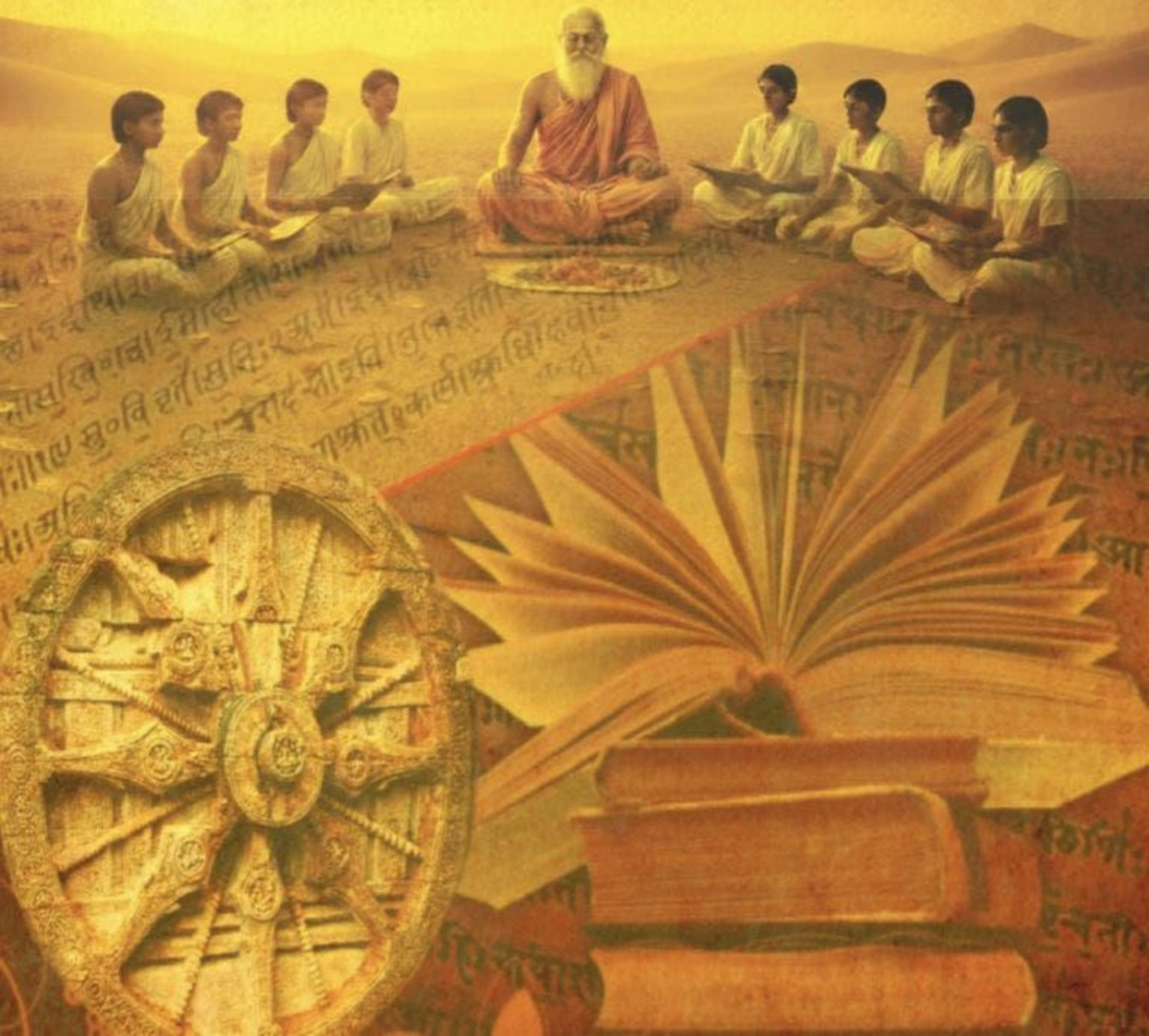
भारतीय ज्ञान प्रणाली
उद्गम और विकास

सम्पादक :
डॉ. दिनेश चन्द्र शर्मा



भारतीय ज्ञान प्रणाली
उद्‌गम और विकास

सम्पादक :

डॉ. दिनेश चन्द्र शर्मा

Ph.D., D.Lit. (Sub.)

निदेशक, कॉम्पिटिशन प्लस ग्रुप ऑफ इन्स्टीट्यूशन,
अजमेर

Books n Books

51/1509, आचमन, वेदविहार,
लोहाखान, अजमेर-305001

प्रकाशक :

पुष्पा शर्मा

Books n Books

51/1509, आचमन, वेदविहार,

लोहाखान, अजमेर-305001

ई-मेल : bnbajmer@gmail.com

मो. 9660111549

संस्करण

प्रथम 2026

***ISBN :* 978-81-993854-1-2**

मूल्य - 400/-

रचना - भारतीय ज्ञान प्रणाली-उद्गम और विकास

सम्पादक - डॉ. दिनेश चन्द्र शर्मा

आवरण - श्रीनिवास प्रिन्टोग्राफिक्स, अजमेर

मुद्रक

श्रीनिवास प्रिन्टोग्राफिक्स, अजमेर

28

Python Basics

Shain Parveen
Assistant Professor (Deptt. of Science and Technology)
Hukumchand Noble Institute of Science and Technology

Introduction

Python is a high-level, interpreted programming language that is widely known for its simplicity, readability, and versatility. Designed by Guido van Rossum and first released in 1991, Python has become one of the most popular programming languages used across various domains such as web development, data science, machine learning, automation, and education. Its easy-to-understand syntax and extensive standard library make it an ideal choice for beginners as well as experienced programmers.

This chapter introduces the fundamental concepts of Python programming, including keywords, identifiers, variables, data types, type conversion, input handling, indentation, and operators. Understanding these basics is essential for writing clear, efficient, and error-free Python programs and forms a strong foundation for learning advanced Python concepts.

Key Characteristics:

* **Readability:** Python's syntax emphasizes clear and concise code, often resembling natural language, which makes it easier to learn and understand. Indentation is used to define code blocks, promoting consistent formatting.

* **Interpreted Language:** Python code is executed line by line by an interpreter, rather than being compiled into machine code beforehand. This allows for rapid prototyping and easier debugging.

* **Dynamic Typing:** Variable types are automatically determined at runtime, eliminating the need for explicit type declarations and offering greater flexibility in coding.

* **Multi-paradigm:** Python supports multiple programming paradigms, including object-oriented, functional, and procedural

programming, allowing developers to choose the approach best suited for their project.

* **Extensive Libraries:** A vast ecosystem of libraries and frameworks exists for Python, catering to diverse applications like data science (NumPy, Pandas), machine learning (TensorFlow, scikit-learn), web development (Django, Flask), and more.

* **Cross-platform:** Python applications can run on various operating systems, including Windows, macOS, and Linux, without requiring significant modifications.

Applications:

Python finds application in numerous fields, including:

* Web Development: Building server-side web applications using frameworks like Django and Flask.

* Data Science and Machine Learning: Data analysis, visualization, and building machine learning models.

* Software Development: Creating desktop applications, scripting, and automation.

* System Scripting: Automating system administration tasks.

* Education: Widely used as a beginner-friendly language for teaching programming concepts.

Keywords in Python

* Predefined and reserved words with special meanings.
* Used to define the syntax and structure of Python code.
* Cannot be used as identifiers, variables, or function names.
* Written in lowercase, except True and False.
* Python 3.11 has 35 keywords.
* The keyword module provides:
  - o **iskeyword()** → checks if a string is a keyword.
  - o **kwlist** → returns the list of all keywords.

Rules for Keywords in Python

* Python keywords cannot be used as identifiers.
* All the keywords in Python should be in lowercase except True and False.

List of Python Keywords

Category	**Keywords**
Value Keywords	True, False, None
Operator Keywords	and, or, not, is, in

Control Flow	if, else, elif, for, while, break, continue,
Keywords	pass, try, except, finally, raise, assert
Function and Class	def, return, lambda, yield, class
Context Management	with, as
Import and Module	import, from
Scope and Namespace	global, nonlocal
Async Programming	async, await

Identifiers in Python

* User-defined names for variables, functions, classes, modules, etc.

* Can include letters, digits, and underscores (_).

* Case-sensitive ? num, Num, and NUM are different identifiers.

* Python provides str.isidentifier() to check if a string is a valid identifier.

Rules for Naming Python Identifiers

* It cannot be a reserved python keyword.

* It should not contain white space.

* It can be a combination of A-Z, a-z, 0-9, or underscore.

* It should start with an alphabet character or an underscore (_).

* It should not contain any special character other than an underscore (_).

Indentation in Python

In Python, indentation is used to define blocks of code. It tells the Python interpreter that a group of statements belongs to a specific block. All statements with the same level of indentation are considered part of the same block. Indentation is achieved using whitespace (spaces or tabs) at the beginning of each line.

For Example:

```
if 10 > 5:
    print("This is true!")
    print("I am tab indentation")
print("I have no indentation")
```

Taking input in Python

```
name = input("Enter your name: ")
```

```
print("Hello,", name, "! Welcome!")
```

Take Multiple Input in Python

```
x, y = input("Enter two values: ").split()
print("Number of boys: ", x)
print("Number of girls: ", y)
 x, y, z = input("Enter three values: ").split()
print("Total number of students: ", x)
print("Number of boys is : ", y)
print("Number of girls is : ", z)
```

Find DataType of Input in Python

```
a = "Hello World"
b = 10
c = 11.22
d = ("Geeks", "for", "Geeks")
e = ["Geeks", "for", "Geeks"]
f = {"Geeks": 1, "for":2, "Geeks":3}
print(type(a))
print(type(b))
print(type(c))
print(type(d))
print(type(e))
print(type(f))
<class 'str'>
<class 'int'>
<class 'float'>
<class 'tuple'>
<class 'list'>
<class 'dict'>
```

Variables

Variables are used to store data that can be referenced and manipulated during program execution. A variable is essentially a name that is assigned to a value.

* Unlike Java and many other languages, Python variables do not require explicit declaration of type.

* The type of the variable is inferred based on the value assigned.

Rules for Naming Variables

To use variables effectively, we must follow Python's naming

rules:

* Variable names can only contain letters, digits and underscores (_).

* A variable name cannot start with a digit.

* Variable names are case-sensitive like myVar and myvar are different.

* Avoid using Python keywords like if, else, for as variable names.

Assigning Values to Variables

Basic Assignment

Variables in Python are assigned values using the = operator.

```
x = 5
y = 3.14
z = "Hi"
```

Dynamic Typing

Python variables are dynamically typed, meaning the same variable can hold different types of values during execution.

```
x = 10
x = "Now a string"
```

Multiple Assignments

Python allows multiple variables to be assigned values in a single line.

Assigning the Same Value

Python allows assigning the same value to multiple variables in a single line, which can be useful for initializing variables with the same value.

```
a = b = c = 100
print(a, b, c)
```

Output:

100 100 100

Assigning Different Values

We can assign different values to multiple variables simultaneously, making the code concise and easier to read.

```
x, y, z = 1, 2.5, "Python"
print(x, y, z)
```

Output:

1 2.5 Python

Type Casting a Variable

Type casting refers to the process of converting the value of one data type into another. Python provides several built-in functions to facilitate casting, including int(), float() and str() among others.

Basic Casting Functions

* int(): Converts compatible values to an integer.
* float(): Transforms values into floating-point numbers.
* str(): Converts any data type into a string.

```
s = "10"
n = int(s)
cnt = 5
f = float(cnt)
age = 25
s2 = str(age)
print(n)
print(f)
print(s2)
```

Output:

```
10
5.0
25
```

Type Conversion in Python

Type conversion means changing the data type of a value. For example, converting an integer (5) to a float (5.0) or a string ("10") to an integer (10). In Python, there are two types of type conversion:

1. Implicit Conversion: Python changes the data type by itself while running the code, to avoid mistakes or data loss.

2. Explicit Conversion: You change the data type on purpose using functions like int(), float() or str().

Implicit Type Conversion

In **implicit conversion**, Python automatically converts one data type into another during expression evaluation. This usually happens when a smaller data type like int is combined with a larger one like float in an operation. **Example:**

```
x = 10          # Integer
y = 10.6        # Float
z = x + y
print("x:", type(x))
```

```
print("y:", type(y))
print("z =", z)
print("z :", type(z))
```

Output

```
x: <class 'int'>
y: <class 'float'>
z = 20.6
z : <class 'float'>
```

Explicit Type Conversion

Explicit conversion (or type casting) is when you manually convert the data type of a value using Python's built-in functions. This is helpful when you want to control how the data is interpreted or manipulated in your code. Some common type casting functions include:

* int() converts a value to an integer
* float() converts a value to a floating point number
* str() converts a value to a string
* bool() converts a value to a Boolean (True/False)

Example:

```
s = "100"  # String
a = int(s)
print(a)
print(type(a))
```

Output

```
100
<class 'int'>
```

Operators

Operators in general are used to perform operations on values and variables.

* Operators: Special symbols like -, + , * , /, etc.
* Operands: Value on which the operator is applied.

Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Operation	Example
+	Addition	5 + 2 = 7
-	Subtraction	4 - 2 = 2
*	Multiplication	2 * 3 = 6

/	Division	4 / 2 = 2
//	Floor Division	10 // 3 = 3
%	Modulo	5 % 2 = 1
**	Power	4 ** 2 = 16

Examples

Here is an example using different arithmetic operators:

Example

```
x = 15
y = 4
print(x + y)
print(x - y)
print(x * y)
print(x / y)                #Division (returns a float)
print(x % y)
print(x ** y)
print(x // y)               #Floor division (returns an integer)
```

Assignment Operators

Operator	Name	Example
=	Assignment Operator	a = 7
+=	Addition Assignment	a += 1 # a = a + 1
-=	Subtraction Assignment	a -= 3 # a = a - 3
*=	Multiplication Assignment	a *= 4 # a = a * 4
/=	Division Assignment	a /= 3 # a = a / 3
%=	Remainder Assignment	a %= 10 # a = a % 10
**=	Exponent Assignment	a **= 10 # a = a ** 10

```
# assign 10 to a
a = 10
# assign 5 to b
b = 5
# assign the sum of a and b to a
a += b      # a = a + b
print(a)
# Output: 15
```

The Walrus Operator

Python 3.8 introduced the := operator, known as the "walrus operator". It assigns values to variables as part of a larger expression:

Example Get your own Python Server

```
numbers = [1, 2, 3, 4, 5]
```

```
count = len(numbers)
if count > 3:
    print(f"List has {count} elements")

if (count := len(numbers)) > 3:
    print(f"List has {count} elements")
```

Comparison Operators

Operator	Meaning	Example
==	Is Equal To	3 == 5 gives us False
!=	Not Equal To	3 != 5 gives us True
>	Greater Than	3 > 5 gives us False
<	Less Than	3 < 5 gives us True
>=	Greater Than or Equal To	3 >= 5 give us False
<=	Less Than or Equal To	3 <= 5 gives us True

```
a = 5
b = 2
# equal to operator
print('a == b =', a == b)
# not equal to operator
print('a != b =', a != b)
# greater than operator
print('a > b =', a > b)
# less than operator
print('a < b =', a < b)
# greater than or equal to operator
print('a >= b =', a >= b)
# less than or equal to operator
print('a <= b =', a <= b)
```

Logical Operators

Operator	Example	Meaning
and	a and b	Logical AND: True only if both the operands are True
or	a or b	Logical OR: True if at least one of the operands is True
not	not a	Logical NOT: True if the operand is False and vice-versa.

```
# logical AND
print(True and True)      # True
print(True and False)     # False
# logical OR
print(True or False)      # True
# logical NOT
print(not True)           # False
```

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.

```
x1 = 5
y1 = 5
x2 = 'Hello'
y2 = 'Hello'
x3 = [1,2,3]
y3 = [1,2,3]
print(x1 is not y1)  # prints False
print(x2 is y2)  # prints True
print(x3 is y3)  # prints False
```

Membership Operators

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

Operator	**Meaning**	**Example**
in	True if value/variable is found in the sequence	5 in x
not in	True if value/variable is not found in the sequence	5 not in x

```
message = 'Hello world'
dict1 = {1:'a', 2:'b'}
# check if 'H' is present in message string
print('H' in message)  # prints True
# check if 'hello' is present in message string
print('hello' not in message)  # prints True
# check if '1' key is present in dict1
print(1 in dict1)  # prints True
# check if 'a' key is present in dict1
```

```
print('a' in dict1)  # prints False
```

Output :

True

True

True

False

Conclusion

In this chapter, we explored the basic building blocks of Python programming. We learned about Python's key features, keywords, identifiers, variables, and the importance of proper indentation. The chapter also covered data types, type casting, input handling, and various operators used to perform operations and comparisons. These concepts are fundamental to writing Python programs and understanding how Python code is executed. Mastering these basics will help learners develop logical thinking and prepare them for more advanced topics such as control structures, functions, object-oriented programming, and real-world Python applications.