Chapter 2

Fundamental concepts in Python

(Literals, Variables and Data types)

2.1 What type of character set is used by Python?

Character set defines the valid characters that can be used in a source program .Python uses the character set as the building block to form the basic program elements such as variables, identifiers ,constants, expressions etc.

Python uses the *traditional ASCII character set.* The latest version also recognizes the *Unicode character set.* The *ASCII character set* is a subset of the *Unicode character set.* The main limitation of ASCII character is its inability to represent more than 256(=2^8). Unicode character set, a set that use 2 bytes (16 bits) per character. Due to its 16 bit per character representation the Unicode character set can represent 216 different characters

2.2 What do you mean by tokens? Different types of tokens in Python.

Token is the smallest unit inside the given program. Tokens can be defined as a punctuator mark, reserved words and each individual word in a statement.

There are following tokens in Python:

- *Identifiers* like (emp_name, emp_id)
- *Keywords* (reserved words like if, elif etc.)
- *Special characters* like (#, " ","""")
- *Literals* like (12, 12.5,'A')
- *Operators* like (/, //, + etc.)

2.3 What do mean by Identifiers?

Identifier is the name given to entities like class, functions, variables etc. in Python. It helps differentiating one entity from another.

Rules for writing identifiers:

- Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_). **Names like my Class, var_1 and print_this_to_screen, all are valid example.**
- An identifier must start with a letter or an underscore. It cannot start with a digit. **variable and _variable are valid identifiers but 1variable is invalid.**
- An identifier cannot be a keyword .Keywords, also called reserved words, have special meanings in Python. For example, import is a keyword, which tells the Python interpreter to import a module to the program.
- An identifier can be of any length.

2.4 What do you mean by Keywords? Write various types of Keywords used in python.

Keywords are the reserved words in Python. Keywords cannot be used as variable name, function name or any other identifier. They are used to define the syntax and structure of the Python language. In Python, keywords are case sensitive.

There are 33 keywords in Python. All the keywords except True, False and None are in lowercase and they must be written as it is. The list of all the keywords are given below.

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

2.5 What do you mean by literals? Explain different types of literals in Python.

A literal is a sequence of one of more characters that stands for itself, such as the literal 12.

Different types of literals:

- ***Numeric Literals***

  A numeric literal is a literal containing only the digits 0–9, an optional sign character (+ or -), and a possible decimal point. If a numeric literal contains a decimal point, then it denotes a floating-point value, or "float" (e.g., 10.24); otherwise, it denotes an integer value (e.g., 10). Commas are never used in numeric literals.

  Numeric literals can belong to following different numerical types:

  a) *Integer (signed)* -Numbers (can be both positive and negative) with no fractional part.e.g. 100

  b) *Float pointing* - Real numbers with both integer and fractional part e.g. -26.2

  c) *Complex* - In the form of a+bj where a forms the real part and b forms the imaginary part of complex number. e.g. 3+4j

- ***String Literals***

  Numerical values are not the only literal values in programming. String literals, or "strings," represent a sequence of characters,

  'Hello' 'Virat, Rohit' "Hello, how are you"

  In Python, string literals may be delimited (surrounded) by a matching pair of either single (') or double (") quotes. Strings must be contained all on one line (except when delimited by triple quotes).

  Types of Strings:

  There are two types of Strings supported in Python.

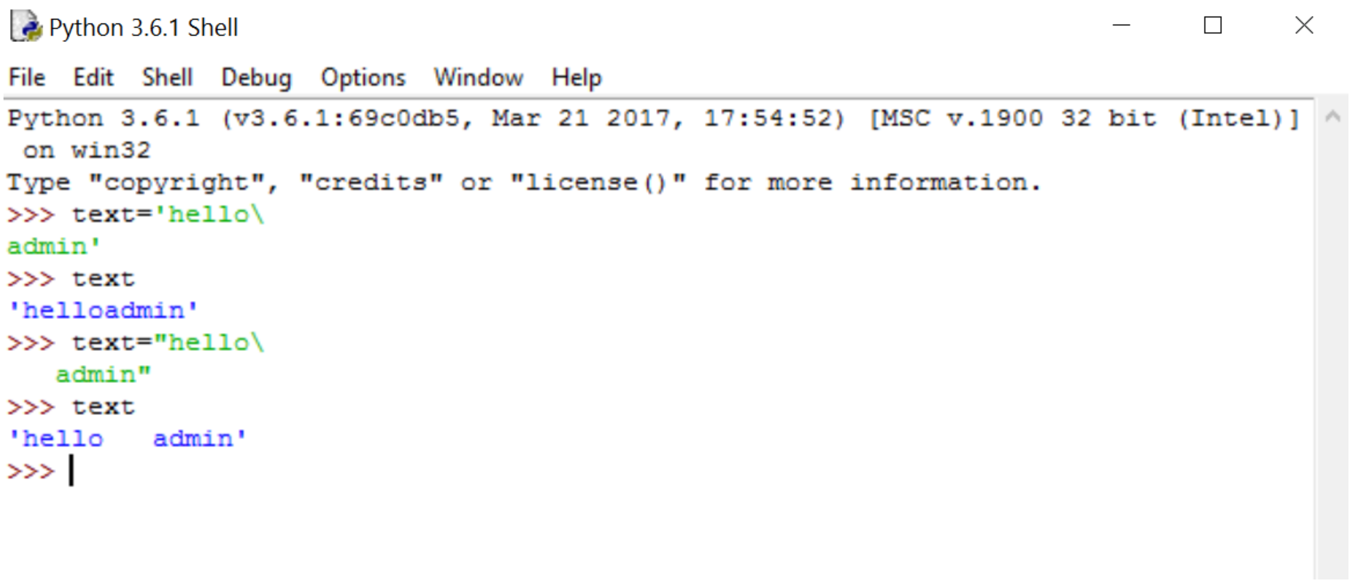
  a) ***Single line String***- Strings that are terminated within a single line are known as Single line Strings.

  Eg: text='hello'
  text1="hello"

b) ***Multi line String-*** A piece of text that is spread along multiple lines is known as multiple line String.

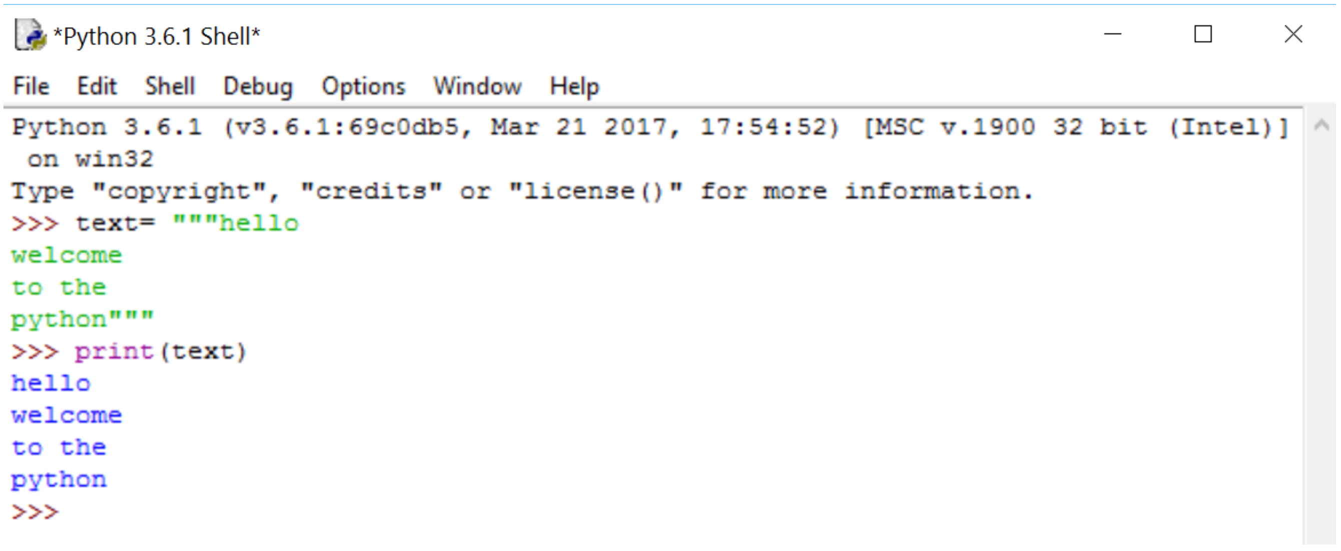
There are two ways to create Multiline Strings:

i. ***Adding black slash at the end of each line.***



```
Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 17:54:52) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> text='hello\
admin'
>>> text
'helloadmin'
>>> text="hello\
   admin"
>>> text
'hello   admin'
>>> 
```

ii. ***Using triple quotation marks:-***



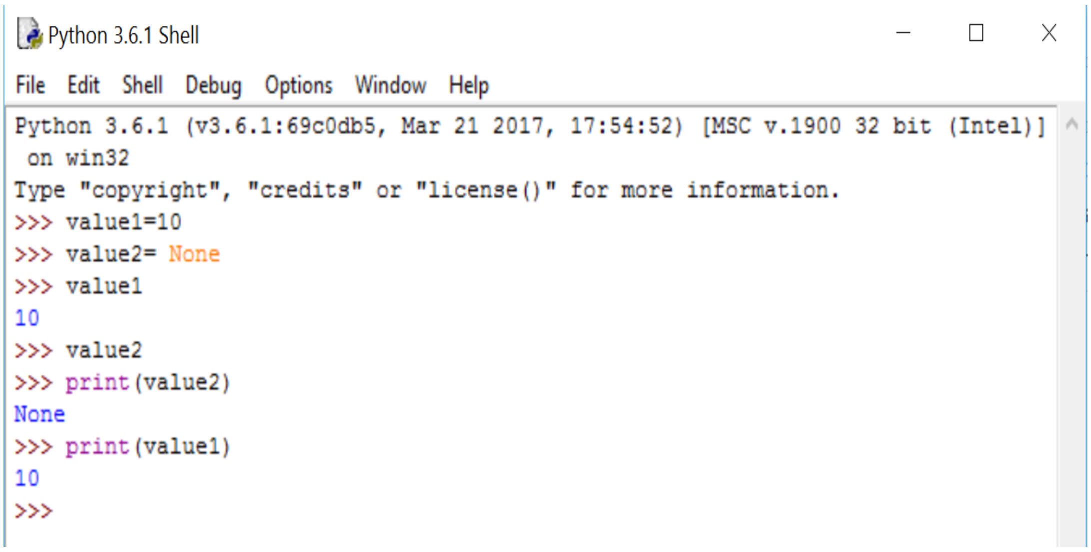
```
Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 17:54:52) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> text= """hello
welcome
to the
python"""
>>> print(text)
hello
welcome
to the
python
>>>
```

- **Boolean literals**

  A Boolean literal can have any of the two values: True or False.

- **Special literals.**

  Python contains one special literal i.e., None. None is used to specify to that field that is not created. It is also used for end of lists in Python.



```
Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 17:54:52) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> value1=10
>>> value2= None
>>> value1
10
>>> value2
>>> print(value2)
None
>>> print(value1)
10
>>>
```

- **Literal Collections.**

  Collections such as tuples, lists and Dictionary are used in Python.

  List:

  List contain items of different data types. Lists are mutable i.e., modifiable. The values stored in List are separated by commas (,) and enclosed within a square brackets ([]). We can store different type of data in a List. Value stored in a List can be retrieved using the slice operator ([] and [:]). The plus sign (+) is the list concatenation and asterisk (*) is the repetition operator.

  ```
  >>> a = [1, 2.2, 'python']
  ```

  Tuple:

  Tuple is an ordered sequence of items same as list.The only difference is that tuples are immutable. Tuples once created cannot be modified. Tuples are used to write-protect data and are usually faster than list as it cannot change dynamically. It is defined within parentheses () where items are separated by commas.

  ```
  >>> t = (5,'program', 1+3j)
  ```

  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.

  ```
  >>> d = {1:'value','key':2}
  >>> type (d)
      <class 'dict'>
  ```

2.6 Variables

- Variables are used to reference values that may be changed in the program.
- Variable is a name of the memory location where data is stored. Once a variable is stored that means a space is allocated in memory.

- ***Assigning values to variables:***

We need not to declare explicitly variable in Python as we do in c, c++, and java.

```
int a, b;
float c, d;
```

When we assign any value to the variable that variable is declared automatically. The statement for assigning a value to a variable is called an assignment statement. In Python, the equal sign (=) is used as the assignment operator. The syntax for assignment statements is as follows:

Variable = expression

An expression represents a computation involving values, variables, and operators that, taken together, evaluate to a value. For example, consider the following code:

```
y = 1                                   # Assign 1 to variable y
Radius = 1.0                            # Assign 1.0 to variable Radius
x = 5 * (3 / 2) + 3 * 2                 # Assign the value of the expression to x
x = y + 1                               # Assign the addition of y and 1 to x
area = radius * radius * 3.14159        # Compute area
```

- ***Simultaneous Assignment***

Python also supports simultaneous assignment in syntax like this:

var1, var2, varn = exp1, exp2, expn

It tells Python to evaluate all the expressions on the right and assign them to the corresponding variable on the left simultaneously. Swapping variable values is a common operation in programming and simultaneous assignment is very useful to perform this operation.
Consider two variables: x and y. How do you write the code to swap their values? A common approach is to introduce a temporary variable as follows:

```
x = 1
y = 2
temp = x                # Save x in a temp variable
x = y                   # Assign the value in y to x
y = temp                # Assign the value in temp to y
```

But you can simplify the task using the following statement to swap the values of x and y.
x, y = y, x # Swap x with y

- ***Multiple Assignment:***

Multiple assignment can be done in Python by assigning single value to multiple variables:

x=y=z=50

2.7 What do mean by data types? Explain various types of data types available in Python.

A data type is a set of values, and a set of operators that may be applied to those values. For example, the integer data type consists of the set of integers, and operators for addition, subtraction, multiplication, and division, among others. Integers, floats, complex numbers and strings are part of a set of predefined data types in Python called the built-in types.

There are various data types in Python. Some of the important types are listed below.

- ***Numbers***

  Integers, floating point numbers and complex numbers falls under Python numbers category. They are defined as int, float and complex class in Python.

  a) Integer (signed) -Numbers (can be both positive and negative) with no fractional part.e.g. 100

  b) Float pointing - Real numbers with both integer and fractional part e.g. -26.

  c) Complex - In the form of a+bj where a forms the real part and b forms the imaginary part of complex number. e.g. 3+4j

- ***Strings***

  String is sequence of Unicode characters. We can use single quotes or double quotes to represent strings. Multi-line strings can be denoted using triple quotes, ''' or """.

  s = "This is a string"

  s = '''a multiline

- ***List***

  List is an ordered sequence of items. It is one of the most used data type in Python and is very flexible. All the items in a list do not need to be of the same type. Declaring a list is pretty straight forward. Items separated by commas are enclosed within brackets [ ].

  a = [1, 2.2, 'python']

  Lists are mutable, meaning, value of elements of a list can be altered.

- ***Tuple***

  Tuple is an ordered sequence of items same as list. The only difference is that tuples are. immutable. Tuples once created cannot be modified. Tuples are used to write-protect data and are usually faster than list as it cannot change dynamically. It is defined within parentheses () where items are separated by commas

  t = (5,'program', 1+3j)

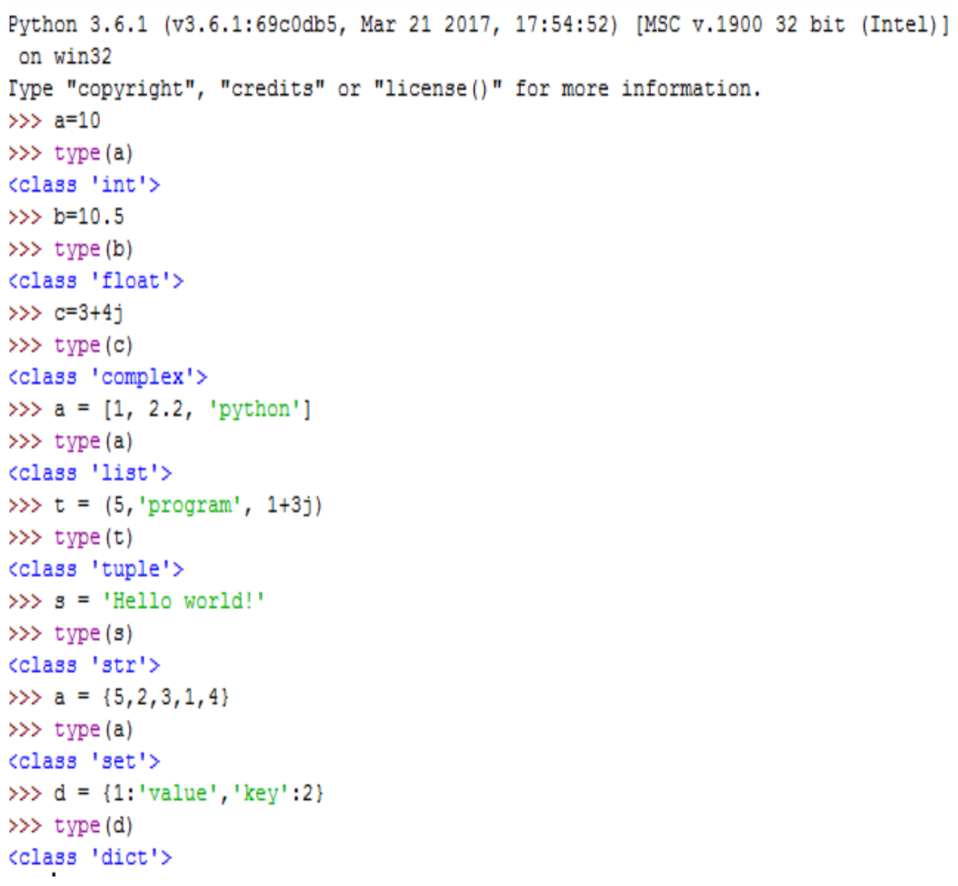
- ***Set***

  Set is an unordered collection of unique items. Set is defined by values separated by comma inside braces { }. Items in a set are not ordered.

- ***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.

Python knows different data types. To find the type of a variable, use the type () function:



```
Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 17:54:52) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=10
>>> type(a)
<class 'int'>
>>> b=10.5
>>> type(b)
<class 'float'>
>>> c=3+4j
>>> type(c)
<class 'complex'>
>>> a = [1, 2.2, 'python']
>>> type(a)
<class 'list'>
>>> t = (5,'program', 1+3j)
>>> type(t)
<class 'tuple'>
>>> s = 'Hello world!'
>>> type(s)
<class 'str'>
>>> a = {5,2,3,1,4}
>>> type(a)
<class 'set'>
>>> d = {1:'value','key':2}
>>> type(d)
<class 'dict'>
```

2.8 What do you mean by an expression?

An expression represents a computation involving values, variables, and operators that, taken together, evaluate to a value. For example, consider the following code:

```
y = 1                              # Assign 1 to variable y
radius = 1.0                       # Assign 1.0 to variable radius
x = 5 * (3 / 2) + 3 * 2            # Assign the value of the expression to x
x = y + 1                          # Assign the addition of y and 1 to x
area = radius * radius * 3.14159   # Compute area
```

You can use a variable in an expression. A variable can also be used in both sides of the = operator. For example,

```
x = x + 1
```

In this assignment statement, the result of x + 1 is assigned to x. If x is 1 before the statement is executed, then it becomes 2 after the statement is executed.

To assign a value to a variable, you must place the variable name to the left of the assignment operator. Thus, the following statement is wrong:

```
1 = x                # Wrong
```

Mixed-type expression

A mixed-type expression is an expression containing operands of different type. The CPU can only perform operations on values with the same internal representation scheme, and thus only on operands of the same type. Operands of mixed-type expressions therefore must be converted to a common type. Values can be converted in one of two ways—by **implicit (automatic) conversion, called coercion, or by explicit type conversion**

Coercion vs. Type Conversion

Coercion is the implicit (automatic) conversion of operands to a common type. Coercion is automatically performed on mixed-type expressions only if the operands can be safely converted, that is, if no loss of information will result. The conversion of integer 2 to floating-point 2.0 below is a safe conversion—the conversion of 4.5 to integer 4 is not, since the decimal digit would be lost,

a=2

b=4.5

c=a+b

2 + 4.5 → 2.0 + 4.5 → 6.5 (automatic conversion of int to float)

Type conversion is the explicit conversion of operands to a specific type. Type conversion can be applied even if loss of information results.

2 + int (4.5) → 2 + 4 → 6

2.9 What Is an Operator? Explain different types of operators available in python.

An operator is a symbol that represents an operation that may be performed on one or more operands. For example, the 1 symbol represents the operation of addition. An operand is a value that a given operator is applied to, such as operands 2 and 3 in the expression 2 + 3. A unary operator operates on only one operand, such as the negation operator in - 12. A binary operator operates on two operands, as with the addition operator.

The different types of operators available in python are:

a) Arithmetic operators

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc.

Operator	Meaning	Example
+	Add two operands or unary plus	5+2 =7 +2
-	Subtract right operand from the left or unary minus	5-2=3 -2
*	Multiply two operands	5*2=10
/	Divide left operand by the right one (always results into float)	5/2=2.5
%	Modulus - remainder of the division of left operand by the right	5%2=1
//	Floor division - division that results into whole number	5 // 2=2
**	Exponent - left operand raised to the power of right	5**2=25

b) Comparison operators

Comparison operators are used to compare values. It either returns True or False according to the condition.

Comparison operators in Python:

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

c) Logical operators

Logical operators are the and, or, not operators.

Logical operators in Python are:

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

d) Bitwise operators

Bitwise operators act on operands as if they were string of binary digits. It operates bit by bit, hence the name.

For example, 2 is 10 in binary and 7 is 111.

In the table below: Let x = 10 (0000 1010 in binary) and y = 4 (0000 0100 in binary)

Bitwise operators in Python are:

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

e) Assignment operators

Assignment operators are used in Python to assign values to variables.

a = 10 is a simple assignment operator that assigns the value 10 on the right to the variable a on the left. There are various compound operators in Python like a += 10 that adds to the variable and later assigns the same. It is equivalent to a = a + 10.

Assignment operators in Python are:

Operator	Example	Equivalent to
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5
|=	x |= 5	x = x | 5
^=	x ^= 5	x = x ^ 5
>>=	x >>= 5	x = x >> 5
<<=	x <<= 5	x = x << 5

f) Special operators

Python language offers some special type of operators like the identity operator or the membership operator. They are described below with examples.

- ***Identity operators***

  is and **is not** are the identity operators in Python. They are used to check if two values (or variables) are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

  Identity operators in Python are:

Operator	Meaning	Example
is	True if the operands are identical (refer to the same object)	x is True
is not	True if the operands are not identical (do not refer to the same object)	x is not True

```
  x1 = 5
  y1 = 5
  x2 = 'Hello'
  y2 = 'Hello'
  x3 = [1,2,3]
y3 = [1,2,3]
```

```
print(x1 is not y1)       #  output is false
print(x2 is y2)           #   output is true
print(x3 is y3)           #    output is false
```

Here, we see that x1 and y1 are integers of same values, so they are equal as well as identical. Same is the case with x2 and y2 (strings).But x3 and y3 are list. They are equal but not identical. Since list are mutable (can be changed), interpreter locates them separately in memory although they are equal.

- ***Membership operators***

  in and **not** in are the membership operators in Python. They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary). In a dictionary we can only test for presence of key, not the value.

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

```
x = 'Hello world'
y = {1:'a', 2:'b'}
print('H' in x)               # output is true
print('hello' not in x)       # output is true
print(1 in y)                 # output is true
print('a' in y)               # output is false
```

Here, 'H' is in x but 'hello' is not present in x (remember, Python is case sensitive). Similarly, 1 is key and 'a' is the value in dictionary y. Hence, 'a' in y returns False.

2.10 Operator Precedence and Associativity

a) *Operator Precedence*

The combination of values, variables, operators and function calls is termed as an expression. The way we commonly represent expressions, in which operators appear between their operands, is referred to as infix notation. For example, the expression 4 + 3 is in infix notation since the + operator appears between its two operands, 4 and 3. There are other ways of representing expressions called prefix and postfix notation, in which operators are placed before and after their operands, respectively.

The expression 4 + (3 * 5) is also in infix x notation. It contains two operators, + and *. The parentheses denote that (3 * 5) is a sub expression. Therefore, 4 and (3 * 5) are the operands of the addition operator, and thus the overall expression evaluates to 19.*To evaluate these type of expressions there is a rule of precedence in Python. It guides the order in which operation are carried out.*

Operator precedence rule in Python:

Operators	Meaning
()	Parentheses
**	Exponent
+x, -x, ~x	Unary plus, Unary minus, Bitwise NOT
*, /, //, %	Multiplication, Division, Floor division, Modulus
+, -	Addition, Subtraction
<<, >>	Bitwise shift operators
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
==,! =, >, >=, <, <=, is, is not, in, not in	Comparisons, Identity, Membership operators
not	Logical NOT
and	Logical AND
or	Logical OR

b) Operator Associativity

We can see in the above table that more than one operator exists in the same group. These operators have the same precedence. When two operators have the same precedence, associativity helps to determine which the order of operations.

Associativity is the order in which an expression is evaluated that has multiple operator of the same precedence. Almost all the operators have left-to-right associativity.

For example, multiplication and floor division have the same precedence. Hence, if both of them are present in an expression, left one is evaluates first.

```
# Left-right associativity

print(5 * 2 // 3)        # Output: 3

# Shows left-right associativity

print(5 * (2 // 3))      # Output: 0
```

c) Non associative operators

Some operators like assignment operators and comparison operators do not have associativity in Python. There are separate rules for sequences of this kind of operator and cannot be expressed as associativity. For example, x < y < z neither means (x < y) < z nor x < (y < z). x < y < z is equivalent to x < y and y < z, and is evaluates from left-to-right.

2.11 Comments

Python, comments are preceded by a pound sign (#) on a line, called a line comment, or enclosed between three consecutive single quotation marks (''') on one or several lines, called a paragraph comment.

When the Python interpreter sees #, it ignores all text after # on the same line. When it sees ''', it scans for the next ''' and ignores any text between the triple quotation marks. Here are examples of comments:

```
# This program displays Welcome to Python

''' This program displays Welcome to Python and

   Python is fun

'''
```

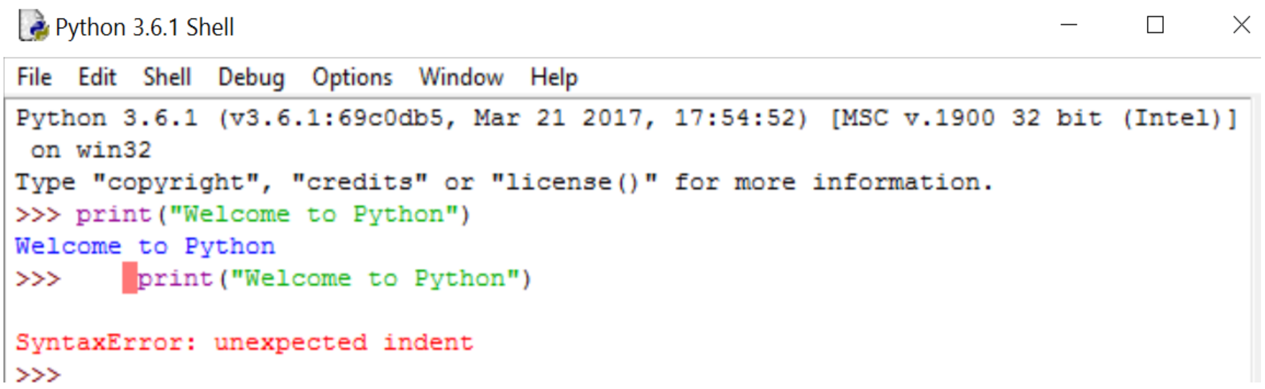
2.12 Indentation

Indentation matters in Python. Note that the statements are entered from the first column in the new line. The Python interpreter will report an error if the program is typed as follows:

```
# Display two messages

   print("Welcome to Python")

        print("Python is fun")
```



```
Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 17:54:52) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Welcome to Python")
Welcome to Python
>>>      print("Welcome to Python")

SyntaxError: unexpected indent
>>>
```

Don't put any punctuation at the end of a statement. For example, the Python interpreter will report errors for the following code:

```
# Display two messages

   print("Welcome to Python").

   print("Python is fun"),
```

2.13 Different types of Programming Errors

Programming errors can be categorized into three types: *syntax errors, runtime errors, and logic errors.*

- *Syntax Errors*

  The most common error you will encounter are syntax errors. Like any programming language, Python has its own syntax, and you need to write code that obeys the syntax rules. If your program violates the rules—for example, if a quotation mark is missing or a word is misspelled—Python will report syntax errors. Syntax errors result from errors in code construction, such as mistyping a statement, incorrect indentation, omitting some necessary punctuation, or using an opening parenthesis without a corresponding closing parenthesis. These errors are usually easy to detect, because Python tells you where they are and what caused them.

- *Runtime Errors*

Runtime errors are errors that cause a program to terminate abnormally. They occur while a program is running if the Python interpreter detects an operation that is impossible to carry out. Input mistakes typically cause runtime errors. An input error occurs when the user enters a value that the program cannot handle. For instance, if the program expects to read in a number, but instead the user enters a string of text, this causes data-type errors to occur in the program. Another common source of runtime errors is division by zero. This happens when the divisor is zero for integer divisions.

- *Logic errors*

Logic errors occur when a program does not perform the way it was intended to. Errors of this kind occur for many different reasons. For example, suppose you wrote the program, to convert a temperature (35 degrees) from Fahrenheit to Celsius.

```
 # Convert Fahrenheit to Celsius

 print("Fahrenheit 35 is Celsius degree ")

 print(5 / 9 * 35 - 32)

#Output

Fahrenheit 35 is Celsius degree

-12.555555555555554
```

You will get Celsius –12.55 degrees, which is wrong. It should be 1.66. To get the correct result, you need to use 5 / 9 * (35 – 32) rather than 5 / 9 * 35 – 32 in the expression. That is, you need to add parentheses around (35 – 32) so Python will calculate that expression first before doing the division. In Python, syntax errors are actually treated like runtime errors because they are detected by the interpreter when the program is executed. In general, syntax and runtime errors are easy to find and easy to correct, because Python gives indications as to where the errors came from and why they are wrong.