Python Programming – I (Introduction)

- By Nimesh Kumar Dagur, CDAC Noida, India

Introduction to Python programming

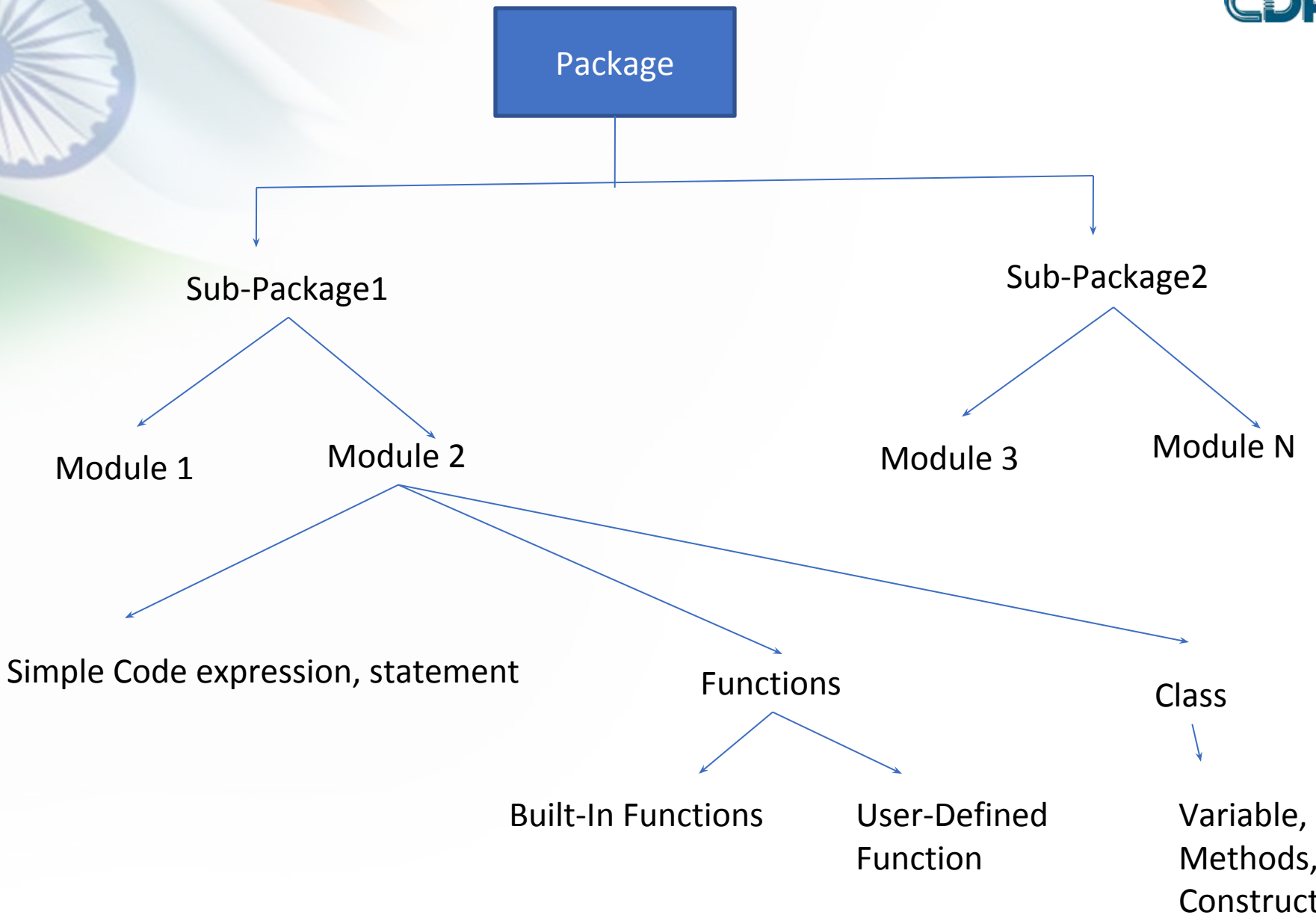
- **Python** is a widely used general-purpose, high-level programming language.
- Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C.
- Python supports multiple programming paradigms, including object-oriented and functional programming or procedural styles.
- Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Introduction to Python programming

- Python is Interpreted
- Python is Interactive
- Python is Object-Oriented
- Python is Beginner's Language

Python's feature

- **Easy-to-learn:** relatively few keywords, simple structure, and a clearly defined syntax
- **Easy-to-read:** much more clearly defined and visible to the eyes
- **Easy-to-maintain:** source code is fairly easy-to-maintain. (**Module & package**)
- **A broad standard library:** bulk of the library is very portable and cross-platform compatible on UNIX, Windows and Macintosh
- **Portable :** run on a wide variety of hardware platforms and has the same interface on all platforms.



Python's feature

- **Extendable:** You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases:** Python provides interfaces to all major commercial databases.
- **GUI Programming:** Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems.
- **Scalable:** Python provides a better structure and support for large programs.

Installation

python.org/downloads/

Python | PSF | Docs | PyPI | Jobs | Community

python™

Donate | Search | GO | Socialize

About | Downloads | Documentation | Community | Success Stories | News | Events

Download the latest version for Windows

Download Python 3.13.7

Looking for Python with a different OS? Python for Windows, Linux/Unix, macOS, Android, other

Want to help test development versions of Python 3.14? Pre-releases, Docker images

Active Python Releases

For more information visit the Python Developer's Guide.

Python version	Maintenance status	First released	End of support	Release schedule
3.14	pre-release	2025-10-01 (planned)	2030-10	PEP 745
3.13	bugfix	2024-10-07	2029-10	PEP 719
3.12	security	2023-10-02	2028-10	PEP 693
3.11	security	2022-10-24	2027-10	PEP 664
3.10	security	2021-10-04	2026-10	PEP 619
3.9	security	2020-10-05	2025-10	PEP 596

Running Python:

There are three different ways to start Python:

(1) Interactive Interpreter: You can enter python and start coding right away in the interactive interpreter by starting it from the command line.

C:>python

(2) Script from the Command-line: A Python script can be executed at command line by invoking the interpreter on your application, as in the following:

C:>python script.py

(3) Integrated Development Environment : PythonWin is the first Windows interface for Python and is an IDE with a GUI.

Python Identifiers:

- A Python identifier is a name used to identify a variable, function, class, module or other object.
- An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
- Python does not allow punctuation characters such as @, $ and % within identifiers.
- Python is a case sensitive programming language. Thus, **Manpower and manpower are two different identifiers in Python.**

Reserved Words:

- Reserved words may not be used as constant or variable or any other identifier names.
- All the Python keywords contain lowercase letters only.

Reserved Words:

```
>>> help("keywords")
```

False	def	if	raise
None	del	import	return
True	elif	in	try
and	else	is	while
as	except	lambda	with
assert	finally	nonlocal	yield
break	for	not	
class	from	or	
continue	global	pass	

Lines and Indentation:

- One of the first caveats programmers encounter when learning Python is the fact that there are no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.
- The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount.

```
if True:
    print("True")
else:
  print("False")
```

Multi-Line Statements

- Statements in Python typically end with a new line.
- Python does, however, allow the use of the line continuation character (\) to denote that the line should continue.

```
total = item_one + \
        item_two + \
        item_three
```

Multi-Line Statements

- Statements contained within the [], {} or () brackets do not need to use the line continuation character

```
days = ['Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday']
```

Quotation in Python:

- Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.
- The triple quotes can be used to span the string across multiple lines.

```
word = 'word'
sentence = "This is a sentence."
paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""
```

Comments in Python:

- A hash sign (#) that is not inside a string literal begins a comment.
- All characters after the # and up to the physical line end are part of the comment and the Python interpreter ignores them.

```
print ("Hello, Python!");  # second comment
```

This will produce the following result:

```
Hello, Python!
```

A comment may be on the same line after a statement or expression:

```
name = "Madisetti" # This is again comment
```

You can comment multiple lines as follows:

```
# This is a comment.
# This is a comment, too.
# This is a comment, too.
# I said that already.
```

Python variable

- Variables are nothing but reserved memory locations to store values.
- when you create a variable you reserve some space in memory.
- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.
- Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Assigning Values to Variables:

- Python variables do not have to be explicitly declared to reserve memory space.
- The declaration happens automatically when you assign a value to a variable.
- The equal sign (=) is used to assign values to variables.

Assigning Values to Variables:

```
counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = "John"       # A string

print (counter)
print (miles)
print (name)
```

```
100
1000.0
John
```

Multiple Assignment:

- Python allows you to assign a single value to several variables simultaneously

```
a = b = c = 1
```

- Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location.
- You can also assign multiple objects to multiple variables.

```
a, b, c = 1, 2, "john"
```

Standard Data Types:

- The data stored in memory can be of many types.
- For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters.
- Python has various standard types that are used to define the operations possible on them and the storage method for each of them.

Standard Data Types:

Python has following standard data types:

- Numbers
- String
- List
- Tuple
- Dictionary
- set

Working With Numbers

- The interpreter acts as a simple calculator: you can type an expression at it and it will write the value.
- Expression syntax is straightforward:
- operators +, -, * and / work just like in most other languages (for example, C);
- parentheses can be used for grouping.

Examples

```
>>> 2+2
4
>>> # This is a comment
... 2+2
4
>>> 2+2  # and a comment on the same line as code
4
>>> (50-5*6)/4

5
>>> # Integer division returns the floor:
... 7/3
2
>>> 7/-3
-3
```

Python Numbers

Python supports following different numerical types

- **int (signed integers)**: positive or negative whole numbers with no decimal point.

- **long (long integers)**: integers of unlimited size, written like integers and followed by an uppercase or lowercase L. It is not supported in Python 3.x version.

- **float (floating point real values)** :represent real numbers and are written with a decimal point dividing the integer and fractional parts.
- Floats may also be in scientific notation, with E or e indicating the power of 10 (2.5e2 = 2.5 x 10^2 = 250).

- **complex (complex numbers)** : are of the form a + bJ, where a and b are floats and J (or j) represents the square root of -1 (which is an imaginary number).
- The real part of the number is a, and the imaginary part is b.

Working With Numbers

- The equal sign ('=') is used to assign a value to a variable. Afterwards, no result is displayed before the next interactive prompt:

```
>>> width = 20
>>> height = 5*9
>>> width * height
900
```

Working With Numbers

- A value can be assigned to several variables simultaneously:

```
>>> x = y = z = 0  # Zero x, y and z
>>> x
0
>>> y
0
>>> z
0
```

Working With Numbers

Variables must be “defined” (assigned a value) before they can be used, or an error will occur:

```
>>> # try to access an undefined variable
... n
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'n' is not defined
```

Working With Numbers

- There is full support for floating point; operators with mixed type operands convert the integer operand to floating point:

```
>>> 3 * 3.75 / 1.5
7.5
>>> 7.0 / 2
3.5
```

a+bj

0+1j

Working With Numbers

- Complex numbers are also supported;
- imaginary numbers are written with a suffix of j or J.
- Complex numbers with a nonzero real component are written as ***(real+imagj),*** or can be created with the ***complex(real, imag)*** function.

```
>>> 1j * 1J
(-1+0j)
>>> 1j * complex(0,1)
(-1+0j)
>>> 3+1j*3
(3+3j)
>>> (3+1j)*3
(9+3j)
>>> (1+2j)/(1+1j)
(1.5+0.5j)
```

Working With Numbers

- Complex numbers are always represented as two floating point numbers, the real and imaginary part.
- To extract these parts from a complex number *z, use* ***z.real*** *and* ***z.imag****.*

```
>>> a=1.5+0.5j
>>> a.real
1.5
>>> a.imag
0.5
```

Working With Numbers

- The conversion functions to floating point and integer (float(), int() and long()) don't work for complex numbers — there is no one correct way to convert a complex number to a real number.
- Use **abs(z)** to get its magnitude (as a float) or ***z.real*** to get its real part.

```
>>> a=3.0+4.0j
>>> float(a)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: can't convert complex to float; use abs(z)
>>> a.real
3.0
>>> a.imag
4.0
>>> abs(a)   # sqrt(a.real**2 + a.imag**2)
5.0
```

Mathematical Functions

- **abs() :** The method **abs()** returns absolute value of **x** - the (positive) distance between x and zero.

Syntax

```
abs( x )
```

Example

```
print "abs(-45) : ", abs(-45)
print "abs(100.12) : ", abs(100.12)
print "abs(119L) : ", abs(119L)
```

```
abs(-45) :  45
abs(100.12) :  100.12
abs(119L) :  119
```

Mathematical Functions

- **ceil()** : The method **ceil()** returns ceiling value of **x** - the smallest integer not less than x.

Syntax

```
import math

math.ceil( x )
```

x is a numeric expression.

Example

```
import math   # This will import math module

print "math.ceil(-45.17) : ", math.ceil(-45.17)
print "math.ceil(100.12) : ", math.ceil(100.12)
print "math.ceil(100.72) : ", math.ceil(100.72)
print "math.ceil(119L) : ", math.ceil(119L)
print "math.ceil(math.pi) : ", math.ceil(math.pi)
```

```
math.ceil(-45.17) :  -45.0
math.ceil(100.12) :  101.0
math.ceil(100.72) :  101.0
math.ceil(119L) :  119.0
math.ceil(math.pi) : 4.0
```

Mathematical Functions

- **floor()** : The method **floor()** returns floor of **x** - the largest integer not greater than x.

```
import math

math.floor( x )
```

Example

```
import math   # This will import math module

print "math.floor(-45.17) : ", math.floor(-45.17)
print "math.floor(100.12) : ", math.floor(100.12)
print "math.floor(100.72) : ", math.floor(100.72)
print "math.floor(119L) : ", math.floor(119L)
print "math.floor(math.pi) : ", math.floor(math.pi)
```

```
math.floor(-45.17) :  -46.0
math.floor(100.12) :  100.0
math.floor(100.72) :  100.0
math.floor(119L) :  119.0
math.floor(math.pi) :  3.0
```

Mathematical Functions

max(x1, x2,...)	The largest of its arguments: the value closest to positive infinity
min(x1, x2,...)	The smallest of its arguments: the value closest to negative infinity
pow(x, y)	The value of x**y.
round(x [,n])	**x** rounded to n digits from the decimal point. Python rounds away from zero as a tie-breaker: round(0.5) is 1.0 and round(-0.5) is -1.0.
sqrt(x)	The square root of x for x > 0