the file as well.
3. **a:**open an existing file for append operation. It won’t override existing data.
4. **r+:**  To read and write data into the file. The previous data in the file will be overridden.
5. **w+:** To write and read data. It will override existing data.
6. **a+:** To append and read data from the file. It won’t override existing data.

## ****What is Deep copy in Python?****

A deep copy creates a new compound object before inserting copies of the items found in the original into it in a recursive manner. It means first constructing a new collection object and then recursively populating it with copies of the child objects found in the original. In the case of deep copy, a copy of the object is copied into another object. It means that **any changes** made to a copy of the object**do not reflect in the original object.**

## ****What is Shallow copy in Python?****

A shallow copy creates a new compound object and then references the objects contained in the original within it, which means it constructs a new collection object and then populates it with references to the child objects found in the original. The copying process does not recurse and therefore won’t create copies of the child objects themselves. In the case of shallow copy, a reference of an object is copied into another object. It means that **any changes** made to a copy of an object **do reflect** in the original object. In python, this is implemented using the “**copy()**” function.

Python descriptors are created to manage the attributes of different classes which use the object as reference. In descriptors we used three different methods that are \_\_getters\_\_(), \_\_setters\_\_(), and \_\_delete\_\_(). If any of those methods are defined for an object, it can be termed as a descriptor.

## Using the LEGB Rule for Python Scope

Python resolves names using the so-called **LEGB rule**, which is named after the Python scope for names. The letters in LEGB stand for Local, Enclosing, Global, and Built-in.

* **Local (or function) scope** is the code block or body of any Python function or [lambda](https://realpython.com/python-lambda/) expression. This Python scope contains the names that you define inside the function. These names will only be visible from the code of the function. It’s created at function call, not at function definition, so you’ll have as many different local scopes as function calls. This is true even if you call the same function multiple times, or [recursively](https://realpython.com/python-recursion/). Each call will result in a new local scope being created.
* **Enclosing (or nonlocal) scope** is a special scope that only exists for nested functions. If the local scope is an [inner or nested function](https://realpython.com/inner-functions-what-are-they-good-for/), then the enclosing scope is the scope of the outer or enclosing function. This scope contains the names that you define in the enclosing function. The names in the enclosing scope are visible from the code of the inner and enclosing functions.
* **Global (or module) scope** is the top-most scope in a Python program, script, or module. This Python scope contains all of the names that you define at the top level of a program or a module. Names in this Python scope are visible from everywhere in your code.
* **Built-in scope** is a special Python scope that’s created or loaded whenever you [run a script](https://realpython.com/run-python-scripts/) or open an interactive session. This scope contains names such as [keywords](https://realpython.com/python-keywords/), functions, [exceptions](https://realpython.com/python-exceptions/), and other attributes that are built into Python. Names in this Python scope are also available from everywhere in your code. It’s automatically loaded by Python when you run a program or script.

An important thing to note here, is that we can get to know what variables are stored inside a Closure with the help of the **\_\_closure\_\_** attribute that makes use of Cell Objects to store the variables of the Outer Function and because of this, the closure can use these variables even when the Outer Function is terminated.

### **When and Why to Use Closures**

1. As Python closures are used as callback functions, they provide some sort of data hiding. This helps us to reduce the use of global variables.
2. When we have few functions in our code, closures in Python prove to be an efficient way. But if we need to have many functions, then go for class (OOP).
3. We may have variables in the global scope that are not used by many functions at times. Instead of defining variables in global scope, consider using a closure. They can be defined in the outer function and used in the inner function. Python Closures are also useful for avoiding the use of a global scope.
4. A class in the Python programming language always has the \_\_init\_\_ method. If you only have one extra method, an elegant solution would be to use a closure rather than a class. Because this improves code readability and even reduces the programmer’s workload. Closures in Python can thus be used to avoid the needless use of a class.

\_\_slots\_\_ is a class variable that is usually assigned a sequence of strings that are variable names used by instances.

## Why Use `\_\_slots\_\_`?

The short answer is slots are more efficient in terms of memory space and speed of access, and a bit safer than the default Python method of data access. By default, when Python creates a new instance of a class, it creates a \_\_dict\_\_ attribute for the class. The \_\_dict\_\_ attribute is a dictionary whose keys are the variable names and whose values are the variable values. This allows for dynamic variable creation but can also lead to uncaught errors. For example, with the default \_\_dict\_\_, a misspelled variable name results in the creation of a new variable, but with \_\_slots\_\_ it raises in an [AttributeError](https://wiki.python.org/moin/AttributeError).

**In Python, the things that are occurring simultaneously are called by different names, including these:**

* **Thread**
* **Task**
* **Process**

**In contrast,**[**threads**](https://en.wikipedia.org/wiki/Thread_(computing))**and**[**asynchronous tasks**](https://en.wikipedia.org/wiki/Asynchrony_(computer_programming))**always run on a single processor, which means they can only run one at a time.**

**The Python Global Interpreter Lock or**[**GIL**](https://wiki.python.org/moin/GlobalInterpreterLock)**, in simple words, is a mutex (or a lock) that allows only one**[**thread**](https://realpython.com/intro-to-python-threading/)**to hold the control of the Python interpreter.**

**This means that only one thread can be in a state of execution at any point in time. The impact of the GIL isn’t visible to developers who execute single-threaded programs, but it can be a performance bottleneck in CPU-bound and multi-threaded code.**

**Python uses reference counting for**[**memory management**](https://realpython.com/python-memory-management/)**. It means that objects created in Python have a reference count variable that keeps track of the number of references that point to the object. When this count reaches zero, the memory occupied by the object is released.**

**Let’s take a look at a brief code example to demonstrate how reference counting works:**

**Python**

**>>> import sys**

**>>> a = []**

**>>> b = a**

**>>> sys.getrefcount(a)**

**3**

**In the above example, the reference count for the empty list object [] was 3. The list object was referenced by a, b and the argument passed to sys.getrefcount().**

**Back to the GIL:**

**The problem was that this reference count variable needed protection from race conditions where two threads increase or decrease its value simultaneously. If this happens, it can cause either leaked memory that is never released or, even worse, incorrectly release the memory while a reference to that object still exists. This can cause crashes or other “weird” bugs in your Python programs.**

**This reference count variable can be kept safe by adding *locks* to all data structures that are shared across threads so that they are not modified inconsistently.**

**But adding a lock to each object or groups of objects means multiple locks will exist which can cause another problem—Deadlocks (deadlocks can only happen if there is more than one lock). Another side effect would be decreased performance caused by the repeated acquisition and release of locks.**

**The GIL is a single lock on the interpreter itself which adds a rule that execution of any Python bytecode requires acquiring the interpreter lock. This prevents deadlocks (as there is only one lock) and doesn’t introduce much performance overhead. But it effectively makes any CPU-bound Python program single-threaded**

**The GIL, although used by interpreters for other languages like Ruby, is not the only solution to this problem. Some languages avoid the requirement of a GIL for thread-safe memory management by using approaches other than reference counting, such as garbage collection.**

**On the other hand, this means that those languages often have to compensate for the loss of single threaded performance benefits of a GIL by adding other performance boosting features like JIT compilers.**