Python

By

Dr. Sachin Bhoite

Python is a general-purpose interpreted, interactive, object-oriented, cross-platform and high-level programming language. It was created by Guido van Rossum during 1985- 1990 at the National Research Institute for Mathematics and Computer Science in the Netherland.

Like Perl, Python source code is also available under the GNU General Public License (GPL).

Python is designed to be highly readable. It uses English keywords frequently and it has fewer syntactical constructions than other languages

- Current Version of python

- Python 3.12.4

Features of Python

Python is Interpreted – Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.

Python is Interactive – You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python is Object-Oriented – Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

- **Python is a Beginner's Language** – Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.
- Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, Smalltalk, and Unix shell and other scripting languages.
- Learn Python - (https://www.youtube.com/watch?v=rfscVS0vtbw)

- Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

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 special symbols like !, @, #, $, % etc. as a identifier.

Python is a case sensitive programming language. Thus, **Manpower** and **manpower** are two different identifiers in Python.

Here are naming conventions for Python identifiers –

Class names start with an uppercase letter.

All other identifiers start with a lowercase letter.

Starting an identifier with a single leading underscore indicates that the identifier is private.
Starting an identifier with two leading underscores indicates a strongly private identifier.
If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

Operators

- Arithmetic: + - * / %
- Relational: == != > < >= <=
- Logical: and or not
- Assignment: = += -=
- Membership: in, not in

Logical Operators

1. a = 10
- b = 5
- if a > 5 **and** b < 10:
- print("Both conditions are true")

2. age = 20
- citizen = True
- if age >= 18 and citizen:
- print("Eligible to vote")

- a = 10
- b = 20

- if a > 15 **or** b > 15:
- print("At least one condition is true")

- x = False
- if **not** x:
- print("Condition is reversed")

Reserved Words

The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. There are 33 keywords in Python 3.3. This number can vary slightly in course of time

All the keywords except True, False and None are in lowercase

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Lines and Indentation

Python provides 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. For example –

```
If True:
    print (“True”)
    print(“India”)
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= item1 + \
  - Item2 + \
  - Item3 +
  - Statements contained within the [], {}, or () brackets do not need to use the line continuation character

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 are used to span the string across multiple lines. For example, all the following are legal

word='word'
Sentence="This is sentence"
Paragraph="""" This is
the paragraph. It is made up of multiple lines"""""

Comments in Python

- A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.
- #First Statement
- Print ("Hello World !!") #Second statement
- Multiple Line Comment
- # This is comment
- # This is also comment

Multiple Statements on a Single Line

- The semicolon (;) allows multiple statements on the single line given that neither statement starts a new code block. Here is a sample snip using the semicolon
- X="MIT"; y=20

Multiple Statement Groups as Suites

- A group of individual statements, which make a single code block are called **suites** in Python. Compound or complex statements, such as if, while, def, and class require a header line and a suite.
- Header lines begin the statement (with the keyword) and terminate with a colon (:) and are followed by one or more lines which make up the suite. For example −
- If expression:
- suite
- elif expression :
- suite
- else:
- suite

Python - Variable Types

- Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in 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 need explicit declaration 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.
- The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable. For example −
- Counter =100 # integer variable
- Counter=100.10 #floating variable
- name="Jhon" # String variable

Multiple Assignment

- Python allows you to assign a single value to several variables simultaneously. For example −
- 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. For example
- a,b,c=1,2,"Jhon"
- Here, two integer objects with values 1 and 2 are assigned to variables a and b respectively, and one string object with the value "john" is assigned to the variable c.

Standard Data Types

- The data stored in memory can be of many types.
- Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Python has five standard data types –

- Numbers
- String
- List
- Tuple
- Dictionary

Python Numbers

- Number data types store numeric values. Number objects are created when you assign a value to them. For example −
- Var1=10
- Var2=20
- You can delete a single object or multiple objects by using the del statement. For example −
- del var1
- del var1,var2

Python supports four different numerical types –

- int (signed integers)
- long (long integers, they can also be represented in octal and hexadecimal)
- float (floating point real values)
- complex (complex numbers)

Conditional Statements

- **If Statement**
- if x > 0:
- print("Positive")
- **If–Else**
- if x % 2 == 0:
- print("Even")
- else:
- print("Odd")

Nested if Statement

```
if condition1:
    if condition2:
        statement
```

```
num = 10

if num > 0:
    if num % 2 == 0:
        print("Positive and Even")
```

elif Statement

- if condition1:
- statement
- elif condition2:
- statement
- else:
- statement
- marks = 85

- if marks >= 75:
- print("Grade A")
- elif marks >= 60:
- print("Grade B")
- elif marks >= 40:
- print("Grade C")
- else:
- print("Fail")

Range() function

- **What is range()?**
- range() generates a **sequence of numbers** commonly used with for loops.

range(stop)

- range(5)
- Generates: 0 1 2 3 4
- for i in range(5):
- print(i)

- **range(start, stop)**
- range(1, 6)
- Generates: 1 2 3 4 5
- **for i in range(1, 6):**
- **   print(i)**
- **range(start, stop, step)**
- range(1, 10, 2)
- Generates: 1 3 5 7 9
  for i in range(1, 10, 2):
- **   print(i)**

- **Important Points About range()**
- Start value is **inclusive**
- Stop value is **exclusive**
- Step value is optional (default = 1)
- range() returns a **range object**, not a list

```
for i in range(1, 6):
    print(i * i)

for i in range(5):
    if i == 3:
        break
    print(i)

for i in range(5):
    if i == 2:
        continue
    print(i)
```

- for i in range(5):
- **pass**

- **Nested for**
- for i in range(1, 4):
- for j in range(1, 3):
- print(i, j)

Using else with for Loop

Executed when loop completes normally (without break)

```
for i in range(3):
    print(i)
else:
    print("Loop completed")
```

Reverse loop

```
for i in range(5, 0, -1):
    print(i)
```

- **Sum of numbers**
- total = 0
- for i in range(1, 6):
- 	total += i
- print(total)

While loop

- while condition:
- statements
- **i = 1**
- while i <= 5:
- print(i)
- i += 1
- **i = 2**
- while i <= 10:
- print(i)
- i += 2

```
i = 5
while i > 0:
    print(i)
    i -= 1

while True:
    print("Hello")
```

```
i = 1
while i <= 5:
    if i == 3:
        break
    print(i)
    i += 1
i = 0
while i < 5:
    i += 1
    if i == 3:
        continue
    print(i)
```

- i = 1
- while i <= 5:
- **   pass**

- **i = 1**
- **while i <= 3:**
- **   print(i)**
- **   i += 1**
- **else:**
- **   print("Loop finished")**

Python Strings

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes.
- str = 'Hello World!'
- print str # Prints complete string
- print str[0] # Prints first character of the string
- print str[2:] # Prints string starting from 3rd character
- print str * 2 # Prints string two times
- print str + "TEST" # Prints concatenated string

- **String Using Single and Double Quotes**
- text1 = 'Hello World'
- text2 = "Welcome to Python”

- **Accessing Characters in a String**
- word = "Python"
- print(word[0]) # P
- print(word[-1]) # n

- **String Concatenation**
- first = "Data"
- second = "Science"
- result = first + " " + second
- print(result)

String Length

```
course = "Machine Learning"
print(len(course))
```

Common String Methods

```
msg = "python programming"

print(msg.upper())
print(msg.capitalize())
print(msg.replace("python", "java"))
```

- **String Slicing Syntax**
- string[start : end : step]
- **start** → starting index (inclusive)
- **end** → ending index (exclusive)
- **step** → gap between characters

- Basic slicing
- text = "PythonProgramming"

- print(text[0:6]) # Python
- print(text[6:17]) # Programming

- text = "Artificial Intelligence"
- print(text[0:10])

- **Slicing Without Start or End**
- text = "Artificial Intelligence"
- print(text[:10]) # Artificial
- print(text[11:]) # Intelligence

- **Negative Index Slicing**
- text = "MachineLearning"

- print(text[-8:]) # Learning
- print(text[:-8]) # Machine

- **Step Value in Slicing**
- **text = "Programming"**

- print(text[::2]) # Pormig
- print(text[1::1]) # rogramming

- **Reverse a String Using Slicing (Easy Explanation for PPT)**
- text = "Python"
- print(text[::-1])

- **String Formatting**
- name = "Sachin"
- subject = "Python"

- print(f"My name is {name} and I teach {subject}")

Python Lists

- It is python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.
- The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator. For example −

- list = ['abcd', 786 , 2.23, 'john', 70.2]
- tinylist = [123, 'john']
- print list # Prints complete list
- print list[0] # Prints first element of the list
- print list[1:3] # Prints elements starting from 1st till 2nd
- print list[2:] # Prints elements starting from 3rd element
- print tinylist * 2 # Prints list two times
- print list + tinylist # Prints concatenated lists

Out Put:

- ['abcd', 786, 2.23, 'john', 70.200000000000003]
- abcd
- [786, 2.23]
- [2.23, 'john', 70.200000000000003]
- [123, 'john', 123, 'john']
- ['abcd', 786, 2.23, 'john', 70.200000000000003, 123, 'john']

Python Tuples

- A tuple is another sequence data type that is similar to the list.
- The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as **read-only** lists. For example −

Python Dictionary

- tuple = ('abcd', 786 , 2.23, 'john', 70.2)
- tinytuple = (123, 'john')

- print tuple # Prints complete list
- print tuple[0] # Prints first element of the list
- print tuple[1:3] # Prints elements starting from 2nd till 3rd
- print tuple[2:] # Prints elements starting from 3rd element
- print tinytuple * 2 # Prints list two times
- print tuple + tinytuple # Prints concatenated lists

This produce the following result –

- ('abcd', 786, 2.23, 'john', 70.200000000000003)
- abcd
- (786, 2.23)
- (2.23, 'john', 70.200000000000003)
- (123, 'john', 123, 'john')
- ('abcd', 786, 2.23, 'john', 70.200000000000003, 123, 'john')

Python Dictionary

- Dictionary is an unordered collection of key-value pairs.
- It is generally used when we have a huge amount of data. Dictionaries are optimized for retrieving data. We must know the key to retrieve the value.
- In Python, dictionaries are defined within braces {} with each item being a pair in the form key:value. Key and value can be of any type.

- dict = {}
- dict['one'] = "This is one"
- dict[2]   = "This is two"

- tinydict = {'name': 'john','code':6734, 'dept': 'sales'}

- print dict['one']   # Prints value for 'one' key
- print dict[2]   # Prints value for 2 key
- print tinydict   # Prints complete dictionary
- print tinydict.keys()   # Prints all the keys
- print tinydict.values() # Prints all the values

Types of Operator

- Python language supports the following types of operators.

- Arithmetic Operators
- Assignment Operators
- Comparison (Relational) Operators
- Logical Operators
- Bitwise Operators
- Identity Operators
- Membership Operators

Arithmetic Operators

+	Add two operands or unary plus	x + y +2
-	Subtract right operand from the left or unary minus	x - y -2
*	Multiply two operands	x * y
/	Divide left operand by the right one (always results into float)	x / y
%	Modulus - remainder of the division of left operand by the right	x % y (remainder of x/y)
//	Floor division - division that results into whole number adjusted to the left in the number line	x // y
**	Exponent - left operand raised to the power of right	x**y (x to the power y)

Assignment Operators

=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / ac /= a is equivalent to c = c / a
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a

Comparison (Relational) Operators

Operator	Meaning	Example
>	Greater that - True if left operand is greater than the right	x > y
<	Less that - True if left operand is less than the right	x < y
==	Equal to - True if both operands are equal	x == y
!=	Not equal to - True if operands are not equal	x != y
>=	Greater than or equal to - True if left operand is greater than or equal to the right	x >= y
<=	Less than or equal to - True if left operand is less than or equal to the right	x <= y

Logical Operators

Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

Bitwise Operators

Operator	Meaning	Example
&	Bitwise AND	x& y = 0 (0000 0000)
|	Bitwise OR	x | y = 14 (0000 1110)
~	Bitwise NOT	~x = -11 (1111 0101)
^	Bitwise XOR	x ^ y = 14 (0000 1110)
>>	Bitwise right shift	x>> 2 = 2 (0000 0010)
<<	Bitwise left shift	x<< 2 = 40 (0010 1000)

Python Input, Output and Import

- Python provides numerous built-in functions that are readily available to us at the Python prompt.
- Some of the functions like input() and print() are widely used for standard input and output operations respectively. Let us see the output section first.
- We use the print() function to output data to the standard output device (screen).
- print('This sentence is output to the screen').
- #Output: This sentence is output to the screen

```
a = 5
print('The value of a is', a)
# Output: The value of a is 5
print(1,2,3,4)
# Output: 1 2 3 4
print(1,2,3,4,sep='*')
# Output: 1*2*3*4
print(1,2,3,4,sep='#',end='&')
# Output: 1#2#3#4&
print('I love {0} and {1}'.format('bread','butter'))
# Output: I love bread and butter
```

- print('I love {1} and {0}'.format('bread','butter'))
- # Output: I love butter and bread
- x = 12.3456789
- print('The value of x is %3.2f' %x)
- The value of x is 12.35
- print('The value of x is %3.4f' %x)
- The value of x is 12.3457

Python Input

- >>> num = input('Enter a number: ')
- Enter a number: 10
- >>> num
- '10‘
- >>> int('10')
- 10
- >>> float('10')
- 10.0
- >>> eval('2+3')
- 5

Python Import

- When our program grows bigger, it is a good idea to break it into different modules.
- A module is a file containing Python definitions and statements. Python modules have a filename and end with the extension .py.
- Definitions inside a module can be imported to another module. We use the import keyword to do this.

Python if...else Statement

- Example of if statement
- num = 3
- if num > 0:
- print(num, "is a positive number.")
- print("This is always printed.")
- num = -1
- if num > 0:
- print(num, "is a positive number.")print("This is also always printed.")

- **Example of if...else**
- num = 3
- # Try these two variations as well.
- # num = -5
- # num = 0
- if num >= 0:
- print("Positive or Zero")
- else: print("Negative number")

if...elif...else Statement

```
num = 3.4
# Try these two variations as well:
# num = 0
# num = -4.5
if num > 0:
print("Positive number")
elif num == 0:
 print("Zero")
else:    print("Negative number")
```

for Loop

- # Program to find the sum of all numbers stored in a list
- # List of numbers
- numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]
- # variable to store the sum
- sum = 0
- # iterate over the list
- for val in numbers:
- sum = sum+val
- # Output: The sum is 48
- print("The sum is", sum)


Thank you!