**Import module in python:**

Import in python is similar to #include header\_file in C/C++

***Syntax:*** *import module\_name*

*e.g. import math*

*math.sqrt()*

*OR*

*import module\_name as any\_name*

*e.g import math as m*

*m.sqrt*

*OR*

*from module\_name import\**

*e.g. from math import\* pi*

*OR*

*from module\_name import member\_name*

*e.g from math import pi*

When import is used, it searches for the module initially in the local scope by calling \_\_import\_\_() function. The value returned by the function are then reflected in the output of the initial code.

**Need of compilation in python**

Python is an interpreted programming language. It doesn’t require compilation to run. However, there’re situations we want to hide the python code and deploy the executable only.

One way to do this is to compile the python py file into pyc file, and deploy the pyc file.

**Compile .py file in python:**

1.Create a compiled version of hello\_world.py

*$python -c "import py\_compile;py\_compile.compile('hello\_world.py')"*

*OR*

*$ python -m py\_compile helloworld.py.*

*OR*

import py\_compile

*py\_compile.compile('test.py')*

We can use the **file** command to see the file type of helloworld.pyc

2.Creating a compiled version of file (bytecode) while importing module

Python first compiles your *source code* (the statements in your file) into a format known as *byte code*. Compilation is simply a translation step, and byte code is a lower-level, and platform-independent, representation of your source code. Roughly, each of your source statements is translated into a group of byte code instructions. This byte code translation is performed to speed execution—byte code can be run much quicker than the original source code statements.

The **bytecode** (.pyc file) is loaded into the Python runtime and interpreted by a **Python Virtual Machine**, which is a piece of code that reads each instruction in the **bytecode** and executes whatever operation is indicated.

# **Why Do We Need Logging?**

# Without proper logging we have no real idea as to why our applications fails and no real recourse for fixing these applications.

# e.g. Imagine you were working on an incredibly important application that your company relied upon in order to generate income. Now imagine that somehow, at say 3am in the morning on a Saturday, your application has fallen over and the company stops generating income. In this scenario logging is your best ally in finding out what went wrong. The very first thing you’ll do when you log in is to check the error logs for your application and ascertain exactly what and where failed. In this particular instance you quickly spot that this is a memory issue and that you need to increase the amount of RAM on the machine running your system. You quickly start up a new AWS EC2 instance with slightly more RAM and deploy your app and suddenly your company is back online.

### **When to use logging?**

Logging provides a set of convenience functions for simple logging usage. These are [debug()](https://docs.python.org/2/library/logging.html" \l "logging.debug), [info()](https://docs.python.org/2/library/logging.html" \l "logging.info), [warning()](https://docs.python.org/2/library/logging.html" \l "logging.warning), [error()](https://docs.python.org/2/library/logging.html" \l "logging.error) and [critical()](https://docs.python.org/2/library/logging.html" \l "logging.critical). To determine when to use logging, see the table below, which states, for each of a set of common tasks, the best tool to use for it.

| **Task you want to perform** | **The best tool for the task** |
| --- | --- |
| Display console output for ordinary usage of a command line script or program | [print()](https://docs.python.org/2/library/functions.html" \l "print) |
| Report events that occur during normal operation of a program (e.g. for status monitoring or fault investigation) | [logging.info()](https://docs.python.org/2/library/logging.html" \l "logging.info) (or [logging.debug()](https://docs.python.org/2/library/logging.html" \l "logging.debug) for very detailed output for diagnostic purposes) |
| Issue a warning regarding a particular runtime event | [warnings.warn()](https://docs.python.org/2/library/warnings.html" \l "warnings.warn) in library code if the issue is avoidable and the client application should be modified to eliminate the warning  [logging.warning()](https://docs.python.org/2/library/logging.html" \l "logging.warning) if there is nothing the client application can do about the situation, but the event should still be noted |
| Report an error regarding a particular runtime event | Raise an exception |
| Report suppression of an error without raising an exception (e.g. error handler in a long-running server process) | [logging.error()](https://docs.python.org/2/library/logging.html" \l "logging.error), [logging.exception()](https://docs.python.org/2/library/logging.html" \l "logging.exception) or [logging.critical()](https://docs.python.org/2/library/logging.html" \l "logging.critical) as appropriate for the specific error and application domain |

The standard levels and their applicability are described below (in increasing order of severity):

| **Level** | **When it’s used** |
| --- | --- |
| DEBUG | Detailed information, typically of interest only when diagnosing problems. |
| INFO | Confirmation that things are working as expected. |
| WARNING | An indication that something unexpected happened, or indicative of some problem in the near future (e.g. ‘disk space low’). The software is still working as expected. |
| ERROR | Due to a more serious problem, the software has not been able to perform some function. |
| CRITICAL | A serious error, indicating that the program itself may be unable to continue running. |

**NOTES:**

The default level is WARNING, which means that only events of this level and above will be tracked, unless the logging package is configured to do otherwise.

Events that are tracked can be handled in different ways. The simplest way of handling tracked events is to print them to the console. Another common way is to write them to a disk file.