

PMDS508L - Python Programming

Dr. B.S.R.V. Prasad
Department of Mathematics
School of Advanced Sciences
Vellore Institute of Technology
Vellore



Python



srvprasad.bh@gmail.com (Personal)



srvprasad.bh@vit.ac.in (Official)



+91-8220417476

Data Science Techniques



Introduction to

Python



What is Python?



- ▶ Python is a general purpose interpreted interactive object oriented and high level programming language
- ▶ It was created by Guido van Rossum, a Dutch computer programmer in early 1990s and released in 1991
- ▶ Named after Monty Python
- ▶ Python supports cross platform development and is available through **open source**
 - ▶ Python comes pre-installed with Mac OS X and Linux
 - ▶ Windows binaries can be obtained from <https://python.org/>

What is Python?



- ▶ Python language places strong emphasis on code reliability and simplicity
- ▶ Python is multi-paradigm programming language
 - ▶ i.e., allows user to code in several different programming styles
- ▶ Python can be easily integrated with C/C++, Java etc.
 - ▶ CPython is Python integrated with C/C++ language
 - ▶ JPython is Python integrated with Java

Why Phthon?



- ▶ Good example of *scripting* language
- ▶ “Pythonic” style is very concise
 - ▶ Most programs in Python require considerably less number of lines of code to perform the same task compared to other languages like C
- ▶ Powerful but unobstructive object system
 - ▶ Every value is an object
 - ▶ When compared to C, C++, Java etc., Python has simplest Syntax
- ▶ Powerful collection and iteration abstractions
 - ▶ Python comes with an extensive collection of third party resources that extend the capabilities of the language
 - ▶ Dynamic typing makes coding life easy

Dynamic Typing - The Key Difference



▶ Java

- ▶ Variables are declared to refer to objects of a given type
- ▶ Methods use type signatures to enforce contracts

▶ Python

- ▶ Variables come into existence when first assigned to
- ▶ A variable can refer to an object of any type
- ▶ All types are (almost) treated the same way
- ▶ **Main drawback:** type errors are caught only at runtime

Why Python?



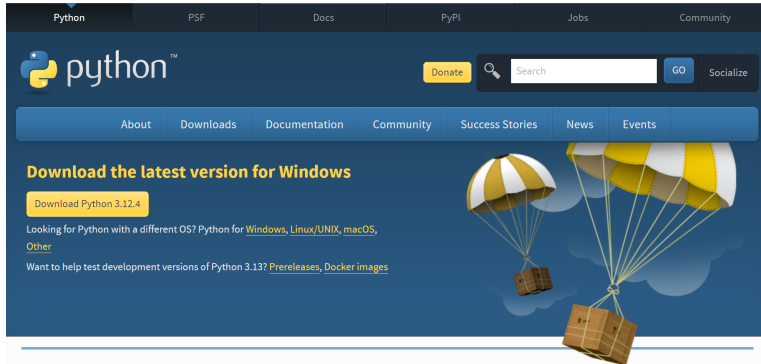
- ▶ Python can be used for large variety of tasks like
 - ▶ Desktop Applications
 - ▶ Database Applications
 - ▶ Network Programming
 - ▶ Game Programming
 - ▶ Mobile Development, etc.
- ▶ Python is a cross platform language
 - ▶ means - Code written for one operating system like Windows, will work equally well with Linux or MacOS without any changes to Python code.

Installing Python on Your PC



- ▶ Python Interpreter can be downloaded from <https://python.org/>
- ▶ There are two flavours of Python available
 - ▶ Python 2.7.x and Python 3.x
 - ▶ Python 3.x is the advanced version of Python 2.x
 - ▶ Many old generation programs are written in Python 2.7.x
 - ▶ Backward compatibility issues and Syntax differences
 - ▶ Python 3.x is the newer version which supports modern techniques like AI, machine learning, and data science
- ▶ Anaconda (<https://anaconda.com>) is a Python programming suite which includes several tools required for Data Science Platform

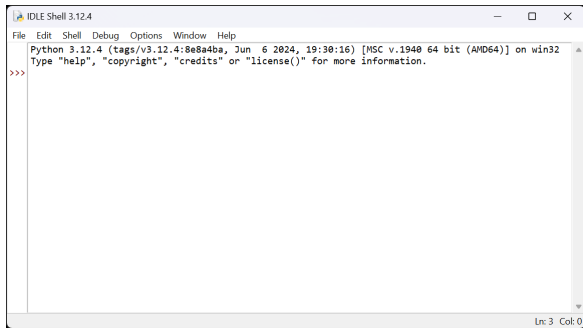
Installing Python on Your PC



Working with Python



- ▶ The Python shell allows us to use Python in interactive mode.

A screenshot of a Python Shell window titled 'IDLE Shell 3.12.4'. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area displays the following text: 'Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun 6 2024, 19:30:16) [MSC v.1940 64 bit (AMD64)] on win32' followed by 'Type "help", "copyright", "credits" or "license()" for more information.' Below this, the prompt '>>>' is visible. The status bar at the bottom right shows 'Ln: 3 Col: 0'.

Working with Python - IDLE



- ▶ IDLE is a graphical user interface for doing Python development
- ▶ It is a standard and free part of the Python system
- ▶ It is usually referred to as an Integrated Development Environment (IDLE)
- ▶ One can write the Python code or script using the IDLE which is like an editor.
- ▶ While the Python shell allows the user to work in interactive mode, the IDLE allows to write the code and save it with a .py extension.

Ex. myProg.py

- ▶ Once the program is saved, it can be executed by clicking the Run module or F5.
- ▶ If there are no errors you observe the results in Python shell.

- ▶ IDLE is not only the editor for Python. Its the default editor comes with Python installation.
- ▶ There are advanced editors such as
 - ▶ Spyder (Default IDE for Anaconda Python Installation)
 - ▶ PyCharm
 - ▶ Atom
 - ▶ Jupyter notebook (IPython)

Python help???



To get “help” on any command/keyword/identifier in Python one can use

help()

Keywords and Identifiers in Python



12

- ▶ Keywords, Identifiers and Variables are the basic building blocks of Python programming.
- ▶ Python keyword is a unique programming term intended to perform some action.
- ▶ There are as many as 33 such keywords in Python, each serving a different purpose.
- ▶ Together, they build the vocabulary of the Python language.
- ▶ They represent the syntax and structure of a Python program.
- ▶ Since all of them are reserved, so you can't use their names for defining variables, classes or functions.

Keywords and Identifiers in Python



- ▶ To know the list of identifiers in Python one can issue the following commands in the Python shell

```
1 >>> help()  
2 help> keywords
```

- ▶ To exit from the help one can issue the command

```
1 help> quit
```

Keywords and Identifiers in Python



```
help> keywords
```

Here is a list of the Python keywords. Enter any keyword to get more help.

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

- ▶ Python Identifiers are user-defined names to represent a variable, function, class, module or any other object.
- ▶ If you assign some name to a programmable entity in Python, then it is nothing but technically called an identifier.

Python language has a set of rules for programmers to create meaningful identifiers.

Guidelines For Creating Identifiers In Python

1. To form an identifier, use a sequence of letters either in lowercase (a to z) or uppercase (A to Z).
However, you can also mix up digits (0 to 9) or an underscore (_) while writing an identifier.
 - ▶ Ex: shapeClass, shape_1, upload_shape_to_db – are all valid identifiers
2. **You can't use digits to begin an identifier name. It'll lead to the syntax error.**
 - ▶ Ex: oShape is incorrect, but shape1 is a valid identifier

Guidelines For Creating Identifiers in Python

3. Also, the Keywords are reserved, so you should not use them as identifiers.

```
1 >>> for = 1
2 SyntaxError: invalid syntax
3 >>> True = 1
4 SyntaxError: can't assign to keyword
```

4. Python Identifiers can also not have special characters
'.', '!', '@', '#', '\$', '%' in their formation. These symbols are forbidden.

```
1 >>> @index=1
2 SyntaxError: invalid syntax
3 >>> isPython?=True
4 SyntaxError: invalid syntax
```

Python Identifier - Rules



You can test whether an identifier is valid or not by using

```
1 >>> import keyword
2 >>> keyword.iskeyword("techbeamers")
3 False
4 >>> keyword.iskeyword("try")
5 True
```

Another way of verifying whether an identifier is valid is or not in Python 3 is by using **str.isidentifier()** command

```
1 >>> 'techbeamers'.isidentifier()
2 True
3 >>> '@techbeamers'.isidentifier()
4 False
5 >>> 'techbeamers_com'.isidentifier()
6 True
7 >>> 'techbeamers.com'.isidentifier()
8 False
```

Best Practices For Identifier Naming



- ▶ Better start the class names with a capital letter. All other identifiers should begin with a lowercase letter.
- ▶ Declare private identifiers by using the ('_') underscore as their first letter. Don't use '_' as the leading and trailing character in an identifier. As Python built-in types already use this notation.
- ▶ Avoid using names with only one character. Instead, make meaningful names. (Only a suggestion)
 - ▶ While `i = 1` is valid, but writing `iter = 1` or `index = 1` would make more sense.
- ▶ You can use underscore to combine multiple words to form a sensible name.
 - ▶ Ex: `count_no_of_letters`

Identifier Naming Conventions



There are certain naming conventions available across all programming languages, which are

- ▶ Snake case
- ▶ Kebab case
- ▶ Camel case
- ▶ Pascal case

Identifier Naming Conventions

Snake case



- ▶ Snake case separates each word with an underscore character (_).
- ▶ When using snake case, all letters need to be lowercase.
- ▶ Here are some examples of how we would use the snake case.
`number_of_students = 34`
`hello_phrase = "Hello World"`
- ▶ Snake case is used for creating variable and method names.
- ▶ Snake case is also a good choice for naming files, as it keeps names readable.
- ▶ We will typically encounter it the most when programming in Python and not so much when programming in Java, JavaScript, or TypeScript.
- ▶ There is also an all-caps version of the snake case where all letters are in the upper case - also known as the screaming snake case, which mostly used to declare constants. Example `NUMBER_OF_STUDENTS = 35`.

Identifier Naming Conventions

Kebab case



- ▶ The kebab case is very similar to snake case.
- ▶ The difference between snake case and kebab case is that kebab case separates each word with a dash character, —, instead of an underscore.
- ▶ So, all words are lowercase, and each word gets separated by a dash.
- ▶ Examples
 - `number-of-students = 34`
 - `hello-phrase = "Hello World"`
- ▶ The kebab case is mostly used in URLs.

Identifier Naming Conventions

Camel case



- ▶ In camel case, we start by making the first word lowercase. Then, you capitalize the first letter of each word that follows.
- ▶ So, a capital letter appears at the start of the second word and at each new subsequent word that follows it.
- ▶ Examples of camel case:
`numberOfStudents = 34`
`helloPhrase = "Hello World"`
- ▶ In the example `numberOfStudents`, the first word `number` is lowercase. Then, the first letter of the second word, `Of`, is capitalized, as is the first letter of the third word, `Students`.
- ▶ We will encounter camel case in Java, JavaScript, and TypeScript for creating variable, function, and method names.

Identifier Naming Conventions

Pascal case



- ▶ Pascal case is similar to camel case.
- ▶ The only difference between the two is that pascal case requires the first letter of the first word to also be capitalized.
- ▶ So, when using pascal case, every word starts with an uppercase letter (in contrast to camel case, where the first word is in lowercase).
- ▶ Some examples of pascal case:
 `NumberOfStudents = 34`
 `HelloPhrase = "Hello World"`
- ▶ We will see the pascal case used for naming classes in most programming languages.

- ▶ A variable in Python represents an entity whose value can change as and when required.
- ▶ Conceptually, it is a memory location which holds the actual value. We can retrieve the value from our code by querying the entity.
- ▶ But it requires assigning a label to that memory location so that we can reference it. We call it as a variable in the programming terms.
- ▶ **Variables names are case sensitive.**
userName is different from username

Key Facts and Rules for Variables in Python

1. Variables don't require declaration. However, you must initialize them before use.

```
1 >>> test = 100
```

2. The above expression will lead to the following

- ▶ Creation of an object to represent the value 100.
- ▶ If the variable 'test' doesn't exist, then it will be created.
- ▶ Association of the variable with the object, so that it can refer the value.

The variable 'test' is a reference to the value '10'. Please refer to the illustration shown below.

		--		----	****
(test)	--	Reference		----	** 10 **
		--		----	****
Variable	--			----	Object

Key Facts and Rules for Variables in Python

3. Whenever the expression changes, Python associates a new object (a chunk of memory) to the variable for referencing that value. And the old one goes to the garbage collector.

```
1 >>> test = 100
2 >>> id(test)
3 9302464
4 >>> test = 110
5 >>> id(test)
6 9302496
```

4. Also, for optimization, Python builds a cache and reuses some of the immutable objects, such as small integers and strings.

Key Facts and Rules for Variables in Python

5. It's the object which has a type, not the variable. However, a variable can hold objects of different types as and when required.

```
1 >>> test = 100
2 >>> type(test)
3 <class 'int'>
4 >>> test = 10.2
5 >>> type(test)
6 <class 'float'>
7 >>> test = 'VIT'
8 >>> type(test)
9 <class 'str'>
10 >>> test = {'C', 'C++', 'Python'}
11 >>> type(test)
12 <class 'set'>
13 >>> test = ['C', 'C++', 100]
14 >>> type(test)
15 <class 'list'>
```

Python 'print()' command



29

To display some message or variable value in Python we will use **'print()'** command:

```
1 print('Hello World!')
```

```
1 x = 5
2 y = 'Hello World!'
3 print(x, y)
```

A Code Sample



```
1 x = 30 - 25                # A comment
2 y = "Hello"               # Another comment
3 z = 1.34
4 if z == 1.34 or y == "Hello":
5     x = x + 1
6     y = y + " World!"      # String Concatenation
7 print(x)
8 print(y)
```


Enough to Understand the Code



31

- ▶ Assignment used `=` and comparison uses `==`.
- ▶ For numbers `+`, `-`, `*`, `/`, `%` are as has usual meaning.
 - ▶ Special use of `+` for string concatenation.
 - ▶ Special use of `%` for string formatting (as with `printf` in C)
- ▶ Logical operators are words (`and`, `or`, `not`) but *not* symbols.
- ▶ The basic printing command is `print`.
- ▶ The first assignment to a variable creates it.
 - ▶ Variable types don't need to be declared.
 - ▶ Python figures out the variables types on its own.

Python Syntax - Indentation



- ▶ Indentation refers to the spaces at the beginning of a code line.
- ▶ Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.
- ▶ Python uses indentation to indicate a block of code.

```
1 if 5 > 2:  
2     print("Five is greater than two!")
```

Python Syntax - Indentation



- ▶ Python will give you an error if you skip the indentation:

```
1 if 5 > 2:  
2 print("Five is greater than two!")
```

- ▶ The number of spaces is up to you as a programmer, but it has to be at least one.

```
1 if 5 > 2:  
2     print("Five is greater than two!")  
3 if 5 > 2:  
4     print("Five is greater than two!")
```

- ▶ You have to use the same number of spaces in the same block of code, otherwise Python will give you an error:

```
1 if 5 > 2:  
2     print("Five is greater than two!")  
3         print("Five is greater than two!")
```

Comments in Python



Comments starts with a #, and Python will ignore them:

- ▶ Comments can be used to explain the Python code and more readable
- ▶ Comments can be used to prevent execution when testing code

```
1 #This is a comment
2 print("Hello, World!")
```

```
1 print("Hello, World!") #This is a comment
```

```
1 #print("Hello, World!")
2 print("Hellooo, Mates!")
```

Multiline Comments



Python will ignore string literals that are not assigned to a variable. So you can add a multiline string (triple quotes) in your code, and place your comment inside it:

```
1 """  
2 This is a comment  
3 written in  
4 more than just one line  
5 """  
6 print("Hello, World!")
```

Simple Code to Illustrate the Assignment in Python



```
1 x = 5
2 y = 10
3 x = y
4 print("x= ", x)
5 print("y= ", y)
```

Save the above program as myProg1.py and run the program by either clicking Run Module or F5.

What is the output if we change the third line to

```
1 y = x
```

Assign Value to Multiple Variables



Python allows you to assign values to multiple variables in one line:

```
1 x, y, z = "Orange", "Banana", "Cherry"
2 print(x)
3 print(y)
4 print(z)
```

Another example:

```
1 x, y, z = 100
2 print(x)
3 print(y)
4 print(z)
```

Python has the following data types built-in by default, in these categories:

Text Type:	<code>str</code>
Numeric Types:	<code>int, float, complex</code>
Sequence Types:	<code>list, tuple, range</code>
Mapping Type:	<code>dict</code>
Set Types:	<code>set, frozenset</code>
Boolean Type:	<code>bool</code>
Binary Types:	<code>bytes, bytearray, memoryview</code>

`type()` command can be used to know the type of a data variable.

Numeric Types



```
1 x = 349          # int
2 y = 53.52        # float
3 z = -232.23e100   # float in scientific notation
4 w = 2j           # complex
5
6 print(type(x))
7 print(type(y))
8 print(type(z))
9 print(type(w))
10
```

Type Conversion



```
1 x = 1    # int
2 y = 2.8  # float
3 z = 1j   # complex
4
5 #convert from int to float:
6 a = float(x)
7
8 #convert from float to int:
9 b = int(y)
10
11 #convert from int to complex:
12 c = complex(x)
13
14 print(a)
15 print(b)
16 print(c)
17
18 print(type(a))
19 print(type(b))
20 print(type(c))
```

You cannot convert complex numbers into another number type.

Specify a Variable Type



- ▶ There may be times when you want to specify a type on to a variable. This can be done with casting.
- ▶ Casting in python is done using constructor functions:
 - ▶ **int()** - constructs an integer number from an integer literal, a float literal, or a string literal
 - ▶ **float()** - constructs a float number from an integer literal, a float literal or a string literal
 - ▶ **str()** - constructs a string from a wide variety of data types, including strings, integer literals and float literals

Specify a Variable Type



```
1 x = int(1)      # x will be 1
2 y = int(2.8)    # y will be 2
3 z = int("3")    # z will be 3
4
5 x = float(1)     # x will be 1.0
6 y = float(2.8)   # y will be 2.8
7 z = float("3")   # z will be 3.0
8 w = float("4.2") # w will be 4.2
9
10 x = str("s1")   # x will be 's1'
11 y = str(2)      # y will be '2'
12 z = str(3.0)    # z will be '3.0'
```

Strings



```
1 s = "Hello"
2 print(s)
3
4 # Multiline strings
5 s1 = """This is the example line.
6 Example of a Python strings
7 with multiple lines
8 This is the fourth line."""
9 print(s1)
10
11 s2 = "Hello, World!"
12 print(s2[1])
13 print(s2[2:5])
```

Strings



```
1 # Negative Indexing - To start the slice from the end
   of the string:
2 s2 = "Hello World!"
3 print(s2[-5:-2])
4
5 s = "Hello World!"
6 print(s.len())           # Length of a string
7 print(s.lower())         # Convert the string into lower
   case
8 print(s.upper())         # Convert the string into upper
   case
9 print(s.strip())         # Remove the white spaces in the
   string
```

Strings



```
1 s = "Hello World!"
2 print(s.replace("H","J")) # Replaces a string with
   another string
3 print(s.split(",")) # Splits the string using the given
   delimiter or character. In this example it returns
   ['Hello', 'World!']
4 s1 = "Good Evening..."
5 print(s+" "+s1)          # String concatenation
```

Booleans represent one of two values: **True** or **False**.

```
1 print(10 > 9)           # Returns True
2 print(10 == 9)          # Returns False
3 print(10 < 9)           # Return False
```


Python divides the operators in the following groups:

- ▶ Arithmetic operators
- ▶ Assignment operators
- ▶ Comparison operators
- ▶ Logical operators
- ▶ Identity operators
- ▶ Membership operators
- ▶ Bitwise operators

Arithmetic Operators



Python accepts all the basic operators like

- ▶ + (addition)
- ▶ − (subtraction)
- ▶ * (multiplication)
- ▶ / (division)
- ▶ // (floor division i.e., quotient)
- ▶ % (modulus) and
- ▶ ** exponent

Arithmetic Operators



49

```
1 x, y = 7, 2
2 print('x+y = ', x+y)
3 print('x-y = ', x-y)
4 print('x*y = ', x*y)
5 print('x/y = ', x/y)
6 print('x//y = ', x//y) #Floor division: rounds off the
   answer to the nearest whole number
7 print('x%y = ', x%y)
8 print('x**y = ', x**y)
```

Additional Assignment Operators



▶ `+=`:

```
1  x+=2
2  print(x)
3  x-=2
4  print(x)
```

▶ Similarly for other operators

Python Comparison Operators



Operator	Name	Example
<code>==</code>	Equal	<code>x == y</code>
<code>!=</code>	Not equal	<code>x != y</code>
<code>></code>	Greater than	<code>x > y</code>
<code><</code>	Less than	<code>x < y</code>
<code>>=</code>	Greater than or equal to	<code>x >= y</code>
<code><=</code>	Less than or equal to	<code>x <= y</code>

Python Logical Operators



Operator	Description	Example
and	Returns True if both statements are true	$x < 5$ and $x < 10$
or	Returns True if one of the statements is true	$x < 5$ or $x < 4$
not	Reverse the result, returns False if the result is true	$\text{not}(x < 5 \text{ and } x < 10)$

Python Identity Operators



Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example
is	Returns true if both variables are the same object	x is y
is not	Returns true if both variables are not the same object	x is not y

Python Membership Operators



Membership operators are used to test if a sequence is presented in an object:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y