# CSC104 LECTURES NOTES

Python **incorporates** the principles of the philosophy that **complex tasks** can be **done in simple** ways.

Python was created to have an extremely fast and simple **learning curve** and development process. As a result, it is considered the **most general-purpose programming language** since users can work in almost any study domain and still be able to find a useful piece of code for themselves.

In Python, **a group of programs** for performing various tasks makes up a **module (package)**. At the time of writing, there are over **117,181** modules that have been submitted by an even larger number of developers around the world.

Python is a ***multipurpose, portable, object-oriented, high-level programming* language** that enables an ***interactive environment*** to code in a minimalistic way.

**High-Level Programming**: provides facilities like library functions for performing the low-level tasks and also provides ways to define the code in a form readable to humans, which is then translated into machine language to be fed to a processor. A **low-level language** is one where users directly feed the machine code to a processor for obtaining results.

**Interactive environment**: To a large extent, Python derives its philosophy from the **ABC language**. The syntax structure was largely derived from C and UNIX's Bourne shell environments. The interpretive nature means that Python presents a REPL-based interactive environment to a developer. The interactive shell of the Python programming language is commonly known as REPL (Read-Evaluate-Print-Loops) because it

- reads what a user types,
- evaluates what it reads,
- prints out the return value after evaluation, and
- loops back and does it all over again.

**Object Orientation**: Most primitive programming languages were procedural in nature. An object-oriented programming (OOP) language deals with data as an object on which different methods act to produce a desired result. Everything computable is treated as an object. Its nature is defined as its properties. These properties can be probed by functions, which are called methods.

**Multipurpose Nature**: Python enables developers from different walks of life to use and enrich the language in their fields of expertise. Virtually all fields of

computations have used Python. You can define a module specific for one kind of problem. In fact, Python modules exist for specific fields of studies, as shown in Table below; It is impossible to list all the modules for a given application as the modules are being created on a daily basis

***Table 1-1.*** *List of Fields of Study and Corresponding Python Modules*

| Field of Study | Name of Python Module |
|---|---|
| Scientific Computation | `scipy, numpy, sympy` |
| Statistics | `pandas` |
| Networking | `networkx` |
| Cryptography | `pyOpenSSL` |
| Game Development | `PyGame` |
| Graphic User Interface | `pyQT` |
| Machine Learning | `scikit-learn, tensorflow` |
| Image Processing | `scikit-image` |
| Plotting | `Matplotlib` |
| Database | `SQLAlchemy` |
| HTML and XML parsing | `BeautifulSoup` |
| Natural Language Processing | `nltk` |
| Testing | `nose` |

**Minimalistic Design**: In Python means that it emphasizes code readability. It also provides a syntax structure that allows programmers to express concepts in fewer lines of code than in languages such as C++ and Java.

The core philosophy of the language is summarized by Tim Peters in the document The Zen of Python, which includes the following aphorisms:

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- Errors should never pass silently.
- Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.

- There should be one— and preferably only one —obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than right now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea—let's do more of those!

**Portability**: Since Python belongs to an open-source community, it has been ported (that is, adapted to work on) to many platforms so that Python code written on one platform can run without modification on others (except system-dependent features).

**Extensibility**: Rather than providing all functionalities in its core program, Python's creators designed it to be highly extensible. Users can thus choose to have functionality as per their requirements. For example, if a user needs to work on differential equations, then that user can use a module for differential equations rather than all users having that functionality but never using it.

## HISTORY

The development of the Python programming language dates back to the 1980s. Guido van Rossum at CWI in the Netherlands began implementing it in December 1989. This was an era when computing devices were becoming increasingly powerful and reliable with every advancing day. Van Rossum named the language after the BBC TV show Monty Python's Flying Circus. Python 1.0 was released to the public in 1994, Python 2.0 in 2000, and Python 3.0 in 2008. Howver, Python 3 was not created to be backward compatible with Python 2, which made it less practical for users who were already developing with Python 2. As a result, a lot of developers have continued using Python 2, even now. Nonetheless, the future belongs to Python 3, which has been developed in a more efficient manner. Hence, we will discuss Python-3-based codes in this Course.

## PYTHON AND ENGINEERING

Engineering problems employ numerical computations both on a small scale and on a large scale. Thus, engineering applications require a programming language to fit well in both these regimes. There are very few languages that can boast this quality, so Python is definitely a winner here.

Python, being an interpretative language, is generally considered to be a slower option in regards to running large computational tasks on bigger computational architectures, but its ability to use faster codes written in C, Java, and Fortran using the interlinking packages cython, jython, and f2p allows speed-intensive tasks to be run in their native language within a Python code.

Another engineering concern is the ability of a programming language to communicate with physical devices efficiently. A variety of microcontrollers allow Python to run its hardware with ease. MicroPython [6] is specially designed for this purpose. MicroPython is a lean and efficient implementation of the Python 3 programming language.

Users of MATLAB argue that Simulink is one of the easiest ways of prototyping and simulating an engineering problem because they don't need to code. Instead, users just stitch together pieces of codes represented by graphical blocks on a graphics terminal. Python still lacks this ability for now.

**What is a program?**

A program is a sequence of instructions that specifies how to perform a computation. The computation might be something mathematical, such as solving a system of equations