St. Aloysius Sr. Sec. School, Cantt., Jabalpur

Class 11 Commerce (Informatics Practices)

Unit 1

Week 3 **Dated – 27/04/2020**

Topic : Python Programming -

Welcome children –

Here are more of the language concepts. In the previous week you have studied how to start with Python language and use the simple print() to display your personal messages on the screen.

Things you already know –

1. **Method to install Python**
2. **Python 3 with command line mode**
3. **Python 3 with script mode**
4. **Method to edit scripts in Python**
5. **Use of print()**
6. **print() for displaying plain text**
7. **Method to display text in different lines**
8. **Method to display text in different lines using single print()**
9. **Method to exit Python**
10. **Task done – Creating your own scripts to display your own messages**

MORE >>>

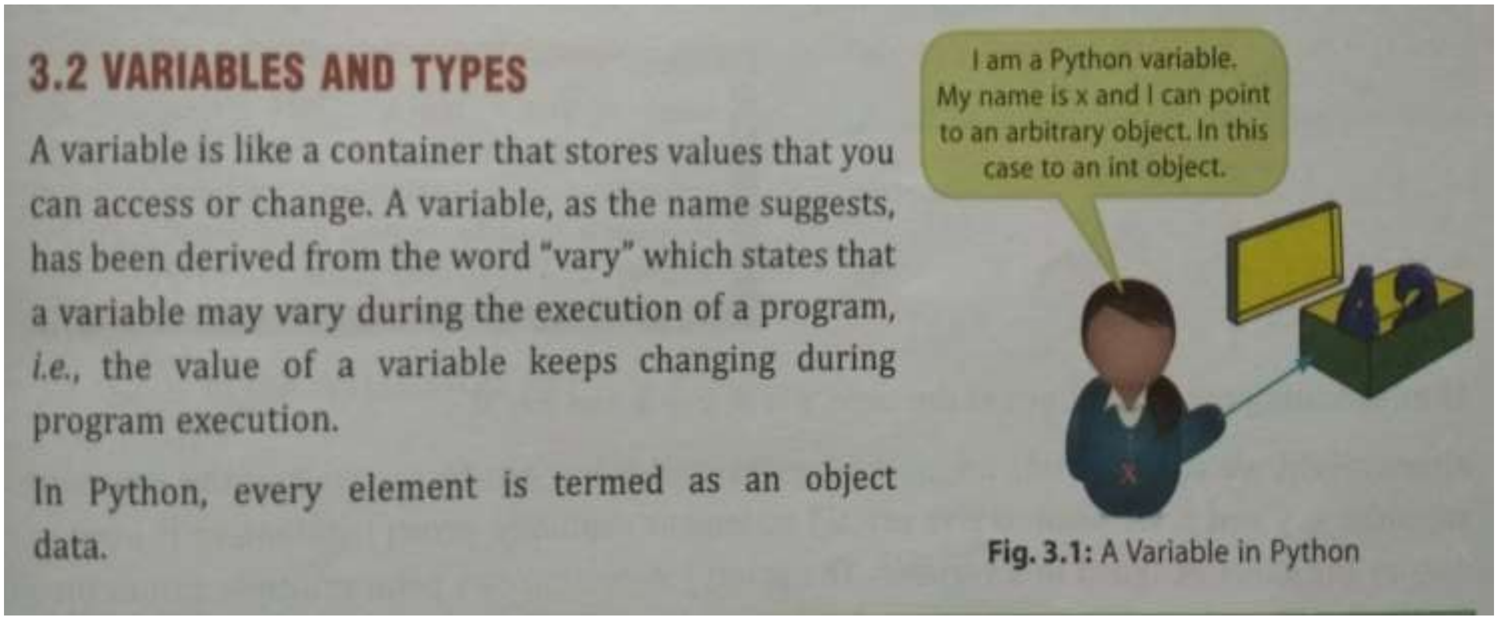
- **<u>Introduction</u> –**

A Python program, sometimes called a script, is a sequence of definitions and commands. These definitions are evaluated and the commands are executed by the Python interpreter which is known as Python Shell.

Python syntax refers to a set of rules that defines how users and the system should write and interpret a Python program. If you want to write and run your program in Python, you must familiarize yourself with its syntax. The syntax to be discussed constitutes the concept of variables and its data types that you should understand first.

Data types in Python programming language are inbuilt and declaration of variables is not required. Also, the memory management is automatically done by Python.

The values that are assigned to the variables, the data type of the variable will be the same as the data type of the value. Hence it is said that Python supports dynamic typing.



3.2 VARIABLES AND TYPES

A variable is like a container that stores values that you can access or change. A variable, as the name suggests, has been derived from the word "vary" which states that a variable may vary during the execution of a program, *i.e.*, the value of a variable keeps changing during program execution.

In Python, every element is termed as an object data.

I am a Python variable. My name is x and I can point to an arbitrary object. In this case to an int object.

Fig. 3.1: A Variable in Python

In English, the word 'object' means a material thing. A material thing can be seen and touched. A material thing has some properties and shows some behaviour. If you look around, you'll find that you are surrounded by innumerable objects such as your television, your mobile, your work desk, your car, the refrigerator in your apartment and so on. Smilarly, in Python also, an object is simply a collection of data (variables) and methods (functions) that act on those data. Hence, a variable is also an object in Python.

Through variable we store the value in computer's memory. Variable is a named unit of storage. A program manipulates its value. Every variable has a type and this type defines the format and behaviour of the variable.

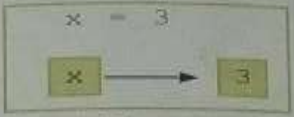
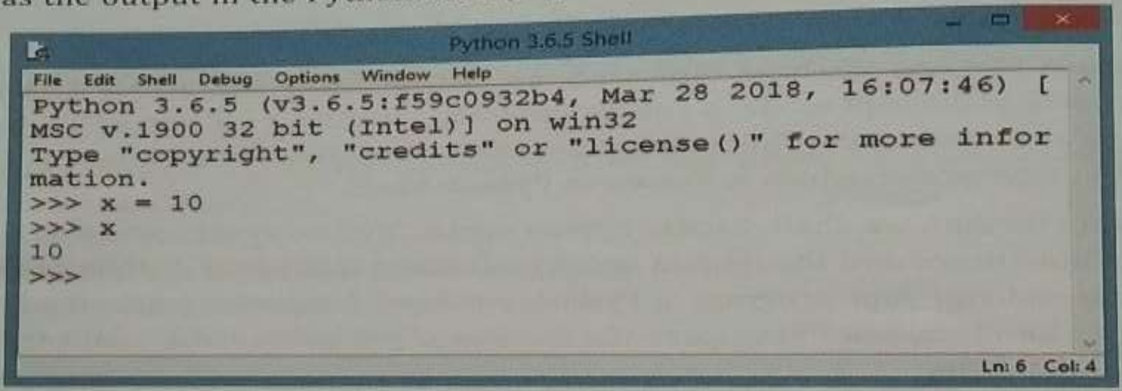


Fig. 3.2: State Diagram of Variable x

For example, if we write x = 3 (as shown in the state diagram, Fig. 3.2), it will automatically assign a value 3 to variable named x, which is an integer.

Similarly, if we type x = 10 (>>> x=10) and then write x (>>>x) and press the Enter key, it will display 10 as the output in the Python shell (Fig. 3.3).



```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [
MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more infor
mation.
>>> x = 10
>>> x
10
>>>
```

Fig. 3.3: Value of Variable x Gets Displayed

The above behaviour exhibits that variable x has been given a value 10, which is stored inside a memory location with the name x. This statement shall store 10 in x but does not get displayed until and unless we type x (>>>x).

On typing x, value 10 shall be displayed as the output.

Let's take another example of working with variables. Consider the following code:

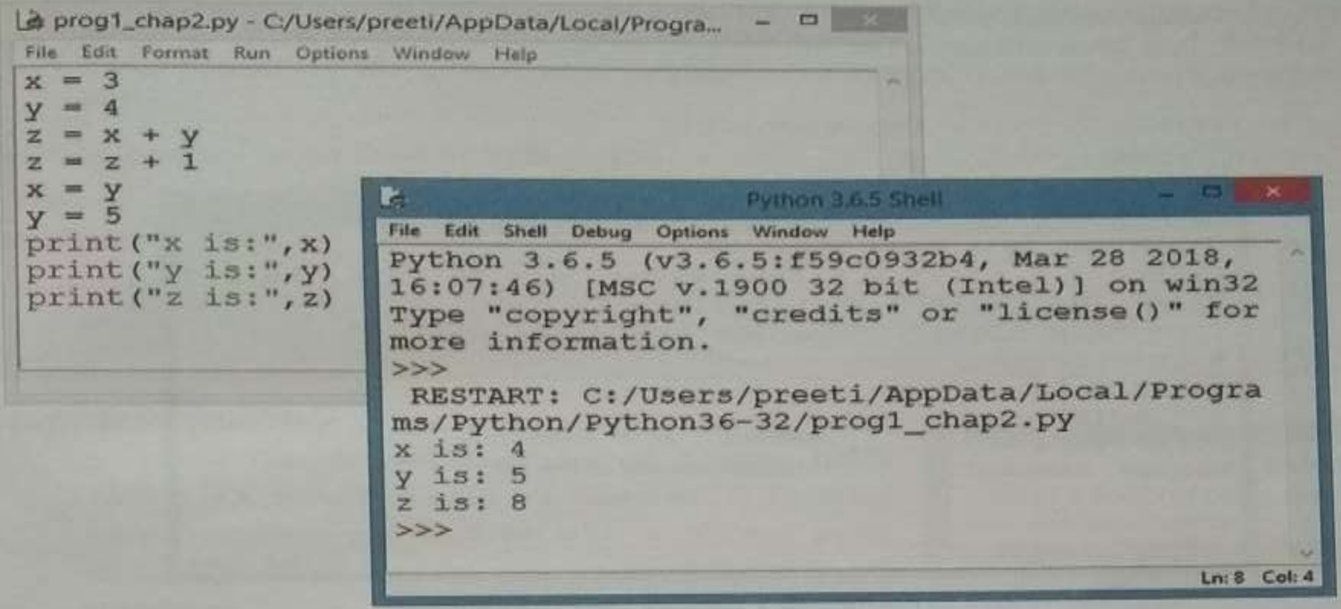
```
>>> x = 3
>>> y = 4
>>> z = x + y
>>> z = z + 1
>>> x = y
>>> y = 5
```

Learning Tip: Python variables are created by assigning value of desired type to them, *e.g.*, to create a numeric variable, assign a numeric value to *variable name*; to create a string variable, assign a string value to *variable name*, and so on.

After executing the above lines of the code, x is 4, y is 5 and z is 8.

Alternatively, we can type this program in script mode also, but for displaying the values of variables x, y and z, we need to give print() statement explicitly. print() statement is used to display the value assigned to a variable. The print() statement can print multiple values on a single line. The comma (,) separates the list of values and variables that are printed.

Now, save the code with a suitable file name with extension .py and then run it (**Run -> Run Module** or press **F5**).


prog1_chap2.py - C:/Users/preeti/AppData/Local/Progra...
File Edit Format Run Options Window Help

```
x = 3
y = 4
z = x + y
z = z + 1
x = y
y = 5
print("x is:",x)
print("y is:",y)
print("z is:",z)
```

Python 3.6.5 Shell
File Edit Shell Debug Options Window Help

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018,
16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for
more information.
>>>
 RESTART: C:/Users/preeti/AppData/Local/Progra
ms/Python/Python36-32/prog1_chap2.py
x is: 4
y is: 5
z is: 8
>>>
```

Fig. 3.4: Running Code inside Script Mode

Let us understand this code line by line:

1. x starts with value 3 and y starts with value 4.
2. In line 3, a variable z is created and equals x + y, which is 7.
3. Then the value of z is changed to equal one more than it currently equals, changing it from 7 to 8.
4. Next, x is changed to the current value of y, which is 4.
5. Finally, y is changed to 5. Note that this does not affect x.
6. So, in the end, x is 4, y is 5 and z is 8.

CTM: print() statement is used to display the value assigned to a variable. The print() statement can print multiple values on a single line. The comma (,) separates the list of values and variables that are printed.

A variable/object has three main components:

A) Identity of the Variable/Object
B) Type of the Variable/Object
C) Value of the Variable/Object

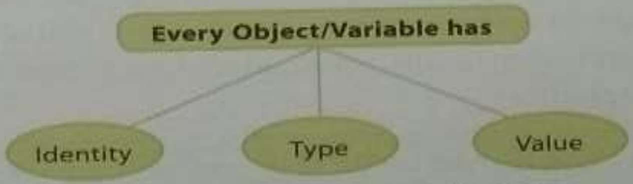


Fig. 3.5: Components of a Variable

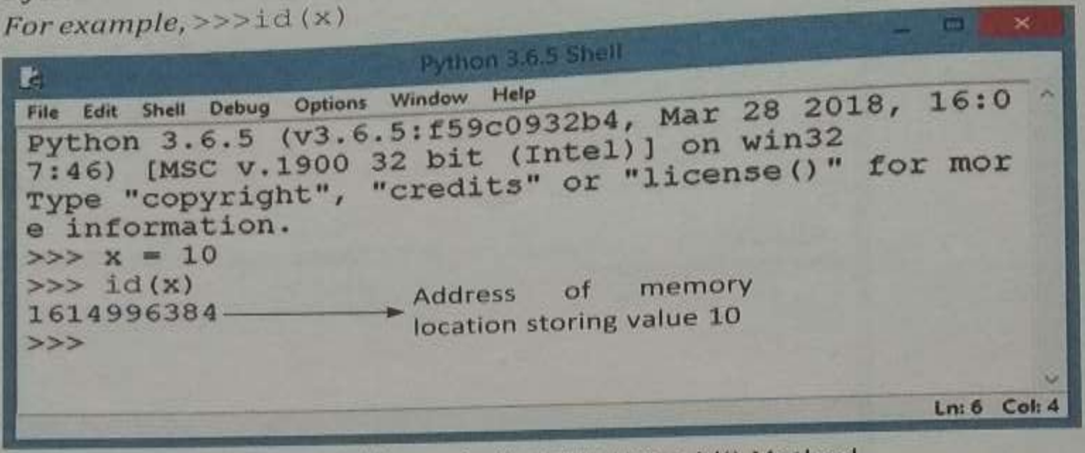
We shall now discuss each of these associated components one by one.

A) **Identity of the Variable/Object**

It refers to the object's (variable in this case) address in the memory which does not change once it has been created. The address of an object can be checked using the method id(object).

Syntax: `>>>id(variable/object)`

For example, `>>>id(x)`



```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:0
7:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for mor
e information.
>>> x = 10
>>> id(x)
1614996384
>>>
```

Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Address of memory location storing value 10
Ln: 6 Col: 4

Fig. 3.6: Address of a Variable Using id() Method

B) **Type of a Variable/Object**

By type we mean the data type of a variable. In Python, each value is an object and has a data type associated with it. Data type of a value refers to the type of value it holds and the allowable operations on those values. A data type in Python can be categorized as follows:

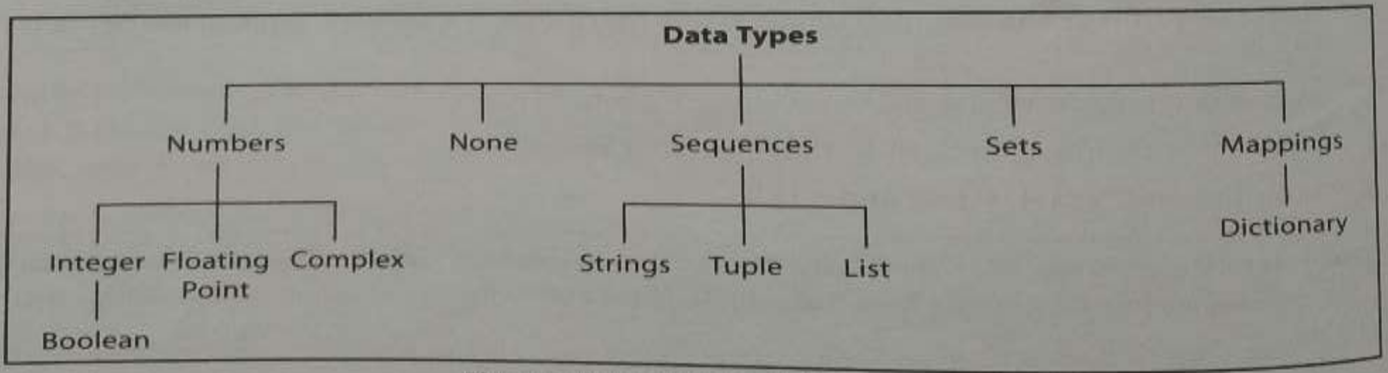


Fig. 3.7: Data Types in Python

The above figure shows the data types available in Python. We are going to discuss only fundamental data types available in Python which are as follows:

1. **Number:** Number data type stores numerical values. One of the most distinguished features of Python is that you don't really have to declare a numeric value to define its type. Python can easily differentiate one data type from another when you write and run your statement. Python supports three types of built-in numeric data types: integer/long, floating point numbers and complex numbers.

(a) **int (integer):** Integer represents whole numbers without any fractional part. They can be positive or negative and have unlimited size in Python. Thus, while working with integers, we need not worry about the size of the integer as a very big-sized integer is automatically handled by Python.

Learning Tip: An integer in Python can be of any length.

Examples of integers recognized by Python are: –6, 793, –255, 4, 0, 23466, –45766782964, etc.

While writing a large integer value, don't use commas to separate digits. Also, integers should not have leading zeros.

CTM: Range of an integer in Python can be from –2147483648 to 2147483647, and long integer has unlimited range subject to available memory.

(b) **float (floating point numbers):** Floating point numbers denote real numbers or floating point values (*i.e.*, numbers with fractional part). Floating point numbers are written with a decimal point that separates the integer from the fractional numbers.

Learning Tip: The fractional part of a floating point number can also be 0 (zero).

Examples of floating point numbers are: 3.14, 6202.3, –43.2, 6.0, 28879.26, etc.

CTM: A value stored as a floating point number in Python can have maximum 53 bits of precision.

(c) **Complex numbers:** Complex numbers in Python are made up of pairs of real and imaginary numbers. They take the form 'x + yJ' or 'x + yj'

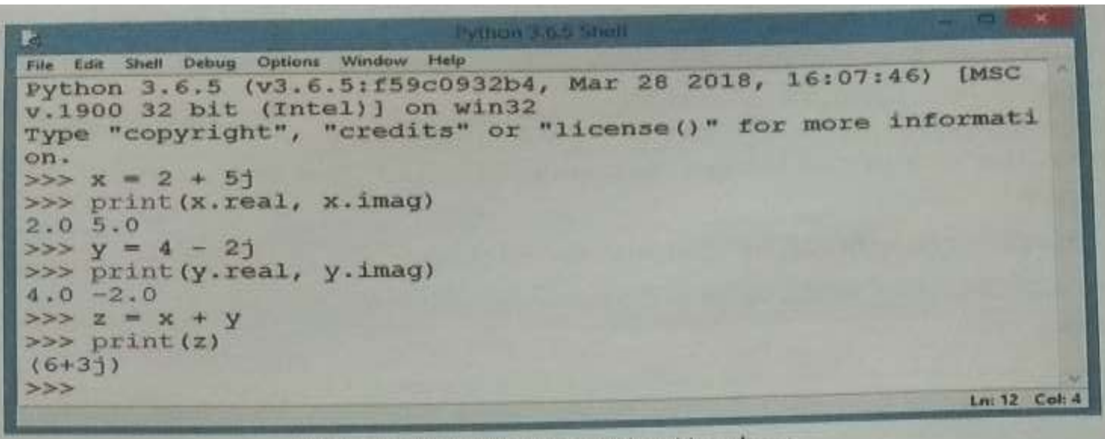
where 'x' is a float and the real part of the complex number. On the other side is yJ where 'y' is a float and J or its lowercase indicates the square root of an imaginary number, –1. This makes 'y' the imaginary part of the complex number.

Here are some examples of complex numbers:

(i)
```
>>>x = 2 + 5j
>>>print(x.real, x.imag)
2.0 5.0
```

(ii)
```
>>>y = 4 - 2j
>>>print(y.real, y.imag)
4.0 - 2.0
```

(iii)
```
>>>z = x + y
>>>print(z)
(6 + 3j)
```



```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC
v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more informati
on.
>>> x = 2 + 5j
>>> print(x.real, x.imag)
2.0 5.0
>>> y = 4 - 2j
>>> print(y.real, y.imag)
4.0 -2.0
>>> z = x + y
>>> print(z)
(6+3j)
>>>
```

Fig. 3.8: Handling Complex Numbers

Complex numbers are not extensively used in Python programming.

2. **str (String):** A string is a sequence of characters that can be a combination of letters, numbers and special symbols, enclosed within quotation marks—single or double or triple (' ' or " " or ''' '''). These quotes are not part of the string. They only define the starting and ending of the string. A string may have any character, sign or even space in between them.

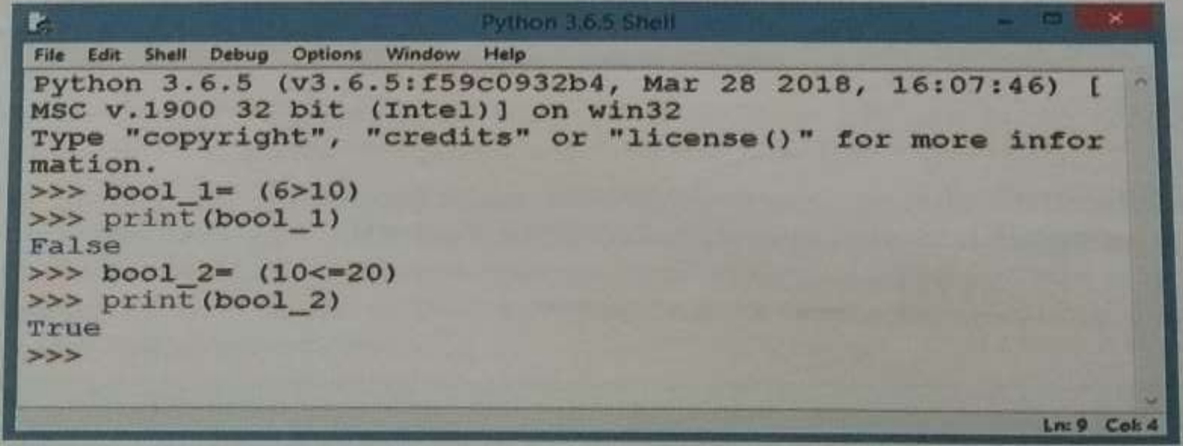
   For example,
   ```
   >>>str = "Hello Python"
   >>>print(str)
   Hello Python
   ```

3. **bool (Boolean):** Boolean data type represents one of the two possible values, True or False. Any expression which can be True or False has the data type bool.

CTM: A Boolean True value is non-zero, non-null and non-empty.

Examples of Boolean expressions are:

➢
```
>>>bool_1 = (6>10)
>>>print(bool_1)
False
```
➢
```
>>>bool_2 = (10<=20)
>>>print(bool_2)
True
```



```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [
MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more infor
mation.
>>> bool_1= (6>10)
>>> print(bool_1)
False
>>> bool_2= (10<=20)
>>> print(bool_2)
True
>>>
```

Fig. 3.9: Handling bool (Boolean) Data Type

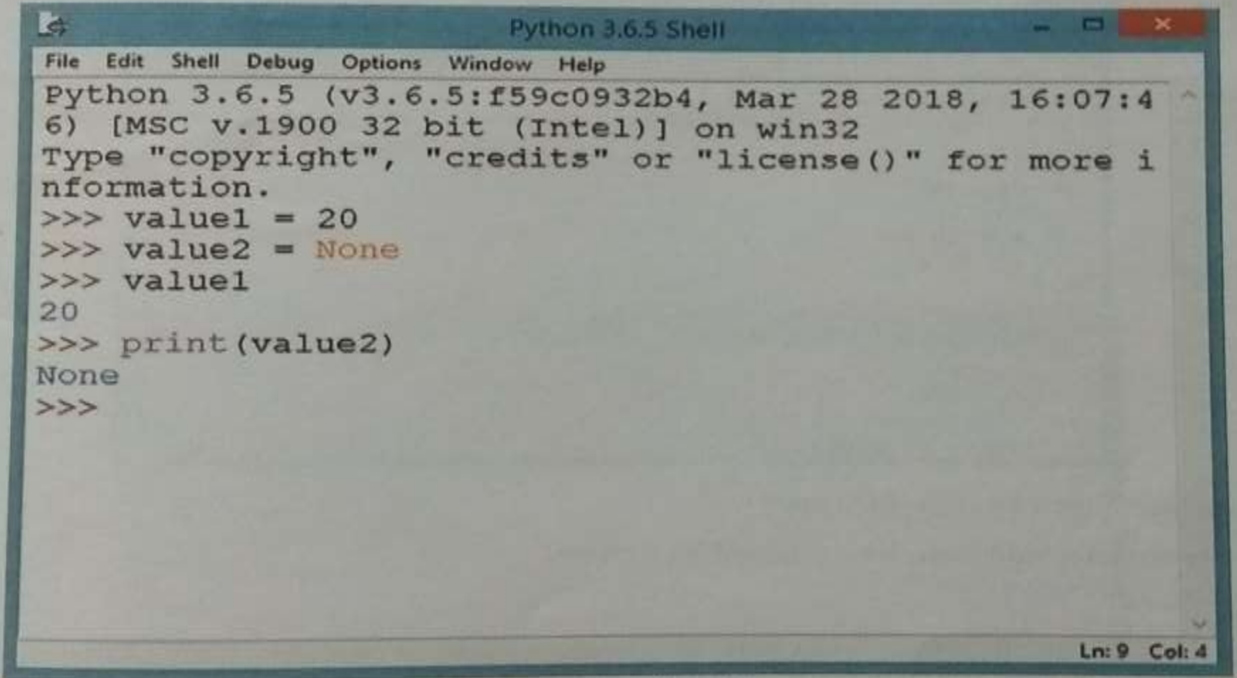
4. **None:** This is a special data type with a single value. It is used to signify the absence of value/condition evaluating to false in a situation. It is represented by **None**. Python doesn't display anything when we give a command to display the value of a variable containing value as None. On the other hand, None will be displayed by printing the value of the variable containing None using print() statement.

   For example,

```
>>>value1 = 20
>>>value2 = None
>>>value1
20
>>>value2
>>>
```

Nothing gets displayed. Alternatively, print() statement can be used to display None value as shown below:

```
>>>print(value2)
None
```



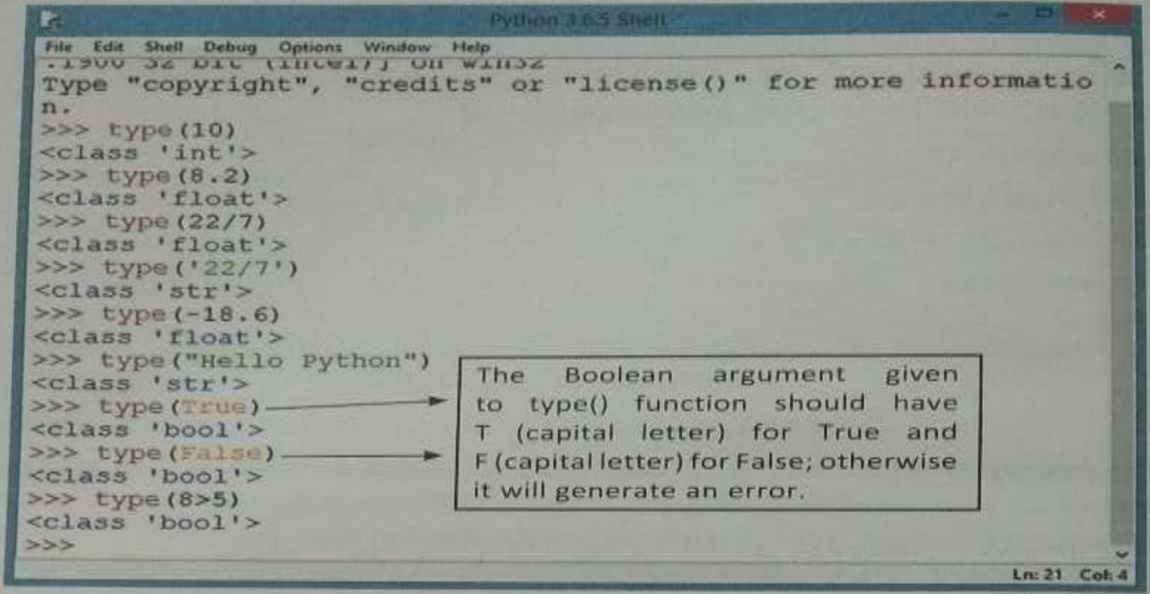
```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:4
6) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more i
nformation.
>>> value1 = 20
>>> value2 = None
>>> value1
20
>>> print(value2)
None
>>>
```

Fig. 3.10: Handling None

3.2.1 type()

If you wish to determine the type of a variable, *i.e.*, what type of value does it hold/point to then type() function can be used.

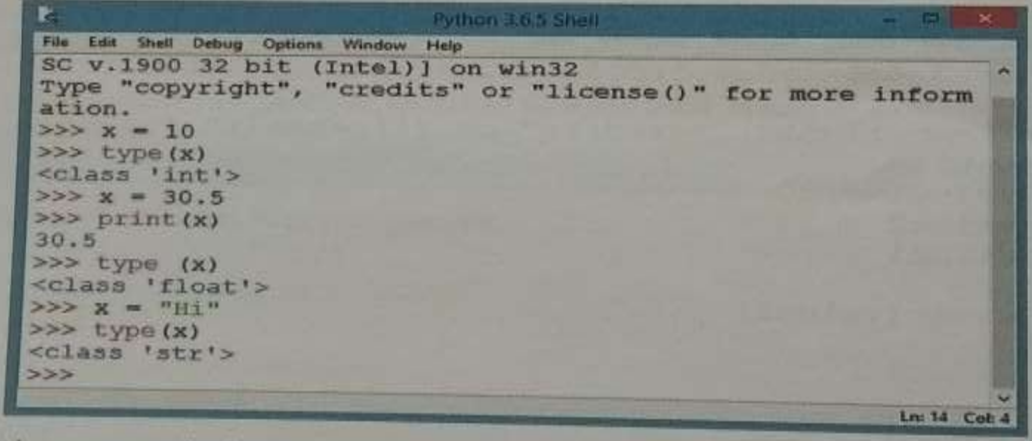
For example,



```
.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more informatio
n.
>>> type(10)
<class 'int'>
>>> type(8.2)
<class 'float'>
>>> type(22/7)
<class 'float'>
>>> type('22/7')
<class 'str'>
>>> type(-18.6)
<class 'float'>
>>> type("Hello Python")
<class 'str'>
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
>>> type(8>5)
<class 'bool'>
>>>
```

The Boolean argument given to type() function should have T (capital letter) for True and F (capital letter) for False; otherwise it will generate an error.

The above code can be alternatively typed as follows:



```
SC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more inform
ation.
>>> x = 10
>>> type(x)
<class 'int'>
>>> x = 30.5
>>> print(x)
30.5
>>> type (x)
<class 'float'>
>>> x = "Hi"
>>> type(x)
<class 'str'>
>>>
```

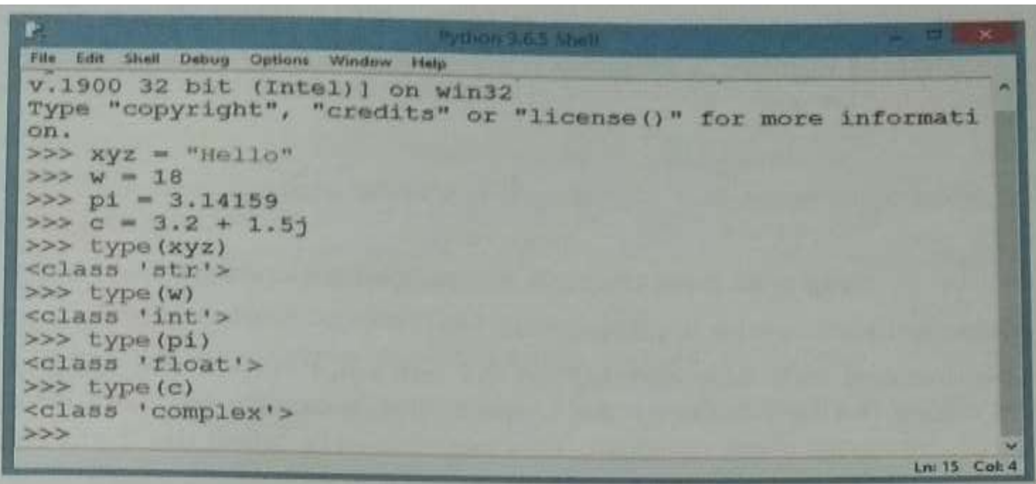
.et us take another example for type().

)ifferent types of values can be assigned as under:

```
xyz = "Hello"
w = 18
pi = 3.14159
c = 3.2 + 1.5j
```

nce values have types, so every variable will also have its type. Now, we will give the following pe() statements for the above values and will observe the output for it (Fig. 3.11).

```
type(xyz)
type(w)
type(pi)
type(c)
```



```
v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more informati
on.
>>> xyz = "Hello"
>>> w = 18
>>> pi = 3.14159
>>> c = 3.2 + 1.5j
>>> type(xyz)
<class 'str'>
>>> type(w)
<class 'int'>
>>> type(pi)
<class 'float'>
>>> type(c)
<class 'complex'>
>>>
```

Fig. 3.11: Use of type() Function

C) **Value of Variable/Object:**

Variables provide a means to name values so that they can be used and manipulated later. To bind value to a variable, we use assignment operator (=). This process is also termed as **building of a variable**.

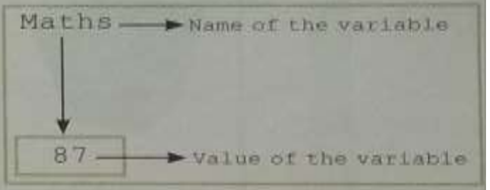
Learning Tip: Operators are the symbols or tokens that trigger some computation/ action when applied to variables and other objects in an expression.

The assignment operator (=)

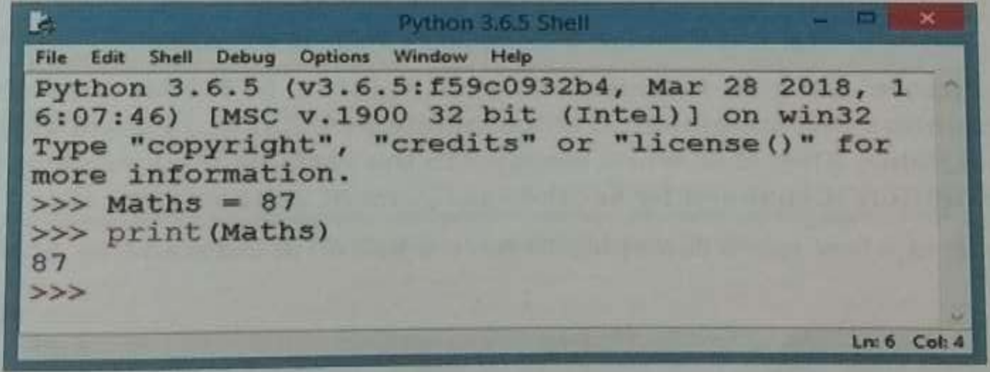
Values are assigned to variables using assignment operator (=).

For example, consider the statement,

```
>>>Maths = 87      ]Maths = 87
>>>print(Maths)   ]print(Maths)
```



The statements shall yield the output as shown below:



```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 1
6:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for
more information.
>>> Maths = 87
>>> print(Maths)
87
>>>
```

In the above example, Python associates the name (variable) Maths with the value 87, *i.e.*, th name (variable) Maths is assigned the value 87 or, in other words, the name (variable) Math refers to value 87. Values are also called objects.

CTM: A variable is not created until some value is assigned to it.

In an assignment statement, the variable that is receiving the value must appear on the le side of the = operator otherwise it will generate an error.

Or we write x = 10, which means assigning value 10 to variable x. Similarly, we write another assignment statement as number_1 assigned with value 100, *i.e.*, number_1 = 100, as shown in the state diagram given below.

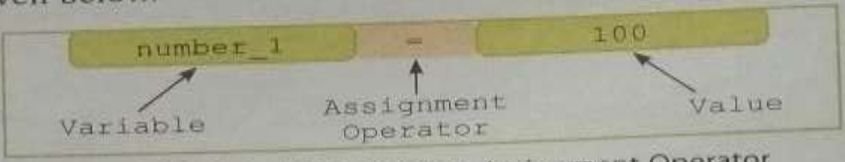


Fig. 3.12: State Diagram of an Assignment Operator

The concept of assignment can be explained as:

There should be one and only one variable on the left-hand side of the assignment operator. This variable is called the Left value or the L-value. There can be any valid expression on the right-hand side of the assignment operator. This expression is called the Right value or R-value. The statement

L-value = R-value

is called the assignment statement. When the interpreter encounters an assignment statement, it evaluates the right-hand side expression (R-value) and assigns the value to the left-hand side variable (L-value).

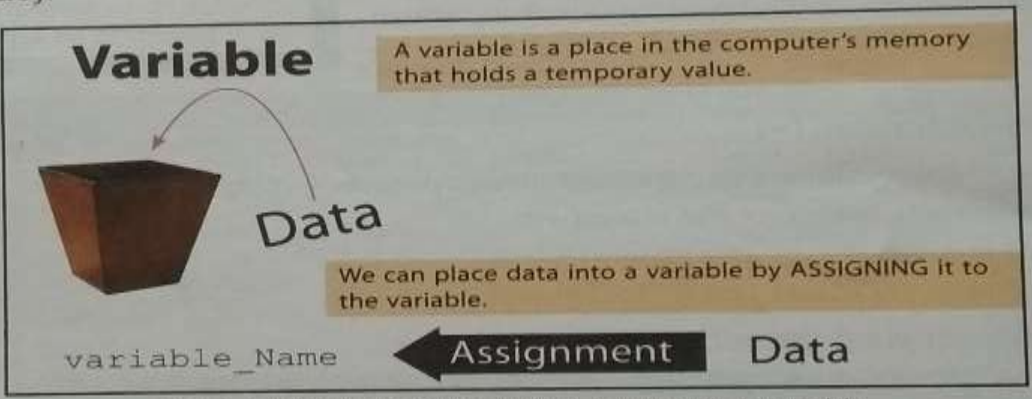
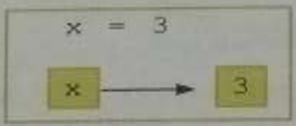

Variable
A variable is a place in the computer's memory that holds a temporary value.
Data
We can place data into a variable by ASSIGNING it to the variable.
variable_Name
Assignment
Data

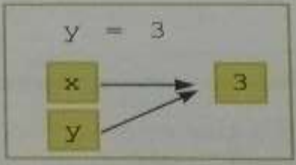
Fig. 3.13: Concept of Assigning a Value to a Variable

For example, suppose we give a statement, final_value = 6+4*2 in a program. When the interpreter encounters this statement, it will evaluate 6+4*2 (*i.e.*, 14), and assign this value to the variable final_value. After this, when we refer to the variable final_value in the program, its value will be 14 until it is changed by another assignment statement.

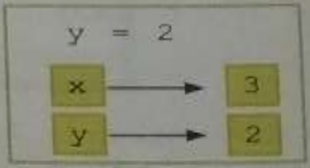
Let us take a look at a few more examples to have a better understanding of this concept.



(i)



(ii)



(iii)

Consider two variables, x and y, and we type the following assignment statements:

```
x = 3    # variable 'x' is assigned 3 as a value, refer to Fig. (i)
y = 3    # variable 'y' is also assigned value 3, so now y shall be pointing to
         x only, refer to Fig. (ii)
```

3.2.2 Multiple Assignments

In Python, we can declare multiple variables in a single statement. It is used to enhance the readability of the program.

Multiple assignments in Python can be done in the following two ways:

(i) **Assigning multiple values to multiple variables:**

var1, var2, var3,. . ., var*n* = value1, value2, value3,. . ., value*n*

This statement will declare variables var1, var2. . . var*n* with values as value1, value2... value*n* respectively.

For example, `msg, day, time = 'Meeting', 'Mon', '9'`

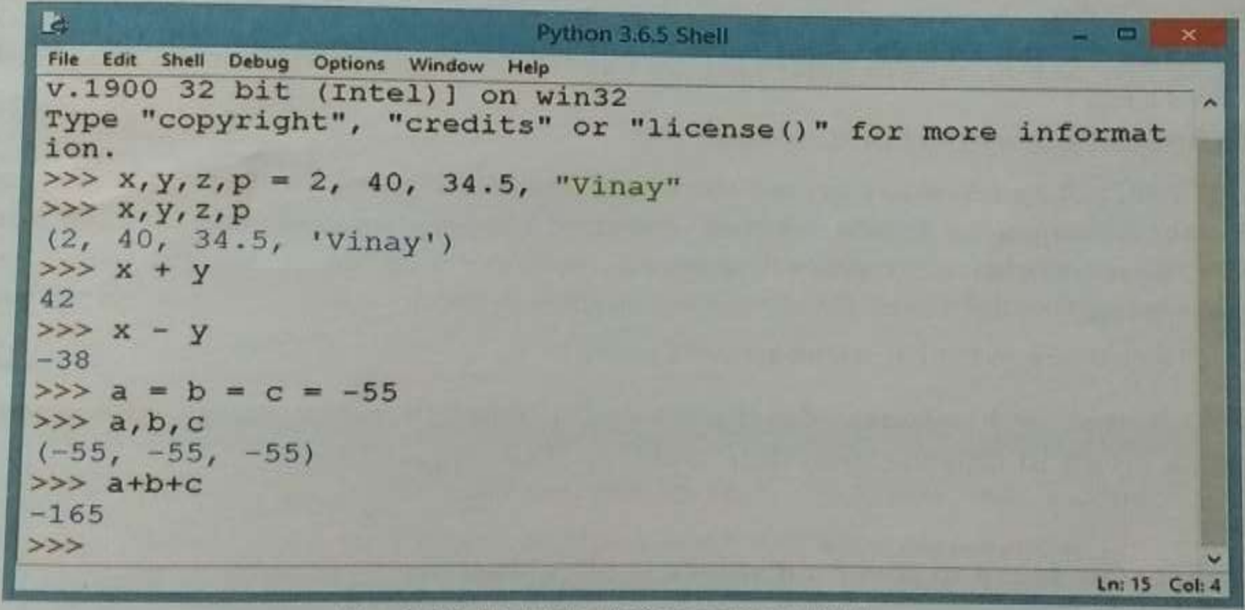
Another statement, `x, y = 2, 3` binds x to 2 and y to 3.

(ii) **Assigning same value to multiple variables:**

var1=var2=var3= . . .=var*n* = value1

This statement will declare variables var1, var2, . . ., var*n*, all with value as value1

For example, `totalMarks = count = 0`



```
v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more informat
ion.
>>> x,y,z,p = 2, 40, 34.5, "Vinay"
>>> x,y,z,p
(2, 40, 34.5, 'Vinay')
>>> x + y
42
>>> x - y
-38
>>> a = b = c = -55
>>> a,b,c
(-55, -55, -55)
>>> a+b+c
-165
>>>
```

Fig. 3.14: Multiple Assignments in Python

3.2.3 Variable Names

There are certain rules in Python which have to be followed to form valid variable names. An variable that violates these rules will be an invalid variable name. The rules to be followed are

(i) A variable name must start with an alphabet (capital or small) or an underscor character (_).

(ii) A variable name cannot contain spaces.

(iii) A variable name can contain any number of letters, digits and underscore (_). No othe characters apart from these are allowed.

(iv) A Python keyword cannot be used as a variable name.

(v) Variable names are case-sensitive, *i.e.*, name and NAME are two different variables.

(vi) You should always choose names for your variables that give an indication of what they are used for. In other words, the name should reflect its functionality. *For example*, a variable that holds the temperature might be named **temperature**, and a variable that holds a car's speed might be named **speed** instead of giving x and b.

Examples of some valid variable names in Python are:

Unit_per_day, dayofweek, stud_name, father_name, TotalMarks, Age, amount, A24, invoice_no.

Following are the examples of some invalid variable names:

Invalid Variable Name	Reason
3dGraph	Name cannot start with a digit
Roll#No	Special symbol (#) is not allowed
First Name	Spaces are not allowed in between
D.O.B	Dots are not allowed
while (while)	Keyword not allowed

Some practical implementation on Variables:

Example 1: To input two numbers and find their sum and product.

Solution:

```
n1 = input("Enter first number:")
n1 = eval(n1)
n2 = input("Enter second number:")
n2 = eval(n2)
sum = n1+n2
print("Sum of the numbers=",sum)
prod = n1*n2
print("Product of the numbers=",prod)
```

Example 2: A student has to travel a distance of 450 km by car at a certain average speed Write Python script to find the total time taken to cover the distance.

Solution:

We know that time taken to cover a distance is calculated as:

Time = Total distance/Average Speed

In this problem, we need a data value (average speed) which is not given and neither can it b calculated. So, this value has to be inputted by the user.

Code is:

```
Distance=450
Speed=eval(input("Enter average speed:")) #to input speed from the use
Time=Distance/Speed                        #Calculate time taken
print("Time taken:",Time,"hr")
```

Example 3: Write a Python code to calculate simple interest by inputting the value of Principal amount and rate from the user for a time period of 5 years.

Solution:

Using the formula:

Simple Interest = Principal × Rate × Time/100

Code:

```
Principal=int(input("Enter the value of Principal:"))
Rate=int(input("Enter the annual rate of interest:"))
Time=5
Simple_Int = Principal*Rate*Time/100
Amount = Principal+Simple_Int
print("Simple Interest =₹",Simple_Int)
print("Amount payable = ₹",Amount)
```

Example 4: Write a program to convert the time inputted in minutes into hours and remaining minutes.

Solution:

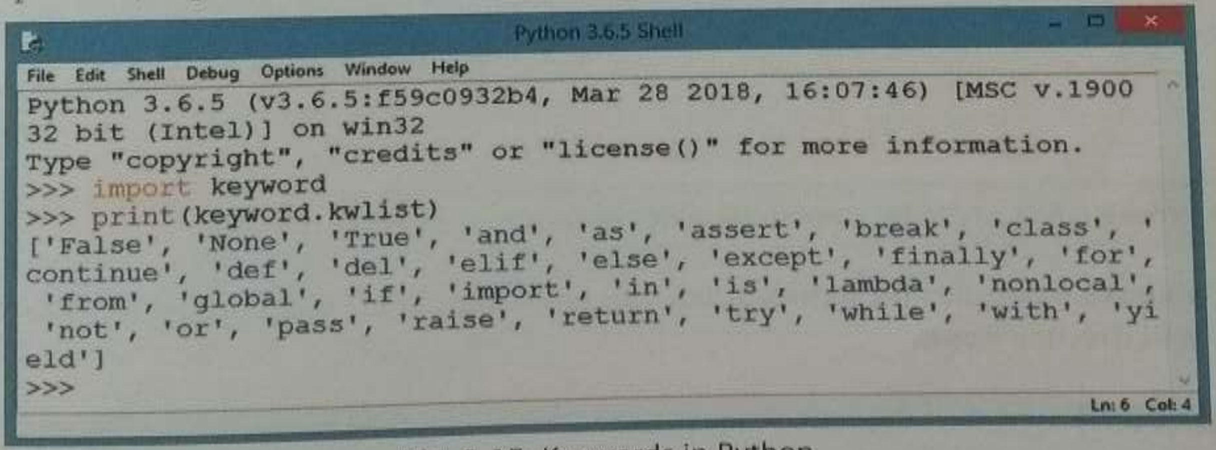
```
time=int(input("Enter the time in minutes:"))
hours= time/60
mins= time%60
print("Hours are:", hours)
print("Minutes are:", mins)
```

Note: In all the above programs, we have used functions for getting the user input (such as int(), eval() and input()), which we shall be discussing in section 3.6.

3.3 KEYWORDS IN PYTHON

Keywords are different from variable names. Keywords are the reserved words used by Python interpreter to recognize the structure of a program. As these words have specific meanings for interpreter, they cannot be used as variable names or for any other purpose. For checking/displaying the list of keywords available in Python, we have to write the following two statements:

```
import keyword
print(keyword.kwlist)
```



```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900
32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import keyword
>>> print(keyword.kwlist)
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', '
continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for',
 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal',
 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yi
eld']
>>>
```

Fig. 3.15: Keywords in Python

3.4 EXPRESSIONS

An expression is a combination of value(s), *i.e.*, constant, variable and operators. It generates a single value, which by itself is an expression. Operands contain the values an operator uses and operators are the special symbols which represent simple calculations like addition, subtraction, multiplication, etc.

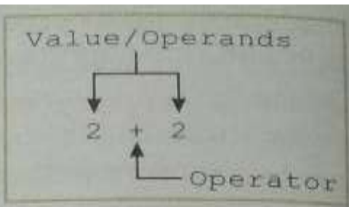


Fig. 3.16: An Expression in Python

Expression	Value
5 + 2 * 4	13
10 / 2 - 3	2
8 + 12 * 2 - 4	28
6 - 3 * 2 + 7 - 1	6

3.4.1 Converting Mathematical Expressions to Equivalent Python Expressions

The mathematical expressions that we use in algebra are not used in the same manner in computers. In maths, you use different operators for mathematical calculations. On the other hand, Python, as well as other programming languages, requires different operators for any mathematical operation.

For example,

Algebraic Expression	Operation Being Performed	Programming Expression
6B	6 times B	6 * B
(3)(12)	3 times 12	3 * 12
4xy	4 times x times y	4 * x * y

When converting some algebraic expressions to programming expressions, you may have to insert parentheses that do not appear in the algebraic expression as shown in the table given below:

Algebraic Expression	Python Statement
$y = 3\left(\frac{x}{2}\right)$	y = 3 * x / 2
$z = 3bc + 4$	z = 3 * b * c + 4
$a = \frac{x+2}{b-1}$	a = (x + 2) / (b - 1)

CTM: An expression is any legal combination of symbols that represents a value.

In all of the above examples for expressions, the most important element used is the 'Operator', which we will discuss now.

3.5 OPERATORS

Operators are special symbols which represent computation. They are applied on operand(s), which can be values or variables. Same operator can behave differently on different data types. Operators when applied on operands form an expression.

Operators are categorized as Arithmetic, Relational, Logical and Assignment. Value and variables, when used with operators, are known as operands.

3.5.1 Mathematical/Arithmetic Operators

A number of operators are available in Python to form and solve arithmetic or algebraic expressions

Symbol	Description	Example 1	Example 2
+	Addition	>>>55+45 100	>>>'Good' + 'Morning' GoodMorning
–	Subtraction	>>>55–45 10	>>>30–80 –50
*	Multiplication	>>> 55*45 2475	>>>'Good'*3 GoodGoodGood
/	Division	>>>17/5 3.4 >>>17/5.0 3.4 >>>17.0/5 3.4	>>>28/3 9.333333333333334
%	Remainder/ Modulo	>>>17%5 2	>>>23%2 1
**	Exponentiation	>>>2**3 8 >>>16**.5 4.0	>>>2**8 256
//	Integer Division	>>>7.0//2 3.0	>>>3//2 1

For example,

```
print("18 + 5 =", 18 + 5)          #Addition
print("18 - 5 =", 18 - 5)          #Subtraction
print("18 * 5 =", 18 * 5)          #Multiplication
print("27 / 5 =", 27 / 5)          #Division
print("27 // 5 =", 27 // 5)        #Integer Division
print("27 % 5 =", 27 % 5)          #Modulus
print("2 ** 3 =", 2 ** 3)          #Exponentiation
print("-2 ** 3 =", -2 ** 3)        #Exponentiation
18 + 5 = 23
18 - 5 = 13
18 * 5 = 90
27 / 5 = 5.4
27 // 5 = 5
27 % 5 = 2
2 ** 3 = 8
-2 ** 3 = -8
```

Concatenating Strings

Strings can be added together with plus (+) operator. To concatenate the string "Hello Python" give the statement using concatenation operator (+):

Output:

```
>>> "Hello" + "Python"
'HelloPython'
```

Syntax:

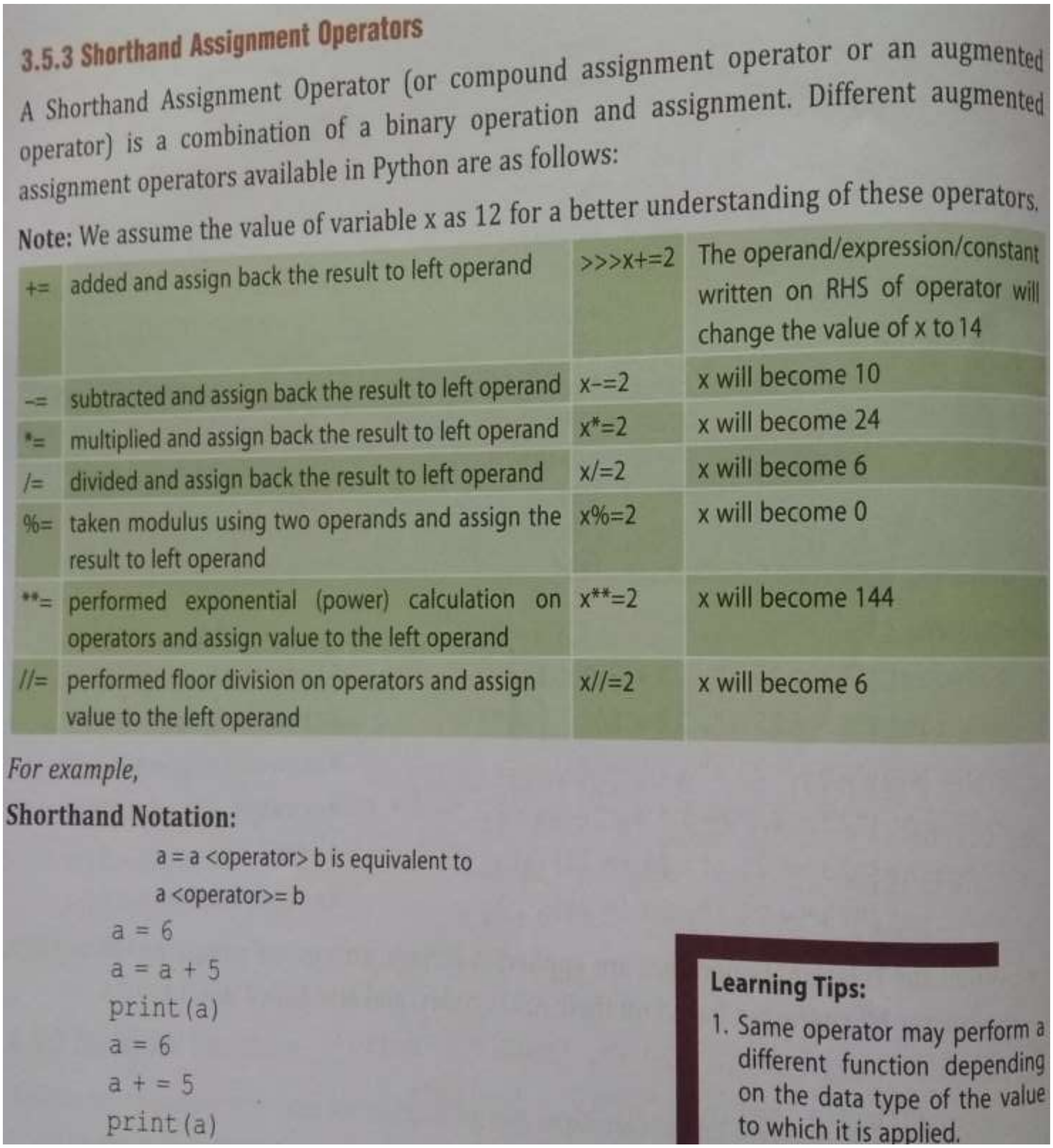
<expression1> <comparison operator> <expression2>

CTM: Relational operators are also known as Comparison operators and yield values, True or False, a the output.

Symbol	Description	Example 1	Example 2
<	less than	>>>7<10 True >>>7<5 False >>>7<10<15 True >>>7<10 and10<15 True	>>>'Hello'<'Goodbye' False >>>'Goodbye'<'Hello' True
>	greater than	>>>7>5 True >>>10>10 False	>>>'Hello'>'Goodbye' True >>>'Goodbye'>'Hello' False
<=	less than equal to	>>>2<=5 True >>>7<=4 False	>>>'Hello'<='Goodbye' False >>>'Goodbye'<='Hello' True
>=	greater than equal to	>>>10>=10 True >>>10>=12 False	>>>'Hello'>='Goodbye' True >>>'Goodbye'>='Hello' False
!=, <>	not equal to	>>>10!=11 True >>>10!=10 False	>>>'Hello'!='HELLO' True >>>'Hello'!='Hello' False
==	equal to	>>>10==10 True >>>10==11 False	>>>'Hello'=='Hello' True >>>'Hello'=='Good Bye' False

For example,

```
>>>print("23 < 25 :", 23 < 25)                    #less than
>>>print("23 > 25 :", 23 > 25)                    #greater than
>>>print("23 <= 23 :", 23 <= 23)                  #less than or equal to
>>>print("23 - 2.5 >= 5 * 4 :", 23 - 2.5 >= 5 * 4) #greater than or equal
>>>print("23 == 25 :", 23 == 25)                  #equal to
                    ", 23 != 25)                  #not equal to
```



3.5.3 Shorthand Assignment Operators

A Shorthand Assignment Operator (or compound assignment operator or an augmented operator) is a combination of a binary operation and assignment. Different augmented assignment operators available in Python are as follows:

Note: We assume the value of variable x as 12 for a better understanding of these operators.

+=	added and assign back the result to left operand	>>>x+=2	The operand/expression/constant written on RHS of operator will change the value of x to 14
-=	subtracted and assign back the result to left operand	x-=2	x will become 10
*=	multiplied and assign back the result to left operand	x*=2	x will become 24
/=	divided and assign back the result to left operand	x/=2	x will become 6
%=	taken modulus using two operands and assign the result to left operand	x%=2	x will become 0
**=	performed exponential (power) calculation on operators and assign value to the left operand	x**=2	x will become 144
//=	performed floor division on operators and assign value to the left operand	x//=2	x will become 6

For example,

Shorthand Notation:

a = a <operator> b is equivalent to

a <operator>= b

```
a = 6
a = a + 5
print(a)
a = 6
a + = 5
print(a)
```

Learning Tips:

1. Same operator may perform a different function depending on the data type of the value to which it is applied.

*************End ******************