M.Kumarasamy College of Engineering
NAAC Accredited Autonomous Institution
Approved by AICTE & Affiliated to Anna University
ISO 9001:2015 Certified Institution
Thalavapalayam, Karur - 639 113, TAMILNADU.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CGB1121 - PYTHON PROGRAMMING

MODULE 1 - Python Basics, Data Types, Expressions and Operators

INTRODUCTION TO PYTHON PROGRAMMING:

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL).

- Python is *easy to learn* yet powerful and versatile scripting language, which makes it attractive for Application Development.
- Python's syntax and *dynamic typing* with its interpreted nature make it an ideal language for scripting and rapid application development.
- Python supports *multiple programming pattern*, including object-oriented, imperative, and functional or procedural programming styles.
- Python is not intended to work in a particular area, such as web programming. That is why it is known as *multipurpose* programming language because it can be used with web, enterprise, 3D CAD, etc.
- We don't need to use data types to declare variable because it is *dynamically typed* so we can write a=10 to assign an integer value in an integer variable.
- Python makes the development and debugging *fast* because there is no compilation step included in Python development, and edit-test-debug cycle is very fast.

FIGURE 1.2
Guido van Rossum—Creator and BDFL of Python Programming Language. (Image courtesy of Wikipedia.org)

PYTHON FEATURES:

Python provides many useful features which make it popular and valuable from the other programming languages. It supports object-oriented programming, procedural programming approaches and provides dynamic memory allocation. We have listed below a few essential features.

1) Easy to Learn and Use

Python is easy to learn as compared to other programming languages. Its syntax is straightforward and much the same as the English language. There is no use of the semicolon or curly-bracket, the indentation defines the code block. It is the recommended programming language for beginners.

2) Expressive Language

Python can perform complex tasks using a few lines of code. A simple example, the hello world program you simply type **print("Hello World")**. It will take only one line to execute, while Java or C takes multiple lines.

3) Interpreted Language

Python is an interpreted language; it means the Python program is executed one line at a time. The advantage of being interpreted language, it makes debugging easy and portable.

4) Cross-platform Language

Python can run equally on different platforms such as Windows, Linux, UNIX, and Macintosh, etc. So, we can say that Python is a portable language. It enables programmers to develop the software for several competing platforms by writing a program only once.

5) Free and Open Source

Python is freely available for everyone. It is freely available on its official website www.python.org. It has a large community across the world that is dedicatedly working towards make new python modules and functions. Anyone can contribute to the Python community. The open-source means, "Anyone can download its source code without paying any penny."

6) Object-Oriented Language

Python supports object-oriented language and concepts of classes and objects come into existence. It supports inheritance, polymorphism, and encapsulation, etc. The object-oriented procedure helps to programmer to write reusable code and develop applications in less code.

7) Extensible

It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our Python code. It converts the program into byte code, and any platform can use that byte code.

8) Large Standard Library

It provides a vast range of libraries for the various fields such as machine learning, web developer, and also for the scripting. There are various machine learning libraries, such as Tensor flow, Pandas, Numpy, Keras, and Pytorch, etc. Django, flask, pyramids are the popular framework for Python web development.

9) GUI Programming Support

Graphical User Interface is used for the developing Desktop application. PyQT5, Tkinter, Kivy are the libraries which are used for developing the web application.

10) Integrated

It can be easily integrated with languages like C, C++, and JAVA, etc. Python runs code line by line like C,C++ Java. It makes easy to debug the code.

11. Embeddable

The code of the other programming language can use in the Python source code. We can use Python source code in another programming language as well. It can embed other language into our code.

12. Dynamic Memory Allocation

In Python, we don't need to specify the data-type of the variable. When we assign some value to the variable, it automatically allocates the memory to the variable at run time. Suppose we are assigned integer value 15 to **x,** then we don't need to write **int x = 15.** Just write x = 15.

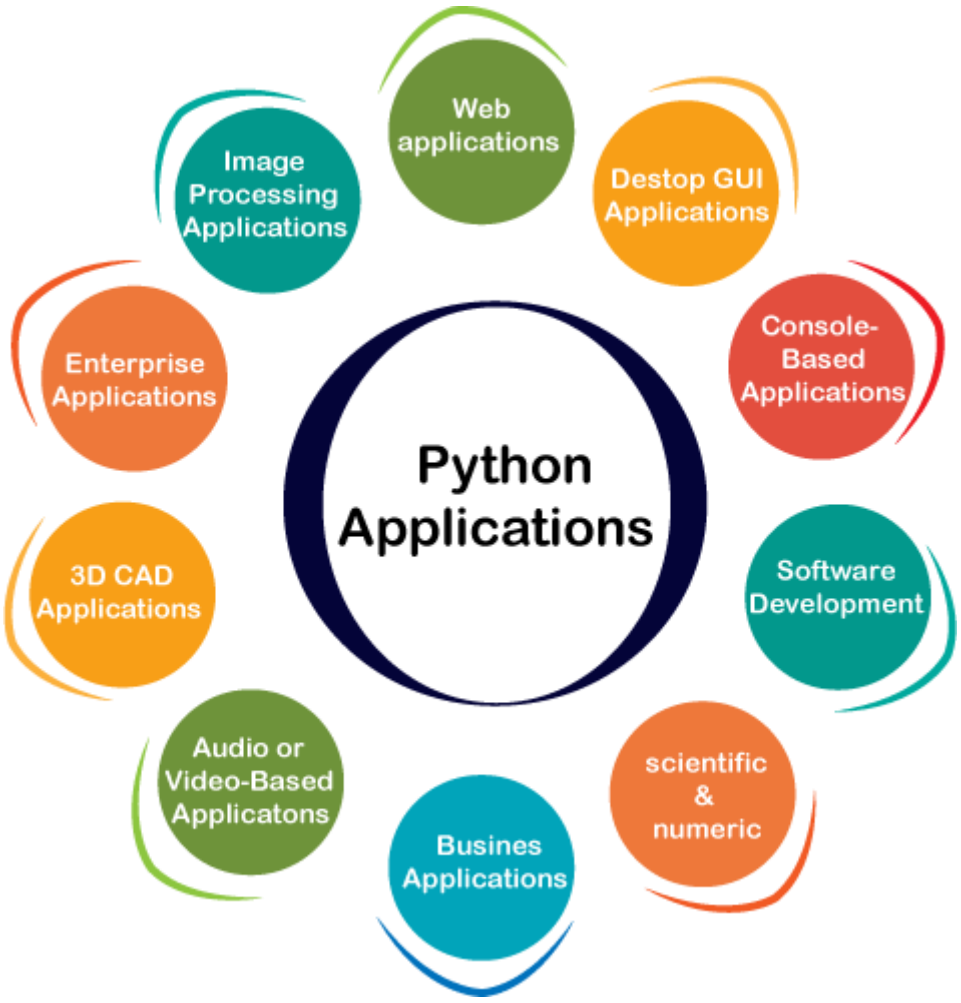
PYTHON HISTORY AND VERSIONS:

- Python laid its foundation in the late 1980s.
- The implementation of Python was started in December 1989 by **Guido Van Rossum** at CWI in Netherland.
- In February 1991, **Guido Van Rossum** published the code (labeled version 0.9.0) to alt.sources.
- In 1994, Python 1.0 was released with new features like lambda, map, filter, and reduce.
- Python 2.0 added new features such as list comprehensions, garbage collection systems.
- On December 3, 2008, Python 3.0 (also called "Py3K") was released. It was designed to rectify the fundamental flaw of the language.
- *ABC programming language* is said to be the predecessor of Python language, which was capable of Exception Handling and interfacing with the Amoeba Operating System.
- The following programming languages influence Python:
- ABC language. Modula-3

PYTHON APPLICATIONS

Python is known for its general-purpose nature that makes it applicable in almost every domain of software development. Python makes its presence in every emerging field. It is the fastest-growing programming language and can develop any application.

Here, we are specifying application areas where Python can be applied.



1) Web Applications

We can use Python to develop web applications. It provides libraries to handle internet protocols such as HTML and XML, JSON, Email processing, request, beautifulSoup, Feedparser, etc. One of Python web-framework named Django is used on **Instagram**. Python provides many useful frameworks, and these are given below:

- Django and Pyramid framework(Use for heavy applications)
- Flask and Bottle (Micro-framework)
- Plone and Django CMS (Advance Content management)

2) Desktop GUI Applications

The GUI stands for the Graphical User Interface, which provides a smooth interaction to any application. Python provides a **Tk GUI library** to develop a user interface. Some popular GUI libraries are given below.

- Tkinter or Tk
- wxWidgetM
- Kivy (used for writing multitouch applications)
- PyQt or Pyside

3) Console-based Application

Console-based applications run from the command-line or shell. These applications are computer program which are used commands to execute. This kind of application was more popular in the old generation of computers. Python can develop this kind of application very effectively. It is famous for having REPL, which means **the Read-Eval-Print Loop** that makes it the most suitable language for the command-line applications.

Python provides many free library or module which helps to build the command-line apps. The necessary **IO** libraries are used to read and write. It helps to parse argument and create console help text out-of-the-box. There are also advance libraries that can develop independent console apps.

4) Software Development

Python is useful for the software development process. It works as a support language and can be used to build control and management, testing, etc.

- **SCons** is used to build control.
- **Buildbot** and **Apache** Gumps are used for automated continuous compilation and testing.
- **Round** or **Trac** for bug tracking and project management.

5) Scientific and Numeric

This is the era of Artificial intelligence where the machine can perform the task the same as the human. Python language is the most suitable language for Artificial intelligence or machine learning. It consists of many scientific and mathematical libraries, which makes easy to solve complex calculations.

Implementing machine learning algorithms require complex mathematical calculation. Python has many libraries for scientific and numeric such as Numpy, Pandas, Scipy, Scikit-learn, etc. If you have some basic knowledge of Python, you need to import libraries on the top of the code. Few popular frameworks of machine libraries are given below.

- SciPy
- Scikit-learn
- NumPy
- Pandas
- Matplotlib

6) Business Applications

Business Applications differ from standard applications. E-commerce and ERP are an example of a business application. This kind of application requires extensively, scalability and readability, and Python provides all these features.

Oddo is an example of the all-in-one Python-based application which offers a range of business applications. Python provides a **Tryton** platform which is used to develop the business application.

7) Audio or Video-based Applications

Python is flexible to perform multiple tasks and can be used to create multimedia applications. Some multimedia applications which are made by using Python are **TimPlayer, cplay,** etc. The few multimedia libraries are given below.

- Gstreamer
- Pyglet
- QT Phonon

8) 3D CAD Applications

The CAD (Computer-aided design) is used to design engineering related architecture. It is used to develop the 3D representation of a part of a system. Python can create a 3D CAD application by using the following functionalities.

- Fandango (Popular)
- CAMVOX
- HeeksCNC
- AnyCAD
- RCAM

9) Enterprise Applications

Python can be used to create applications that can be used within an Enterprise or an Organization. Some real-time applications are OpenERP, Tryton, Picalo, etc.

10) Image Processing Application

Python contains many libraries that are used to work with the image. The image can be manipulated according to our requirements. Some libraries of image processing are given below.

- OpenCV
- Pillow
- SimpleITK

In this topic, we have described all types of applications where Python plays an essential role in the development of these applications. In the next tutorial, we will learn more concepts about Python.

PYTHON INTERPRETER:

Two kinds of programs process high-level languages into low-level languages: **interpreters** and **compilers**. An interpreter reads a high-level program and executes it, meaning that it does what the program says. It processes the program a little at a time, alternately reading lines and performing computations.

A compiler reads the program and translates it completely before the program starts running. In this context, the high-level program is called the **source code**, and the translated program is called the **object code** or the **executable**. Once a program is compiled, you can execute it repeatedly without further translation.

Figure 1.1: An interpreter processes the program a little at a time, alternately reading lines and performing computations.

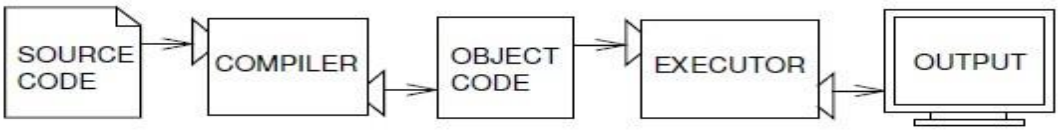


Figure 1.2: A compiler translates source code into object code, which is run by a hardware executor.

INTERACTIVE MODE AND SCRIPT MODE:

Python is considered an interpreted language because Python programs are executed by an interpreter. There are two ways to use the interpreter: **interactive mode** and **script mode**

Interactive Mode Programming

Invoking the interpreter without passing a script file as a parameter brings up the following prompt

```
$ python
Python 2.4.3 (#1, Nov 11 2010, 13:34:43)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Type the following text at the Python prompt and press the Enter −

```
>>> print "Hello, Python!"
```

If you are running new version of Python, then you would need to use print statement with parenthesis as in **print ("Hello, Python!");**. However in Python version 2.4.3, this produces the following result −

Hello, Python!

Script Mode Programming

Invoking the interpreter with a script parameter begins execution of the script and continues until the

script is finished. When the script is finished, the interpreter is no longer active.

Let us write a simple Python program in a script. Python files have extension **.py**. Type the

following source code in a test.py file −

```
print "Hello, Python!"
```

We assume that you have Python interpreter set in PATH variable. Now, try to run this program as follows −

```
$ python test.py
```

This produces the following result −

Hello, Python!

VALUES AND TYPES

- A **value** is one of the basic things a program works with, like a letter or a number. The values we have seen so far are 1, 2, and 'Hello, World!'.
- These values belong to different **types**: 2 is an integer, and 'Hello, World!' is a **string**, so- called because it contains a "string" of letters. You (and the interpreter) can identify strings because they are enclosed in quotation marks.

Python Variables

- Variable is a name that is used to refer to memory location. Python variable is also known as an identifier and used to hold value.
- In Python, we don't need to specify the type of variable because Python is a infer language and smart enough to get variable type.
- Variable names can be a group of both the letters and digits, but they have to begin with a letter or an underscore.
- It is recommended to use lowercase letters for the variable name. Rahul and rahul both are two different variables.

Identifier Naming

Variables are the example of identifiers. An Identifier is used to identify the literals used in the program. The rules to name an identifier are given below.

- The first character of the variable must be an alphabet or underscore (_).
- All the characters except the first character may be an alphabet of lower-case(a-z), upper-case (A-Z), underscore, or digit (0-9).
- Identifier name must not contain any white-space, or special character (!, @, #, %, ^, &, *).
- Identifier name must not be similar to any keyword defined in the language.
- Identifier names are case sensitive; for example, my name, and MyName is not the same.
- Examples of valid identifiers: a123, _n, n_9, etc.
- Examples of invalid identifiers: 1a, n%4, n 9, etc.

Declaring Variable and Assigning Values

Python does not bind us to declare a variable before using it in the application. It allows us to create a variable at the required time.

We don't need to declare explicitly variable in Python. When we assign any value to the variable, that variable is declared automatically.

The equal (=) operator is used to assign value to a variable.

Object References

It is necessary to understand how the Python interpreter works when we declare a variable. The process of treating variables is somewhat different from many other programming languages.

Python is the highly object-oriented programming language; that's why every data item belongs to a specific type of class. Consider the following example.

```
print("John")
```

Output:

```
John
```

The Python object creates an integer object and displays it to the console. In the above print statement, we have created a string object. Let's check the type of it using the Python built-in **type()** function.

```
type("John")
```

Output:

```
<class 'str'>
```

In Python, variables are a symbolic name that is a reference or pointer to an object. The variables are used to denote objects by that name.

Let's understand the following example

```
a = 50
```

In the above image, the variable **a** refers to an integer object.

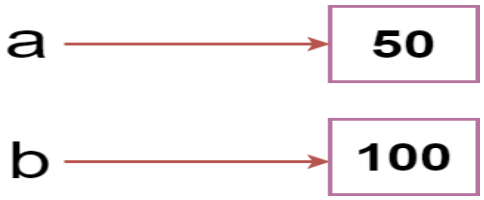
Suppose we assign the integer value 50 to a new variable b.

```
a = 50

b = a
```

The variable b refers to the same object that a points to because Python does not create another object.

Let's assign the new value to b. Now both variables will refer to the different objects.

```
a = 50

b =100
```



Python manages memory efficiently if we assign the same variable to two different values.

Object Identity

In Python, every created object identifies uniquely in Python. Python provides the guaranteed that no two objects will have the same identifier. The built-in **id()** function, is used to identify the object identifier. Consider the following example.

```
1. a = 50
2. b = a
3. print(id(a))
4. print(id(b))
5. # Reassigned variable a
6. a = 500
7. print(id(a))
```

Output:

```
140734982691168
140734982691168
2822056960944
```

We assigned the **b = a, a** and **b** both point to the same object. When we checked by the **id()** function it returned the same number. We reassign **a** to 500; then it referred to the new object identifier.

Variable Names

We have already discussed how to declare the valid variable. Variable names can be any length can have uppercase, lowercase (A to Z, a to z), the digit (0-9), and underscore character(_). Consider the following example of valid variables names.

```
name = "Devansh"
age = 20
marks = 80.50

print(name)
print(age)
print(marks)
```

Output:

```
Devansh
20
80.5
```

Consider the following valid variables name.

```
name = "A"
Name = "B"
naMe = "C"
NAME = "D"
n_a_m_e = "E"
_name = "F"
name_ = "G"
_name_ = "H"
na56me = "I"
print(name,Name,naMe,NAME,n_a_m_e, NAME, n_a_m_e, _name, name_,_name, na56me)
```

Output:

```
A B C D E D E F G F I
```

In the above example, we have declared a few valid variable names such as name, _name_ , etc. But it is not recommended because when we try to read code, it may create confusion. The variable name should be descriptive to make code more readable.

Multiple Assignment

Python allows us to assign a value to multiple variables in a single statement, which is also known as multiple assignments.

We can apply multiple assignments in two ways, either by assigning a single value to multiple variables or assigning multiple values to multiple variables. Consider the following example.

1. Assigning single value to multiple variables

Eg:

```
1. x=y=z=50
2. print(x)
3. print(y)
4. print(z)
```

Output:

```
50
50
50
```

2. Assigning multiple values to multiple variables:

Eg:

```
1. a,b,c=5,10,15
2. print a
3. print b
4. print c
```

Output:

```
5
10
15
```

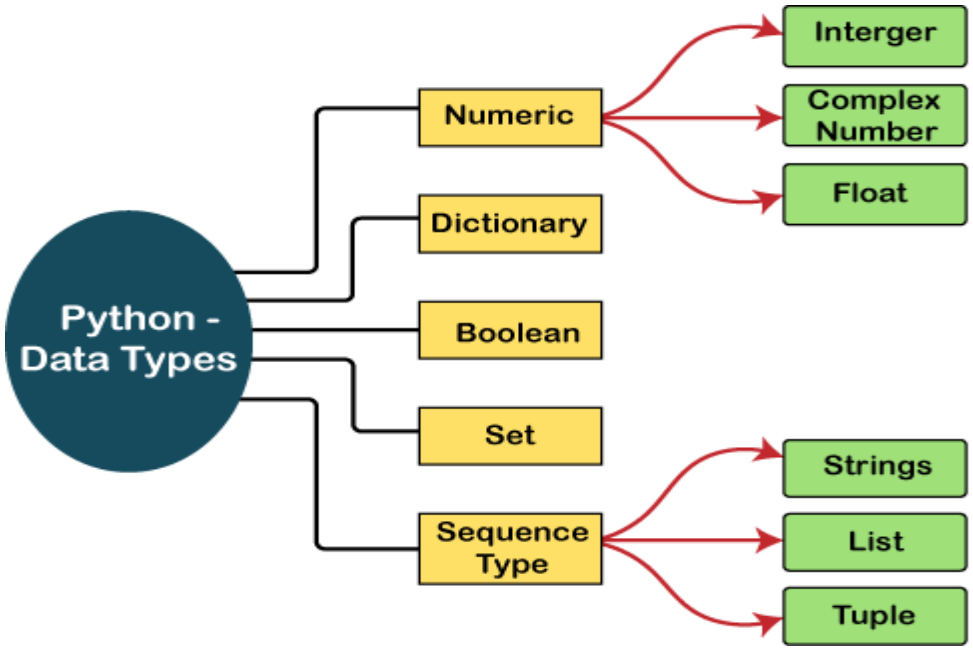
The values will be assigned in the order in which variables appear.

STANDARD DATA TYPES

A variable can hold different types of values. For example, a person's name must be stored as a string whereas its id must be stored as an integer.

Python provides various standard data types that define the storage method on each of them. The data types defined in Python are given below.

1. Numbers
2. Sequence Type
3. Boolean
4. Set
5. Dictionary



Numbers

Number stores numeric values. The integer, float, and complex values belong to a Python Numbers data-type. Python provides the **type()** function to know the data-type of the variable. Similarly, the **isinstance()** function is used to check an object belongs to a particular class.

Python creates Number objects when a number is assigned to a variable. For example;

```
a = 5
print("The type of a", type(a))
b = 40.5
print("The type of b", type(b))
c = 1+3j
print("The type of c", type(c))
print(" c is a complex number", isinstance(1+3j,complex))
```

Output:

```
The type of a <class 'int'>
The type of b <class 'float'>
The type of c <class 'complex'>
c is complex number: True
```

Python supports three types of numeric data.

1. **int -** Integer value can be any length such as integers 10, 2, 29, -20, -150 etc. Python has no restriction on the length of an integer. Its value belongs to **int**
2. **float -** Float is used to store floating-point numbers like 1.9, 9.902, 15.2, etc. It is accurate upto 15 decimal points.
3. **complex -** A complex number contains an ordered pair, i.e., x + iy where x and y denote the real and imaginary parts, respectively. The complex numbers like 2.14j, 2.0 + 2.3j, etc.

Sequence Type

String

- The string can be defined as the sequence of characters represented in the quotation marks. In
- Python, we can use single, double, or triple quotes to define a string.
- String handling in Python is a straightforward task since Python provides built-in functions and operators to perform operations in the string.
- In the case of string handling, the operator + is used to concatenate two strings as the operation *"hello"+" python"* returns *"hello python"*.
- The operator * is known as a repetition operator as the operation "Python" *2 returns 'Python Python'.

Example - 1

```
str = "string using double quotes"
print(str)
s = """A multiline
string"""
print(s)
```

Output:

```
string using double quotes
A multiline
string
```

Example - 2

```
str1 = 'hello javatpoint' #string str1
str2 = ' how are you' #string str2
print (str1[0:2]) #printing first two character using slice operator
print (str1[4]) #printing 4th character of the string
print (str1*2) #printing the string twice
print (str1 + str2) #printing the concatenation of str1 and str2
```

Output:

```
he
o
hello javatpointhello javatpoint
hello javatpoint how are you
```

List

- Python Lists are similar to arrays in C. However, the list can contain data of different types.
- The items stored in the list are separated with a comma (,) and enclosed within square brackets [].
- We can use slice [:] operators to access the data of the list.
- The concatenation operator (+) and repetition operator (*) works with the list in the same way as they were working with the strings.

Consider the following example.

```
list1  = [1, "hi", "Python", 2]

#Checking type of given list
print(type(list1))
#Printing the list1
print (list1)
# List slicing
print (list1[3:])
 # List slicing
print (list1[0:2])
 # List Concatenation using + operator
print (list1 + list1) 16.
# List repetation using * operator
print (list1 * 3)
```

Output:

```
[1, 'hi', 'Python', 2]
[2]
[1, 'hi']
[1, 'hi', 'Python', 2, 1, 'hi', 'Python', 2]
[1, 'hi', 'Python', 2, 1, 'hi', 'Python', 2, 1, 'hi', 'Python', 2]
```

Tuple

- A tuple is similar to the list in many ways.
- Like lists, tuples also contain the collection of the items of different data types.
- The items of the tuple are separated with a comma (,) and enclosed in parentheses ().
- A tuple is a read-only data structure as we can't modify the size and value of the items of a tuple.

```
tup  = ("hi", "Python", 2)
# Checking type of tup
print (type(tup))
   #Printing the tuple
print (tup)
   # Tuple slicing
print (tup[1:])
print (tup[0:1])
 # Tuple concatenation using + operator
print (tup + tup)
# Tuple repatation using * operator
print (tup * 3)
 # Adding value to tup. It will throw an error.
t[2] = "hi"
```

Output:

```
<class 'tuple'>
('hi', 'Python', 2)
('Python', 2)
('hi',)
('hi', 'Python', 2, 'hi', 'Python', 2)
('hi', 'Python', 2, 'hi', 'Python', 2, 'hi', 'Python', 2)

Traceback (most recent call last):
  File "main.py", line 14, in <module>
    t[2] = "hi";
TypeError: 'tuple' object does not support item assignment
```

Dictionary

- Dictionary is an unordered set of a key-value pair of items.
- It is like an associative array or a hash table where each key stores a specific value.
- Key can hold any primitive data type, whereas value is an arbitrary Python object.
- The items in the dictionary are separated with the comma (,) and enclosed in the curly braces {}.

Example:

```
d = {1:'Jimmy', 2:'Alex', 3:'john', 4:'mike'}
# Printing dictionary
print (d)
  # Accesing value using keys
print("1st name is "+d[1])
print("2nd name is "+ d[4]) 9.
print (d.keys())
print (d.values())
```

Output:

```
1st name is Jimmy
2nd name is mike
{1: 'Jimmy', 2: 'Alex', 3: 'john', 4: 'mike'}
dict_keys([1, 2, 3, 4])
dict_values(['Jimmy', 'Alex', 'john', 'mike'])
```

Boolean

- Boolean type provides two built-in values, True and False.
- These values are used to determine the given statement true or false.
- It denotes by the class bool. True can be represented by any non-zero value or 'T' whereas false can be represented by the 0 or 'F'.
- Consider the following example.

```
# Python program to check the boolean type
print(type(True))
print(type(False))
print(false)
```

Output:

```
<class 'bool'>
<class 'bool'>
NameError: name 'false' is not defined
```

Set

- Python Set is the unordered collection of the data type.
- It is iterable, mutable(can modify after creation), and has unique elements.
- In set, the order of the elements is undefined; it may return the changed sequence of the element.
- The set is created by using a built-in function **set(),** or a sequence of elements is passed in the curly braces and separated by the comma.
- It can contain various types of values.
- Consider the following example.

```
# Creating Empty set
set1 = set()
set2 = {'James', 2, 3,'Python'}
#Printing Set value
print(set2)
# Adding element to the set 10.
set2.add(10)
print(set2)
#Removing element from the set
set2.remove(2)
print(set2)
```

Output:

```
{3, 'Python', 'James', 2}
{'Python', 'James', 3, 2, 10}
{'Python', 'James', 3, 10}
```

Data Type Conversion

Sometimes, you may need to perform conversions between the built-in types. To convert between types, you simply use the type name as a function.

There are several built-in functions to perform conversion from one data type to another. These functions return a new object representing the converted value.

S. No.	Function & Description
1	**int(x [,base]) -** Converts x to an integer. base specifies the base if x is a string.
2	**long(x [,base]) -** Converts x to a long integer. base specifies the base if x is a string.
3	**float(x) -** Converts x to a floating-point number.
4	**complex(real [,imag]) -** Creates a complex number.
5	**str(x) -** Converts object x to a string representation.

6	**repr(x) -** Converts object x to an expression string.
7	**eval(str) -** Evaluates a string and returns an object.
8	**tuple(s) -** Converts s to a tuple.
9	**list(s) -** Converts s to a list.
10	**set(s) -** Converts s to a set.
11	**dict(d) -** Creates a dictionary. d must be a sequence of (key,value) tuples.
12	**frozenset(s) -** Converts s to a frozen set.
13	**chr(x) -** Converts an integer to a character.
14	**unichr(x) -** Converts an integer to a Unicode character.
15	**ord(x) -** Converts a single character to its integer value.
16	**hex(x) -** Converts an integer to a hexadecimal string.
17	**oct(x) -** Converts an integer to an octal string.

EXPRESSIONS AND STATEMENTS

An **expression** is a combination of values, variables, and operators. A value all by itself is considered an expression, and so is a variable, so the following are all legal expressions (assuming that the variable x has been assigned a value):

17

x

x + 17

A **statement** is a unit of code that the Python interpreter can execute. We have seen two kinds of statement: print and assignment.Technically an expression is also a statement, but it is probably simpler to think of them as different things. The important difference is that an expression has a value; a statement does not.

Python Keywords

Python Keywords are special reserved words that convey a special meaning to the compiler/interpreter. Each keyword has a special meaning and a specific operation. These keywords can't be used as a variable. Following is the List of Python Keywords.

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

Consider the following explanation of keywords.

1. **True -** It represents the Boolean true, if the given condition is true, then it returns "True". Non-zero values are treated as true.
2. **False -** It represents the Boolean false; if the given condition is false, then it returns "False". Zero value is treated as false
3. **None -** It denotes the null value or void. An empty list or Zero can't be treated as **None**.
4. **and -** It is a logical operator. It is used to check the multiple conditions. It returns true if both conditions are true.

PYTHON OPERATORS

The operator can be defined as a symbol which is responsible for a particular operation between two operands. Operators are the pillars of a program on which the logic is built in a specific programming language. Python provides a variety of operators, which are described as follows.

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

Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations between two operands. It includes + (addition), - (subtraction), *(multiplication), /(divide), %(reminder), //(floor division), and exponent (**) operators.

Operator	Description
+ (Addition)	It is used to add two operands. For example, if a = 20, b = 10 => a+b = 30
- (Subtraction)	It is used to subtract the second operand from the first operand. If the first operand is less than the second operand, the value results negative. For example, if a = 20, b = 10 => a - b = 10
/ (divide)	It returns the quotient after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a/b = 2.0
*** (Multiplication)**	It is used to multiply one operand with the other. For example, if a = 20, b = 10 => a * b = 200
% (reminder)	It returns the reminder after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a%b = 0

**** (Exponent)**	It is an exponent operator represented as it calculates the first operand power to the second operand.
// (Floor division)	It gives the floor value of the quotient produced by dividing the two operands.

Comparison operator

Comparison operators are used to comparing the value of the two operands and returns Boolean true or false accordingly. The comparison operators are described in the following table.

Operator	Description
==	If the value of two operands is equal, then the condition becomes true.
!=	If the value of two operands is not equal, then the condition becomes true.
<=	If the first operand is less than or equal to the second operand, then the condition becomes true.
>=	If the first operand is greater than or equal to the second operand, then the condition becomes true.
>	If the first operand is greater than the second operand, then the condition becomes true.
<	If the first operand is less than the second operand, then the condition becomes true.

Assignment Operators

The assignment operators are used to assign the value of the right expression to the left operand. The assignment operators are described in the following table.

Operator	Description
	It assigns the value of the right expression to the left operand.
+=	It increases the value of the left operand by the value of the right operand and assigns the modified value back to left operand. For example, if a = 10, b = 20 => a+ = b will be equal to a = a+ b and therefore, a = 30.
-=	It decreases the value of the left operand by the value of the right operand and assigns the modified value back to left operand. For example, if a = 20, b = 10 => a- = b will be equal to a = a- b and therefore, a = 10.
*=	It multiplies the value of the left operand by the value of the right operand and assigns the modified value back to then the left operand. For example, if a = 10, b = 20 => a* = b will be equal to a = a* b and therefore, a = 200.
%=	It divides the value of the left operand by the value of the right operand and assigns the reminder back to the left operand. For example, if a = 20, b = 10 => a % = b will be equal to a = a % b and therefore, a = 0.
**=	a**=b will be equal to a=a**b, for example, if a = 4, b =2, a**=b will assign 4**2 = 16 to a.
//=	A//=b will be equal to a = a// b, for example, if a = 4, b = 3, a//=b will assign 4//3 = 1 to a.

Bitwise Operators

The bitwise operators perform bit by bit operation on the values of the two operands.

Operator	Description
& (binary and)	If both the bits at the same place in two operands are 1, then 1 is copied to the result. Otherwise, 0 is copied.
| (binary or)	The resulting bit will be 0 if both the bits are zero; otherwise, the resulting bit will be 1.
^ (binary xor)	The resulting bit will be 1 if both the bits are different; otherwise, the resulting bit will be 0.
~ (negation)	It calculates the negation of each bit of the operand, i.e., if the bit is 0, the resulting bit will be 1 and vice versa.
<< (left shift)	The left operand value is moved left by the number of bits present in the right operand.
>> (right shift)	The left operand is moved right by the number of bits present in the right operand.

For example,

1. **if** a = 7
2. b = 6
3. then, binary (a) = 0111
4. binary (b) = 0011
5. hence, a & b = 0011
6. a | b = 0111
7. a ^ b = 0100
8. ~ a = 1000

Bitwise and (&)	Bitwise or (|)
a = 0011 1100 → (60) b = 0000 1101 → (13) a & b = 0000 1100 → (12)	a = 0011 1100 → (60) b = 0000 1101 → (13) a | b = 0011 1101 → (61)
Bitwise exclusive or (^)	**One's Complement (~)**
a = 0011 1100 → (60) b = 0000 1101 → (13) a ^ b = 0011 0001 → (49)	a = 0011 1100 → (60) ~ a = 1100 0011 → (-61)
Binary left shift (<<)	**Binary right shift (>>)**
a = 0011 1100 → (60) a << 2 = 1111 0000 → (240) left shift of 2 bits	a = 0011 1100 → (60) a >> 2 = 0000 1111 → (15) right shift of 2 bits

FIGURE 2.1
Examples of bitwise logical operators.

FIGURE 2.2 shows how the expression 60 << 2 would be evaluated in a byte.

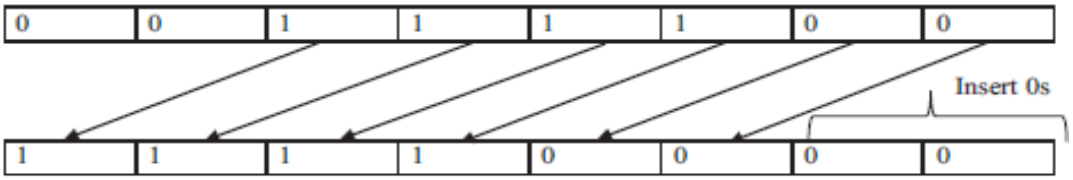


FIGURE 2.2
Example of bitwise left shift of two bits.

Due to this operation,

- Each of the bits in the operand (60) is shifted two places to the left.
- The two bit positions emptied on the right end are filled with 0s.
- The resulting value is 240.

For example,

1. >>> p =60
2. >>> p << 2
   240
3. >>> p = 60
4. >>> p >> 2
   15
5. >>> q = 13
6. >>> p & q
   12
7. >>> p | q
   61

Logical Operators

The logical operators are used primarily in the expression evaluation to make a decision. Python supports the following logical operators.

Operator	Description
and	If both the expression are true, then the condition will be true. If a and b are the two expressions, a → true, b → true => a and b → true.
or	If one of the expressions is true, then the condition will be true. If a and b are the two expressions, a → true, b → false => a or b → true.
not	If an expression **a** is true, then not (a) will be false and vice versa.

Membership Operators

Python membership operators are used to check the membership of value inside a Python data structure. If the value is present in the data structure, then the resulting value is true otherwise it returns false.

Operator	Description
in	It is evaluated to be true if the first operand is found in the second operand (list, tuple, or dictionary).
not in	It is evaluated to be true if the first operand is not found in the second operand (list, tuple, or dictionary).

Identity Operators

The identity operators are used to decide whether an element certain class or type.

Operator	Description
is	It is evaluated to be true if the reference present at both sides point to the same object.
is not	It is evaluated to be true if the reference present at both sides do not point to the same object.

Operator Precedence

The precedence of the operators is essential to find out since it enables us to know which operator should be evaluated first. The precedence table of the operators in Python is given below.

Operator	Description
**	The exponent operator is given priority over all the others used in the
	expression.
~ + -	The negation, unary plus, and minus.
* / % //	The multiplication, divide, modules, reminder, and floor division.
+ -	Binary plus, and minus
>> <<	Left shift. and right shift
&	Binary and.
^ |	Binary xor, and or
<= < > >=	Comparison operators (less than, less than equal to, greater than, greater then equal to).
<> == !=	Equality operators.

= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

Comments

Comments are an important part of any program. A comment is a text that describes what the program or a particular part of the program is trying to do and is ignored by the Python interpreter. Comments are used to help you and other programmers understand, maintain, and debug the program. Python uses two types of comments: single-line comment and multiline comments.

Single Line Comment

In Python, use the hash (#) symbol to start writing a comment. Hash (#) symbol makes all text following it on the same line into a comment. For example,

#This is single line Python comment

Multiline Comments

If the comment extends multiple lines, then one way of commenting those lines is to use hash (#) symbol at the beginning of each line. For example,

#This is
#multiline comments
#in Python

Another way of doing this is to use triple quotes, either ''' or """. These triple quotes are generally used for multiline strings. However, they can be used as a multiline comment as well.

For example,

'''This is
multiline comment
in Python using triple quotes'''

Indentation

In Python, Programs get structured through indentation. Usually, we expect indentation from any program code, but in Python it is a requirement and not a matter of style. This principle makes the code look cleaner and easier to understand and read. Any statements written under another statement with the same indentation is interpreted to belong to the same code block. If there is a next statement with less indentation to the left then it just means the end of the previous code block.In other words, if a code block has to be deeply nested, then the nested statements need to be indented further to the right. In the above diagram, *Block 2* and *Block 3* are nested under *Block 1*. Usually, four whitespaces are used for indentation and are preferred overtabs. Incorrect indentation will result in *IndentationError*.