2.1 Identifiers

An identifier is a name given to a variable, function, class or module. Identifiers may be one or more characters in the following format:

- 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 myCountry, other_1 and good_ morning, all are valid examples. A Python identifier can begin with an alphabet (A – Z and a – z and _).
- An identifier cannot start with a digit but is allowed everywhere else. 1plus is invalid, but plus1 is perfectly fine.

2.2 Keywords

Keywords are a list of reserved words that have predefined meaning. Keywords are special vocabulary and cannot be used by programmers as identifiers for variables, functions, constants or with any identifier name. Attempting to use a keyword as an identifier name will cause an error. The following TABLE 2.1 shows the Python keywords.

TABLE 2.1

List of Keywords in Python

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

2.3 Statements and Expressions

A statement is an instruction that the Python interpreter can execute. Python program consists of a sequence of statements. Statements are everything that can make up a line (or several lines) of Python code. For example, z = 1 is an assignment statement.

Expression is an arrangement of values and operators which are evaluated to make a new value. Expressions are statements as well. A value is the representation of some entity like a letter or a number that can be manipulated by a program. A single value >>> 20 or a single variable >>> z or a combination of variable, operator and value >>> z + 20 are all examples of expressions. An expression, when used in interactive mode is evaluated by the interpreter and result is displayed instantly. For example,

```
>>> 8 + 2
10
```

But the same expression when used in Python program does not show any output altogether. You need to explicitly print the result.

2.4 Variables

Variable is a named placeholder to hold any type of data which the program can use to assign and modify during the course of execution. In Python, there is no need to declare a variable explicitly by specifying whether the variable is an integer or a float or any other type. *To define a new variable in Python, we simply assign a value to a name.* If a need for variable arises you need to think of a variable name based on the rules mentioned in the following subsection and use it in the program.

2.4.1 Legal Variable Names

Follow the below-mentioned rules for creating legal variable names in Python.

- Variable names can consist of any number of letters, underscores and digits.
- Variable should not start with a number.
- Python Keywords are not allowed as variable names.
- Variable names are case-sensitive. For example, computer and Computer are different variables.

Also, follow these guidelines while naming a variable, as having a consistent naming convention helps in avoiding confusion and can reduce programming errors.

- Python variables use lowercase letters with words separated by underscores as necessary to improve readability, like this whats_up, how_are_you. Although this is not strictly enforced, it is considered a best practice to adhere to this convention.
- Avoid naming a variable where the first character is an underscore. While this is legal in Python, it can limit the interoperability of your code with applications built by using other programming languages.
- Ensure variable names are descriptive and clear enough. This allows other programmers to have an idea about what the variable is representing.

2.4.2 Assigning Values to Variables

The general format for assigning values to variables is as follows:

variable_name = expression

The equal sign (=) also known as simple assignment operator is used to assign values to variables. In the general format, the operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the expression which can be a value or any code snippet that results in a value. That value is stored in the variable on the execution of the assignment statement. Assignment operator should not be confused with the = used in algebra to denote equality. For example, enter the code shown below in interactive mode and observe the results.

1. >>> number =100
2. >>> miles =1000.0

```
3. >>> name ="Python"
4. >>> number
   100
5. >>> miles
   1000.0
6. >>> name
   'Python'
```

In ① integer type value is assigned to a variable *number*, in ② float type value has been assigned to variable *miles* and in ③ string type value is assigned to variable *name*. ④, ⑤ and ⑥ prints the value assigned to these variables.

In Python, not only the value of a variable may change during program execution but also the type of data that is assigned. You can assign an integer value to a variable, use it as an integer for a while and then assign a string to the variable. A new assignment overrides any previous assignments. For example,

```
1. >>> century = 100
2. >>> century
   100
3. >>> century = "hundred"
4. >>> century
   'hundred'
```

In ① an integer value is assigned to *century* variable and then in ③ you are assigning a string value to *century* variable. Different values are printed in each case as seen in ② and ④.

Python allows you to assign a single value to several variables simultaneously. For example,

```
1. >>> a = b = c =1
2. >>> a
   1
3. >>> b
   1
4. >>> c
   1
```

An integer value is assigned to variables a, b and c simultaneously ①. Values for each of these variables are displayed as shown in ②, ③ and ④.

2.5 Operators

Operators are symbols, such as +, −, =, >, and <, that perform certain mathematical or logical operation to manipulate data values and produce a result based on some rules. An operator manipulates the data values called operands.

Consider the expression,

```
>>> 4 + 6
```

where 4 and 6 are operands and + is the operator.

Python language supports a wide range of operators. They are

1. Arithmetic Operators
2. Assignment Operators
3. Comparison Operators
4. Logical Operators
5. Bitwise Operators

2.5.1 Arithmetic Operators

Arithmetic operators are used to execute arithmetic operations such as addition, subtraction, division, multiplication etc. The following TABLE 2.2 shows all the arithmetic operators.

TABLE 2.2

List of Arithmetic Operators

Operator	Operator Name	Description	Example
+	Addition operator	Adds two operands, producing their sum.	p + q = 5
–	Subtraction operator	Subtracts the two operands, producing their difference.	p – q = –1
*	Multiplication operator	Produces the product of the operands.	p * q = 6
/	Division operator	Produces the quotient of its operands where the left operand is the dividend and the right operand is the divisor.	q / p = 1.5
%	Modulus operator	Divides left hand operand by right hand operand and returns a remainder.	q % p = 1
**	Exponent operator	Performs exponential (power) calculation on operators.	p**q = 8
//	Floor division operator	Returns the integral part of the quotient.	9//2 = 4 and 9.0//2.0 = 4.0

Note: The value of p is 2 and q is 3.

For example,

1. >>> 10+35
   45
2. >>> –10+35
   25
3. >>> 4*2
   8
4. >>> 4**2
   16

```
5. >>> 45/10
   4.5
6. >>> 45//10.0
   4.0
7. >>> 2025%10
   5
8. >>> 2025//10
   202
```

Above code illustrates various arithmetic operations ①–⑧.

2.5.2 Assignment Operators

Assignment operators are used for assigning the values generated after evaluating the right operand to the left operand. Assignment operation always works from right to left. Assignment operators are either simple assignment operator or compound assignment operators. Simple assignment is done with the equal sign (=) and simply assigns the value of its right operand to the variable on the left. For example,

```
1. >>> x = 5
2. >>> x = x + 1
3. >>> x
   6
```

In ① you assign an integer value of 5 to variable *x*. In ② an integer value of 1 is added to the variable *x* on the right side and the value 6 after the evaluation is assigned to the variable *x*. The latest value stored in variable *x* is displayed in ③.

Compound assignment operators support shorthand notation for avoiding the repetition of the left-side variable on the right side. Compound assignment operators combine assignment operator with another operator with = being placed at the end of the original operator.

For example, the statement

```
>>> x = x + 1
```

can be written in a compactly form as shown below.

```
>>> x += 1
```

If you try to update a variable which doesn't contain any value, you get an error.

```
1. >>> z = z + 1
   NameError: name 'z' is not defined
```

Trying to update variable z which doesn't contain any value results in an error because Python evaluates the right side before it assigns a value to z ①.

```
1. >>> z = 0
2. >>> x = z + 1
```

Before you can update a variable ②, you have to assign a value to it ①.

The following TABLE 2.3 shows all the assignment operators.

TABLE 2.3

List of Assignment Operators

Operator	Operator Name	Description	Example
=	Assignment	Assigns values from right side operands to left side operand.	z = p + q assigns value of p + q to z
+=	Addition Assignment	Adds the value of right operand to the left operand and assigns the result to left operand.	z += p is equivalent to z = z + p
-=	Subtraction Assignment	Subtracts the value of right operand from the left operand and assigns the result to left operand.	z -= p is equivalent to z = z - p
*=	Multiplication Assignment	Multiplies the value of right operand with the left operand and assigns the result to left operand.	z *= p is equivalent to z = z * p
/=	Division Assignment	Divides the value of right operand with the left operand and assigns the result to left operand.	z /= p is equivalent to z = z / p
**=	Exponentiation Assignment	Evaluates to the result of raising the first operand to the power of the second operand.	z**= p is equivalent to z = z ** p
//=	Floor Division Assignment	Produces the integral part of the quotient of its operands where the left operand is the dividend and the right operand is the divisor.	z //= p is equivalent to z = z // p
%=	Remainder Assignment	Computes the remainder after division and assigns the value to the left operand.	z %= p is equivalent to z = z % p

For example,

```
1. >>> p = 10
2. >>> q = 12
3. >>> q += p
4. >>> q
   22
5. >>> q *= p
6. >>> q
   220
7. >>> q /= p
8. >>> q
   22.0
9. >>> q %= p
10. >>> q
    2.0
11. >>> q **= p
12. >>> q
    1024.0
13. >>> q //= p
14. >>> q
    102.0
```