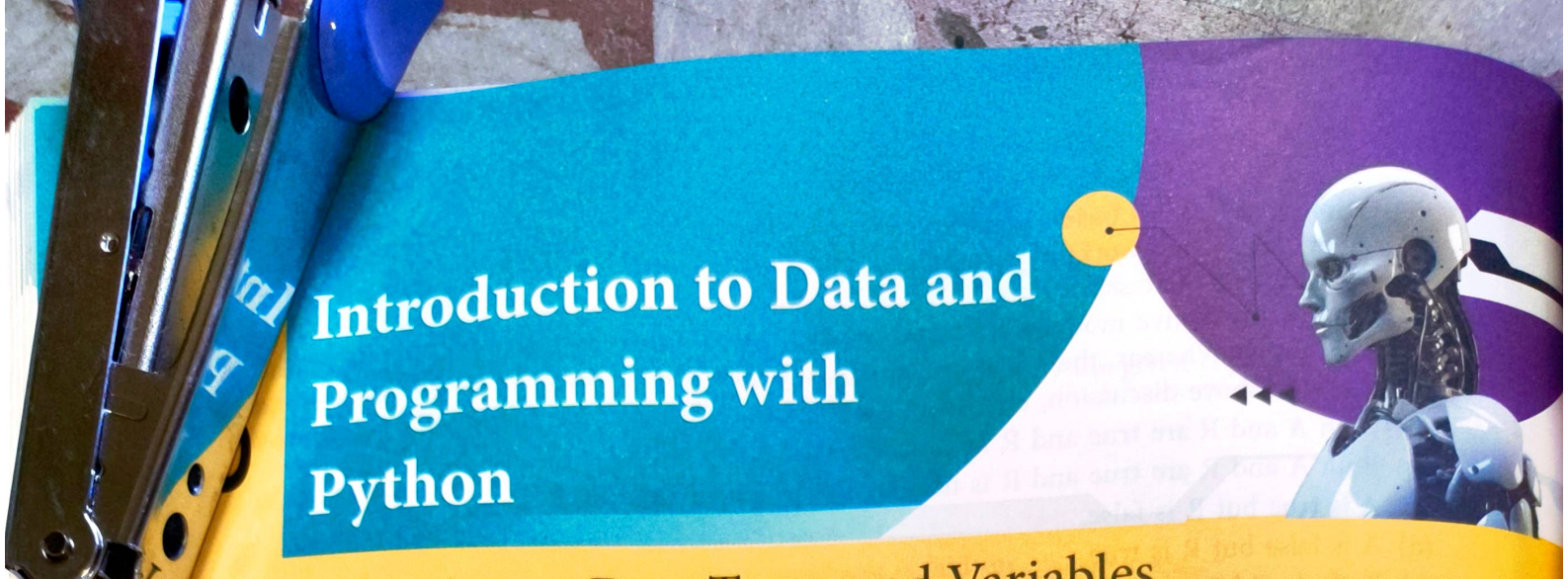


Introduction to Data and Programming with Python

3.2 Introduction to Data Types and Variables

Learning Scope

Introduction, Basic structure of Python programming, Indentation in Python programming, Key points about indentation, Data types in Python viz. numeric (int, float), Boolean, Sequence type (tuple, list, strings), Sets and Dictionary, Introduction to variables and assignment of values.

INTRODUCTION

In the previous chapter, you learnt some elementary aspects of Python. It includes some basic elements that are frequently used in Python programming viz. character sets, literals, identifiers, keywords, punctuators. You also learnt that Python works in two different modes viz. Interactive and Script.

In this segment, you will learn more about Python programming that includes data types in Python, variables and assignments.

Let us understand the concept in details.

BASIC STRUCTURE OF PYTHON PROGRAM

It must be noted that every programming language has its own basic structure to design a program to perform some specific tasks. In the context of Python programming, we should also be aware of basic structure of Python programming as a good programmer. The basic structure of a Python program consists of the following components:

- **Comments:** Comments are non-executable statements. They are used to explain the purpose of the code or to make notes for other programmers. In Python programming, it starts with a '#' symbol and are ignored by the interpreter.
- **Import Statements:** While designing Python programs, import statements are used to include modules or libraries into the program. These modules contain predefined functions that can be used to accomplish tasks.
- **Variables:** The variables are one of the most significant components. They are used to store data in memory to perform various arithmetical tasks and also for later use.
- **Data Types:** Python supports several built-in data types including integers, floats, strings, boolean, lists, tuple, etc. to perform various tasks during programming.

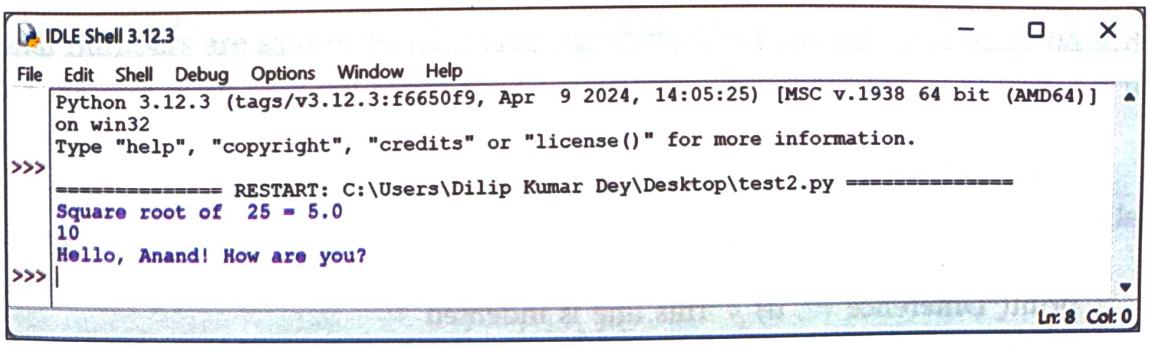
- **Operators:** Operators are the symbols or tokens to perform various mathematical or logical operations on data. Python supports arithmetic, comparison and logical operators for these purposes.
- **Control Structures:** Control structures are used to control the flow of instructions in programs. The various Python control statements include if-else statements, for loops and while loops.
- **Functions:** Functions are used to group a set of related statements together and give them a name. They can be reused throughout a program.

On the whole, we should be aware of the basic structure of a Python program which consists of these components working together to accomplish a specific task or solve a particular problem. These components are used as per the need of a program.

Here's an example of a simple Python program that demonstrates the basic structure of Python:

```
# A comment to illustrate basic structure of Python program
import math # Importing a module
# Defining variables
x = 10
y = 12
z = 25
# Performing arithmetic operation
result = math.sqrt(z)
print("Square root of", z, "=", result)
# use of if-else statement
if (x<y):
    print(x)
else:
    print(y)
# Defining a function
def welcome(name):
    print("Hello," + name+ "How are you?")
# Using Python function
welcome("Anand!")
```

Output:



In this example, we start with a comment that explains the purpose of the code. Thereafter, we import the math module to find out the square root of an integer which is to be displayed in the later part of the program.

Next, we define three variables that store three integer values. We then perform an arithmetic operation on the variable **z** and stores the answer in the variable result.

Then, we have applied a conditional statement with the help of an if-else statement to check the smaller of two integers between **x** and **y** to print the value of the variable accordingly.

Finally, we define a function called welcome( ) that takes a parameter called name to print a greeting message. We call the function with the argument "Anand" to print an appropriate message when executed.

Indentation in Python Programming

Indentation is the fundamental aspect of Python programming that defines the structure and flow of code. Unlike many other programming languages that use curly braces { } or keywords to delimit blocks of code, Python relies on indentation levels to group statements.

Thus, the indentation refers to the aligning of a statement or group of statements which are part of another statement to get the desired execution of the specific block.

Key Points about Indentation in Python:

- **Indentation defines code blocks:**

  In Python, code blocks are defined by their indentation level. All the statements within a block must be indented by the same amount. The common constructs that require indentation include:

  - ➤ if, elif, else statements
  - ➤ def for defining functions
  - ➤ try, except, finally for exception handling
  - ➤ for, while loops
  - ➤ class for defining classes

- **Consistent Indentation:**
  - ➤ All lines of code within a block must have the same level of indentation.
  - ➤ Python does not enforce a specific number of spaces, but it is common practice to use 4 spaces per indentation level. Mixing tabs and spaces is not allowed and will result in an Indentation Error.

  It must be noted, if the indentation is incorrect, Python will raise an Indentation Error. This can happen, if the code is not indented at all, if the indentation levels are inconsistent or if there is unexpected indentation.

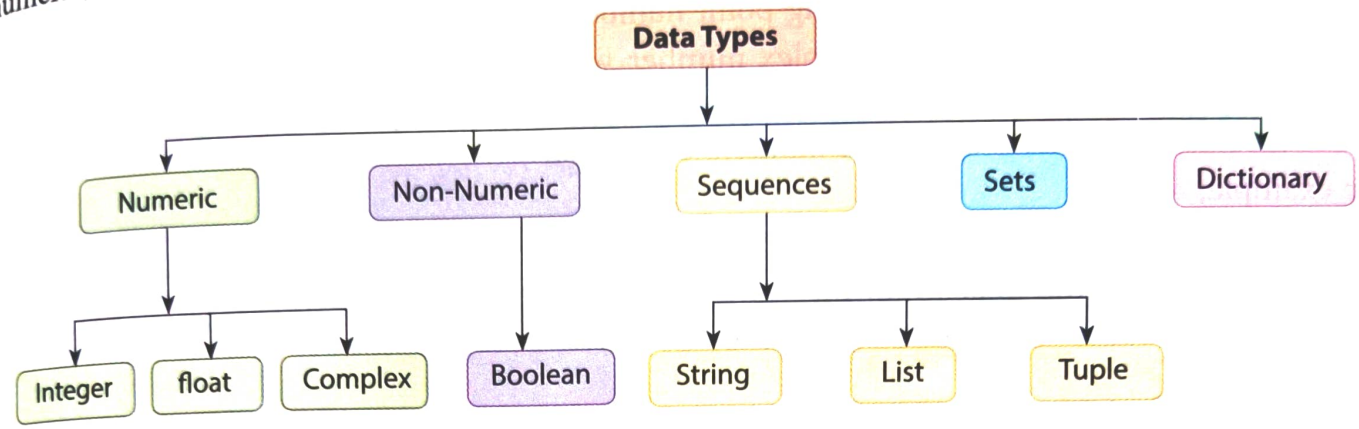
Let us understand the significance of indentation with the help of an example:

```
# a program to illustrate indentation in an if-else statement
a = 45
b = 60
if a > b:
    d = a - b # This line is indented
    print("Difference =", d) # This line is indented
else:
    d = b - a # This line is indented
    print("Difference =", d) # This line is indented
    ←——Indentation
```

Therefore, the proper indentation is crucial in Python as it directly affects the execution of the code and readability.

DATA TYPES IN PYTHON

The data types are referred to as the classification or categorisation of data items. They represent a particular type of value which determines what operations can be performed on that particular data. Python also promotes programming with their built-in data types. It mainly includes numeric, non-numeric (Boolean literals, dictionary and sequences, illustrated as under:



Let us discuss these data types in brief.

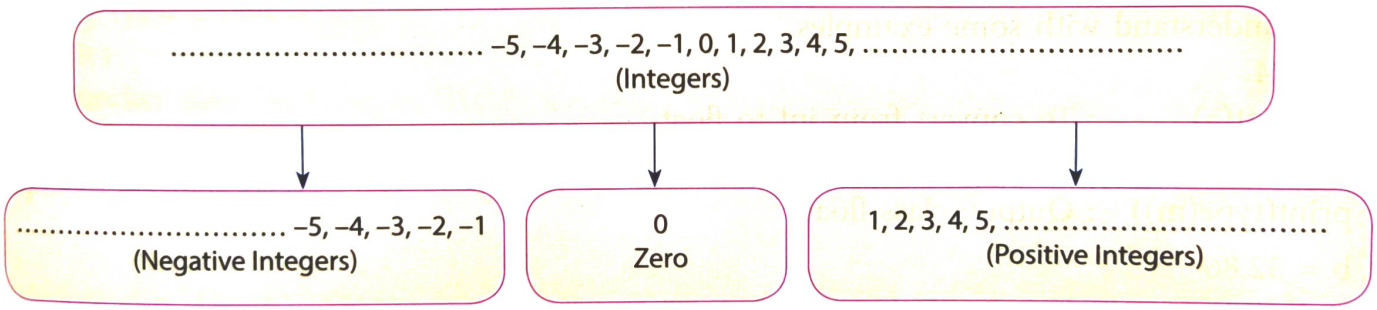
Numeric

It is referred to the representation of data which has a numeric value.

Python interpreter is designed in such a way that it automatically defines the data type of the variable based on the value assigned in that variable. There are three types of numbers under this category. They are:

• Integers

All whole numbers (positive or negative numbers) without a decimal point are said to be *int* type numbers. They are also called fixed point numbers. The integer numbers are shown below:

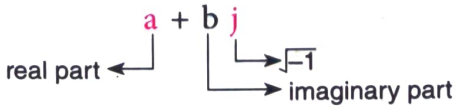


• Float

All fractional numbers are said to be *float* type numbers. They are called floating point numbers because the placement of decimal point does not have a fixed position.

• Complex Number

Complex number is the combination of a real number and an imaginary number. In Python, it is expressed as:



Where, a represents a real number and b represents the imaginary part, with j.

For example, (4+i), (15+3i), (–8+5i) are some complex numbers.

Numeric Type Conversion

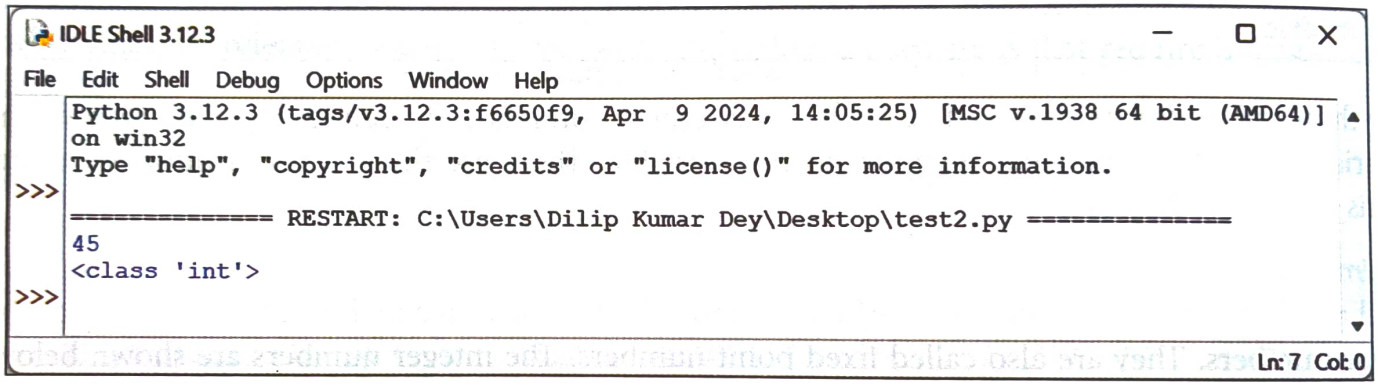
Sometimes, you may need to convert a number explicitly from one type to another. To do so, Python provides the following ways, by which you can convert a number from one data type to another.

- **int(p)** : converts p to an integer
- **float(p)** : converts p to a floating-point number
- **complex(p)** : converts p to a complex number with real part p and imaginary part zero
- **complex(p,q)** : converts p and q to a complex number with real part p and imaginary part q (where, p and q are numeric expressions)

For example,

```
p = 45.24
q = int(p) # converting float to an int
print(q)
print(type(q))
```

Output:



Let us understand with some examples.

1. a = 24
   - m=float(a) : To convert from int to float
   - print(m) : Output: 24.0
   - print(type(m)) : Output: class float
2. b = 32.86
   - n=int(b) : To convert from float to int
   - print(n) : Output: 32
   - print(type(n)) : Output: class int
3. c = 3
   - p=complex(c) : To convert from int to complex
   - print(p) : Output: 3+0j
   - print(type(p)) : Output: class complex
4. a=3; b=2
   - d=complex(a,b) : To convert from int to complex
   - print(d) : Output: 3+2j
   - print(type(d)) : Output: class complex



Knowledge Corner

In Python, the type( ) is a built-in function. It returns the type of the data element stored in variables.

Implicit and Explicit Type Conversion

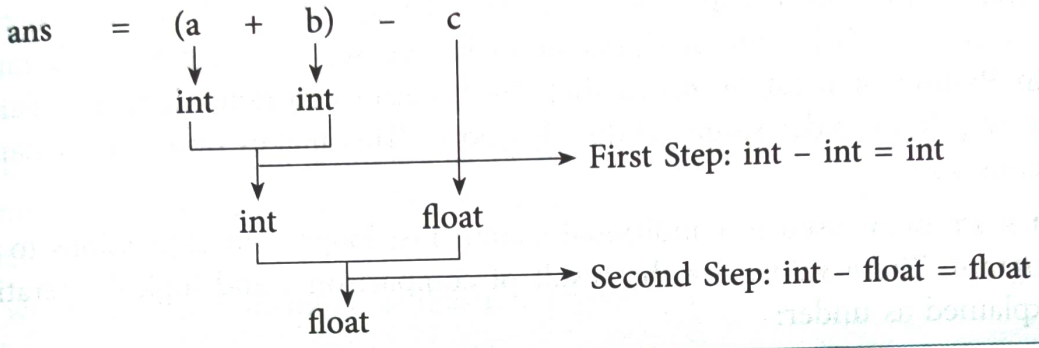
When the various data types into a single type, such conversion is termed as *Type Conversion*. In Python, the type conversion takes place in the following two ways:

Implicit Type Conversion

In a mixed mode expression, the result gets automatically converted into the highest data type available in the expression, without any intervention from the user. This system of conversion is known as *implicit type conversion or coercion.*

For example,

a =15; b = 4; c =12.4



Thus, the above statement will result in 6.6, when executed.

Just to Remember

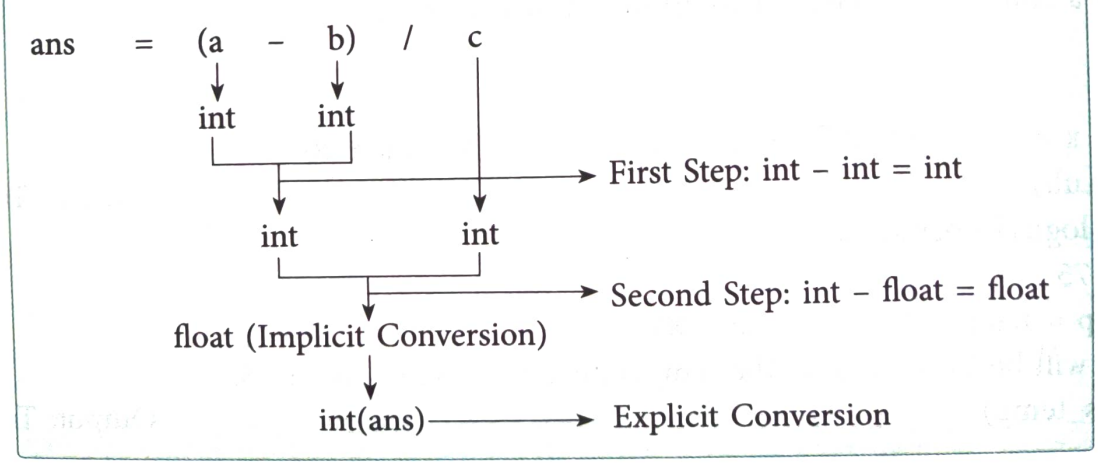
In Python, *complex* is the highest data type whereas the *int* is the lowest data type). If an expression is a combination of all the three data types then the result will be a complex type.

Explicit Type Conversion

Explicit type conversion is another way of type conversion, in which one data type gets converted into another data type depending upon the user's choice. It means when the data type gets converted into another data type after the user's intervention, the type conversion is known as *explicit type conversion or type casting.*

For example,

a =32; b = 12; c = 4



When the above expression is executed, it will result in 5.0 (float type), which is an implicit conversion. But, *int* provided by the user before the expression will produce the result to be forcibly converted to an *int* type. Hence, this type of conversion is referred to as *type casting*.

Just to Remember

Type casting is applicable in converting the data type from a higher data type to a lower data type or vice-versa.

Boolean

In Python, the Boolean constants are represented as True or False. These constants are written with their initial characters in capital (*i.e.*, 'T' or 'F') respectively. It means that the constant written as true or false (with initial character in lowercase) is wrong and will raise an error when used in Python. It must be noted that the Boolean constants True and False are used without enclosing them under single or double quotes. This makes Boolean constants different from string constants.

Boolean values are often used in conditional statements, loops, and expressions to control the flow of a program. These values are the result of comparison-s and logical operations. It can further be explained as under:

- **Common comparison operators include:**
  - == (equal to)
  - != (not equal to)
  - > (greater than)
  - < (less than)
  - >= (greater than or equal to)
  - <= (less than or equal to)
- **Logical operators include:**
  - and: It returns True, if both operands are True.
  - or: It returns True, if at least one operand is True.
  - not: It returns True, if the operand is False (and vice versa).
- **In Conditional Statements:**

  Booleans are commonly used in if, elif and else statements to control program flow.

  For example,

```
# using a comparison operator to create a Boolean expression
x = 10
y = 20
result = x < y # result will be True because 10 is less than 20
print(result)
```

Output: True

```
# using logical operations
temp = 75
res_temp = temp > 60 and temp < 85
# result will be True because the temperature is between 60 and 85
print(res_temp)
```

Output: True

Sequence

In Python, a sequence is referred to an ordered collection of similar or different data types. It provides the following built-in sequence data types, which are mentioned as under:

String

In Python, a string is defined as a collection of one or more characters enclosed within single, double or triple quotes.

For example,

wd = "Artificial Intelligence"

"Artificial Intelligence" — a string

wd — a string variable

When a string is assigned to a variable, each character will set itself to an index and can be accessed individually with reference to the memory locations.

Let us understand the internal structure of Python strings, when stored in a variable.

Suppose, you want to store a word "PYTHON" as a string. It is stored as:

wd="PYTHON"

This word will be stored in memory, as illustrated below:

Forward Indexing

0	1	2	3	4	5
P	Y	T	H	O	N
-6	-5	-4	-3	-2	-1

Backward Indexing

With reference to the memory locations, you can observe the following:

- Each character of the string will be assigned a memory location.
- In the forward direction, the index starts with 0 (zero).
- In the backward direction, the index starts with –1.
- Any character of the string can be accessed with reference to their respective indexing.

To access each character of the string, perform the following tasks:

- **Forward indexing:**

Index	Character accessed
wd[0]	P
wd[1]	Y
wd[2]	T
wd[3]	H
wd[4]	O
wd[5]	N

- **Backward indexing:**

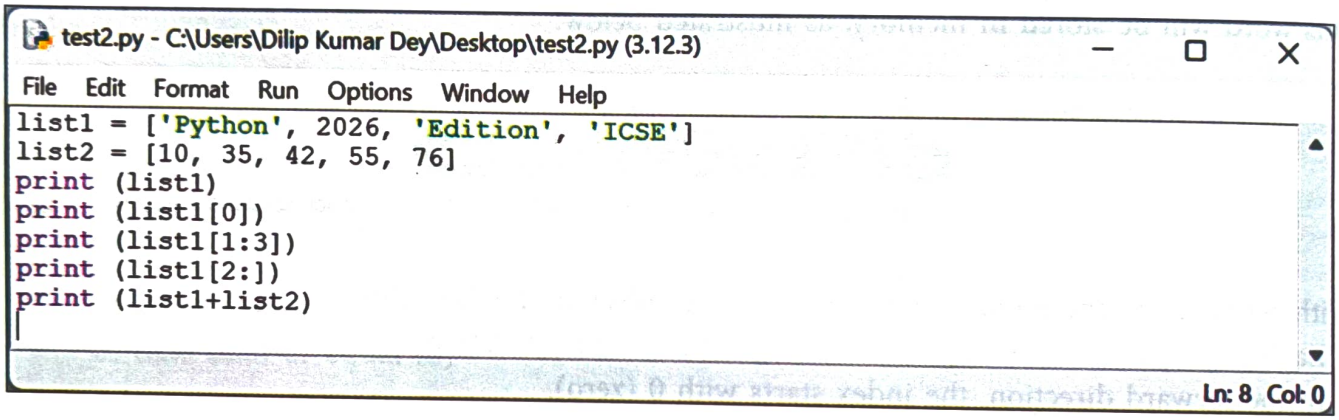
Index	Character accessed
wd[-6]	P
wd[-5]	Y
wd[-4]	T
wd[-3]	H
wd[-3]	O
wd[-1]	N

List

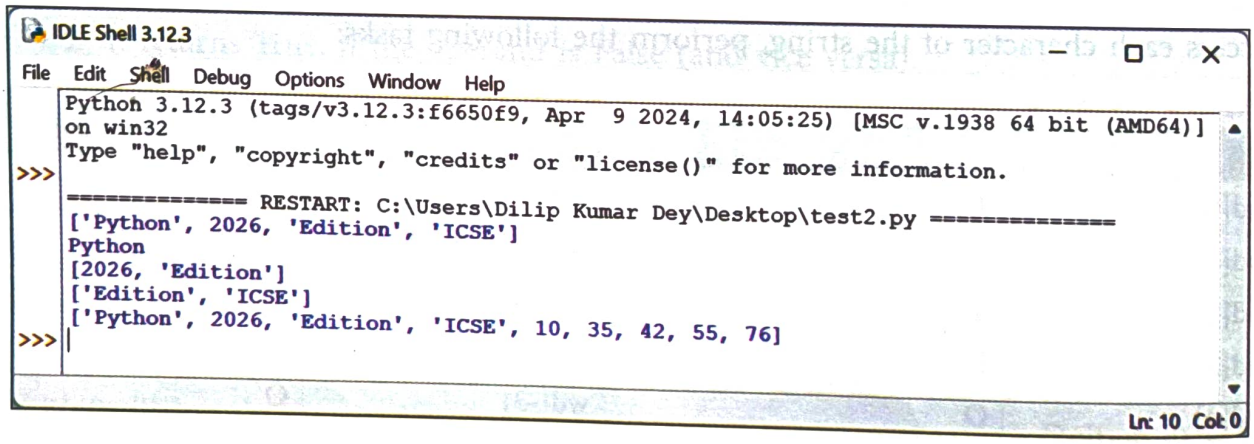
A list is defined as the ordered sequence of one or more data items of the same or different data types. The data items in a list are separated by commas and enclosed within square brackets ([ ]). One of the most important features of the list is that it is mutable (*i.e.*, an element can be altered in its place). The values stored in a list can be accessed with their indices, starting from the 0th index till the last index (*i.e.* (length –1)th index) (same as in strings).

Let us understand the concept of accessing List elements with the help of following illustration:
For example,

list1 = ['Python', 2026, 'Edition', 'ICSE']
list2 = [10, 35, 42, 55, 76]
print (list1) : It will display complete list.
print (list1[0]) : It will display first element of the list.
print (list1[1:3]) : It will display the elements starting from 1st index up to 2nd index.
print (list1[2:]) : It will display the elements starting from 2nd index till last.
print (list1+list2) : It will display the concatenated lists.
The above program is as under:



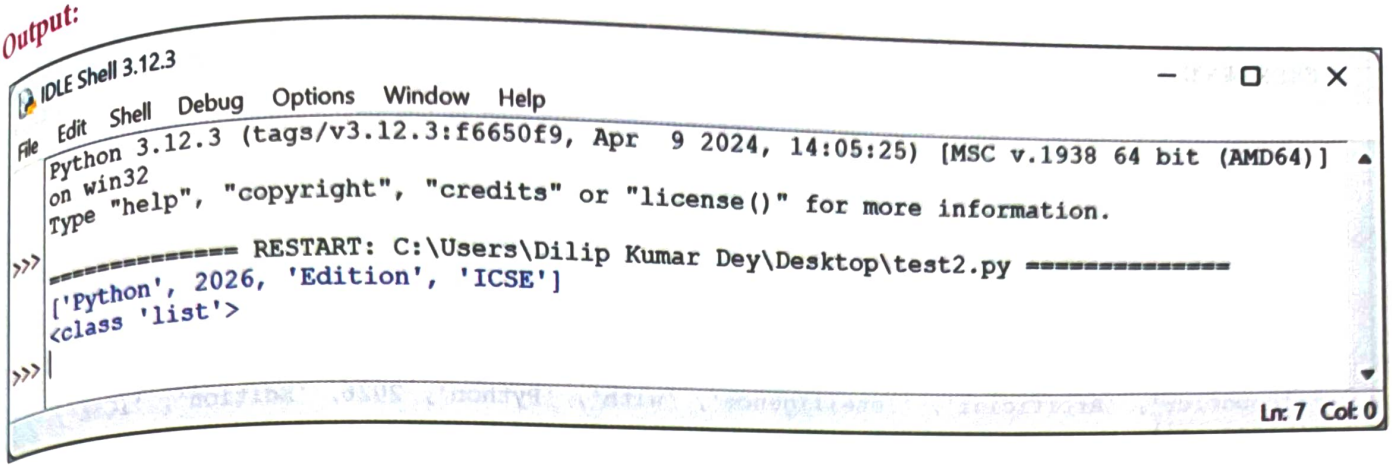
Output:



Further, you can also use type( ) function that will let you know the data type of the above assignment.
For example,

list1 = ['Python', 2026, 'Edition', 'ICSE']
print(list1)
print(type(list1))

Output:



Tuple

Like the list, tuple is also an ordered set of elements containing one or more data items of the same or different types. The data items in a tuple are separated by commas but enclosed within parentheses ( ). The main difference between lists and tuples is that a list is mutable (*i.e.*, it can be modified and updated) whereas, a tuple is immutable (*i.e.*, the elements cannot be updated).

The values stored in a tuple can be accessed starting from 0th index till the last element (*i.e.*, (length – 1)th index) (same as in strings).

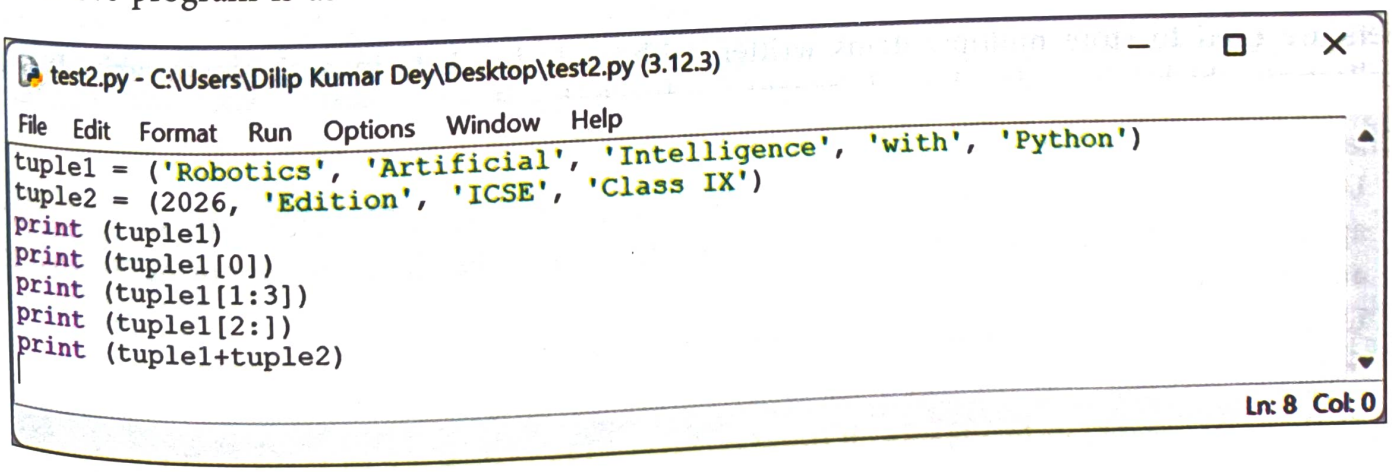
Let us understand the concept of accessing Tuple elements with the help of following illustration:

For example:

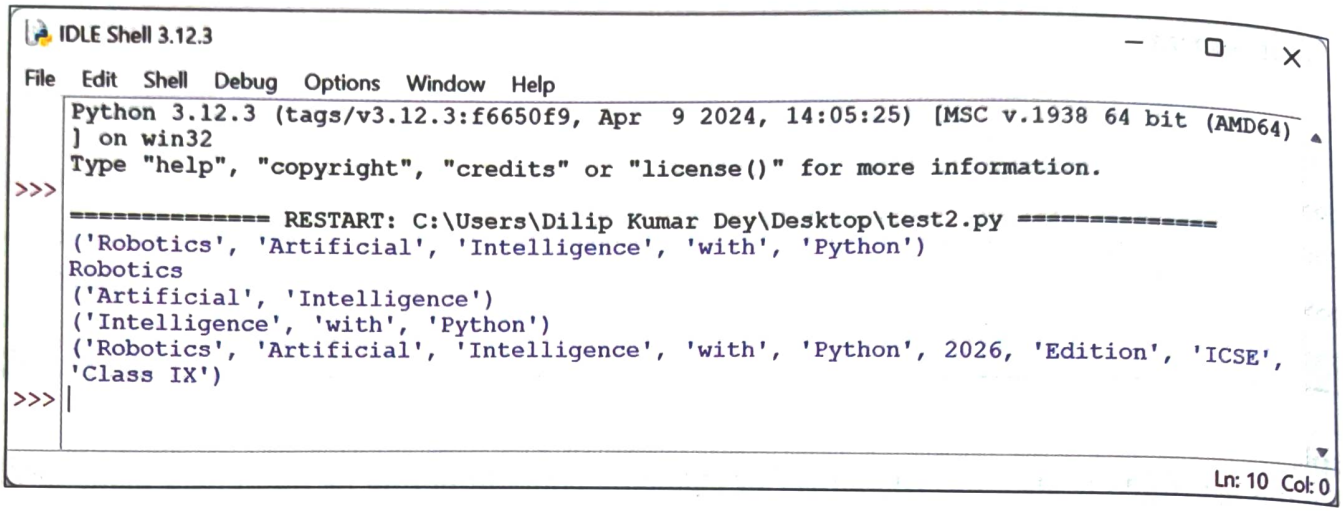
tuple1 = ('Robotics', 'Artificial', 'Intelligence', 'with', 'Python')
tuple2 = (2026, 'Edition', 'ICSE', 'Class IX')

print (tuple1) : It will display complete elements of the tuple.
print (tuple1[0]) : It will display first element of the tuple.
print (tuple1[1:3]) : It will display the elements starting from 1st index up to 2nd index.
print (tuple1[2:]) : It will display the elements starting from 2nd index till last.
print (tuple1+tuple2) : It will display concatenated tuples.

The above program is as under:



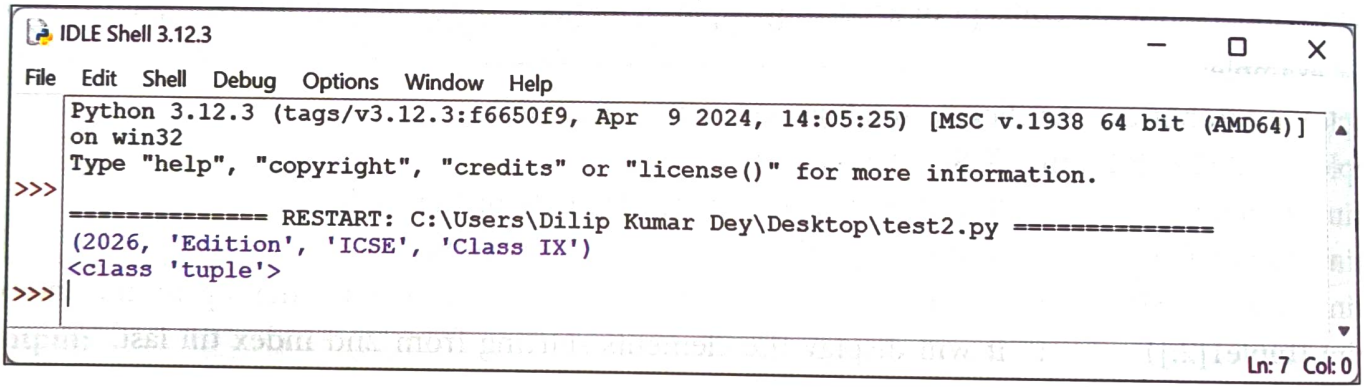
Output:



Similarly, you can also determine the data type of above assignment by using type( ) function. For example,

```
tuple2 = (2026, 'Edition', 'ICSE', 'Class IX')
print(tuple2)
print(type(tuple2))
```

Output:



Set

In Python, a set is a built-in data type that represents an unordered collection of unique elements. Sets are used to store multiple items written with curly brackets in a single variable. It is a collection which is unordered, unchangeable and unindexed.

Characteristics of a Set

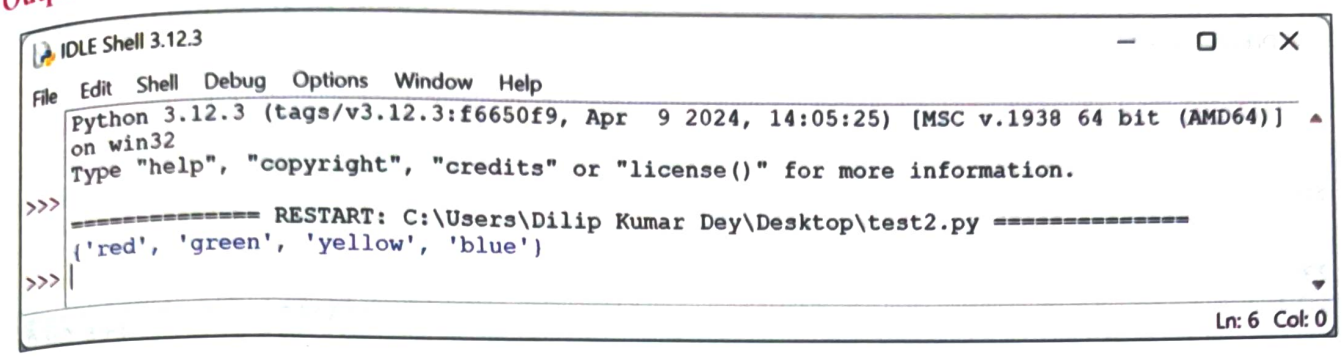
- **Unordered:** The elements in a set do not have a specific order. Therefore, the set items can appear in a different order every time you use them and the elements cannot be referred to an index or a key.
- **Unique Elements:** A set automatically removes duplicate elements.
- **Mutable:** You can add or remove elements from a set after its creation.

In Python, we create sets by placing all the elements inside curly braces { }, separated by commas.

For example,

```
My_set = {"red", "blue", "green", "yellow"}
print(My_set)
```

Output:



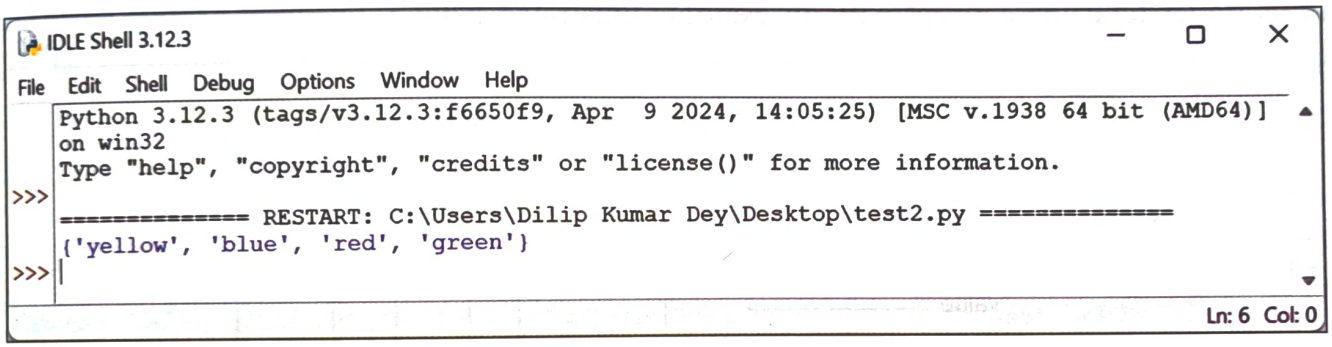
A set is unordered. It means that the items will appear in a random order.

Let us understand the concept of set with some examples.

Example 1: Sets do not allow duplicates, means it cannot have two items with the same value.

```
My_set = {"red", "blue", "green", "yellow", "blue", "green",}
print(My_set)
```

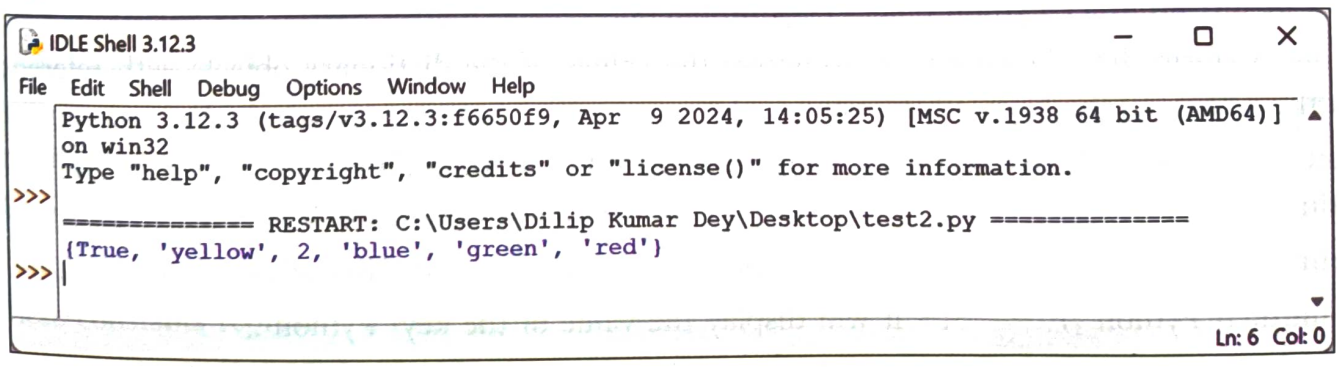
Output:



Example 2: The values True and 1 are considered the same value in sets, and are treated as duplicates.

```
My_set = {"red", "blue", "green", "yellow", True, 1, 2}
print(My_set)
```

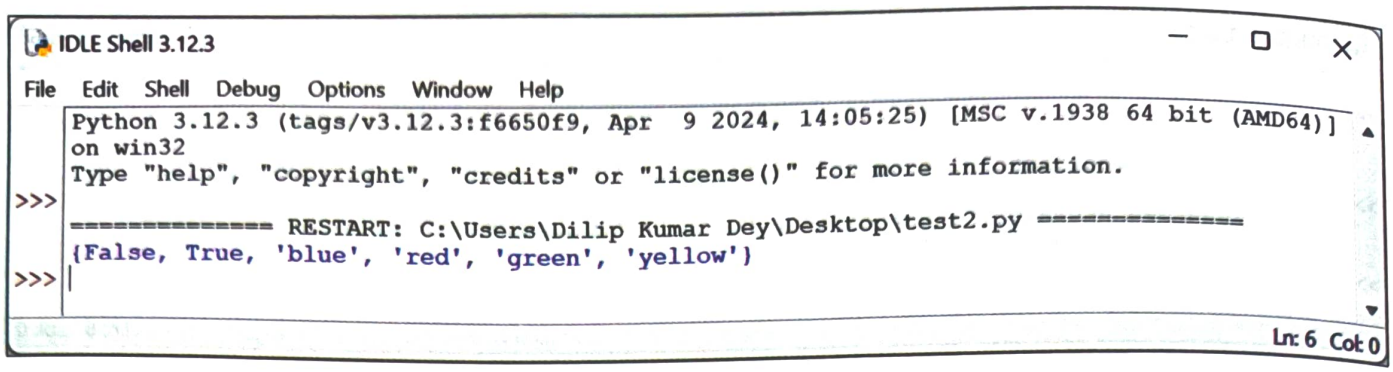
Output:



Example 3: The values False and 0 are considered the same value in sets, and are treated as duplicates.

```
My_set = {"red", "blue", "green", "yellow", True, False, 0, 1}
print(My_set)
```

Output:



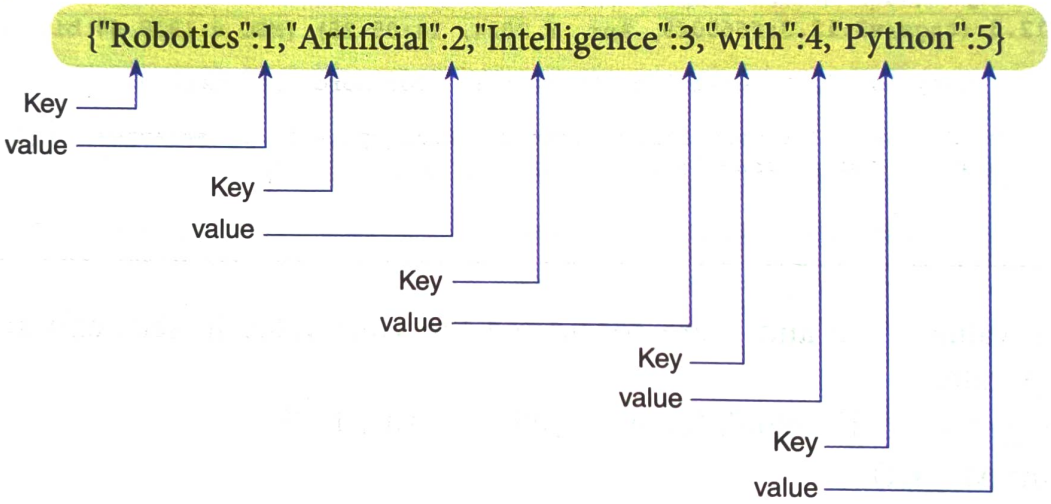
Note: Set items are unchangeable, but you can remove items and add new items.

Dictionary

Unlike list and tuple, a dictionary is an unordered set of elements. The collection of data in a dictionary is referred to as, keys along with their values. This means every key has its corresponding key value which forms a pair in the form of *Key: Value*. Finally, the collection of such pairs is enclosed within curly brackets.

Let us understand the concept with an example illustrated below:

For example,



Generally, the dictionary keys are referred to as strings whereas, the values on the other hand, can be any arbitrary Python object (say, numbers).

Now, perform the following tasks to access the values of the dictionary objects with reference to their keys, all the keys, all the values, etc.

```
dict = {"Robotics":1,"Artificial":2,"Intelligence":3,"with":4,"Python":5}
```

print(dict) : It will display the complete dictionary.

print(dict["Robotics"]) : It will display the value of the key "Robotics".

print(dict["Python"]) : It will display the value of the key "Python".

print(dict.keys( )) : It will display all the keys of the dictionary.

print(dict.values( )) : It will display all the values of the dictionary.

The above program is as under:

test2.py - C:\Users\Dilip Kumar Dey\Desktop\test2.py (3.12.3)

File Edit Format Run Options Window Help

```
dict = {"Robotics":1,"Artificial":2,"Intelligence":3,"with":4,"Python":5}
print(dict)
print(dict["Robotics"])
print(dict["Python"])
print(dict.keys( ))
print(dict.values( ))
```

25 Ln: 7 Col: 0

Output:

IDLE Shell 3.12.3

File Edit Shell Debug Options Window Help

```
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> ========== RESTART: C:\Users\Dilip Kumar Dey\Desktop\test2.py ==========
{'Robotics': 1, 'Artificial': 2, 'Intelligence': 3, 'with': 4, 'Python': 5}
1
5
dict_keys(['Robotics', 'Artificial', 'Intelligence', 'with', 'Python'])
dict_values([1, 2, 3, 4, 5])
>>>
```

Ln: 10 Col: 0

Constants and Variables

Constants and variables are the two basic elements which are frequently used in Python programming. Let us understand them in details.

Constants

When you write a code, you may come across some quantities which remain fixed (i.e., do not change) throughout the execution of the program. Such quantities are termed as Constants. These constants are further categorised as:

- **Integer Constants**

The numbers which are represented without decimal points are called integer constants. They are the whole numbers having positive or negative values.

For example: 14, 345, 8, 6392, –18, –391, etc.

- **Floating Point Constants**

These constants represent numbers with decimal points.

For example: 24.6, 0.0072, –3.652, –51.000025, etc.

- **String Constants**

The letters (uppercase or lowercase), digits, special symbols, word, texts, etc. enclosed within single quotes, double quotes or triple quotes, are referred to as string literals.

For example: 'A', 'a', '10' 'V2' '@', "Computer", "Science", "Python", "Year 2024", "10% per annum", etc.

Note: String constants do not perform any mathematical operations.

• Boolean Constants

Boolean constants are special literals. They represent only two Boolean values, *i.e.*, *True* or *False* and can be used in a Python code to check whether a given logical condition is satisfied or not.

You must note that Boolean constants (*i.e.*, True or False) are never enclosed within quotes. This characteristic makes Boolean constants different from string constants.

Variables

The variable is a named memory location which contains a value. It means when you create a variable, it reserves some space in the memory. The value of a variable can change depending upon the requirements in a program.

For example, n = 25

It means that the integer value 25 is assigned to the variable 'n'. The data value 25 will be stored in the memory location which is identified by the name 'n'.

Now, whenever we refer to 'n', we will get 25. However, the value of 'n' can be changed any time. So, the name 'n' is called a variable.

```
>>> n = 25
>>> print(n)
25
>>> n = 45
>>> print(n)
45
```

n ⟶ 25

n ⟶ 45

When a variable is reassigned with a new value, the original value will get replaced by the new value.

Rules for naming a variable

- A variable may have any number of characters.
- A variable name cannot start with a number, but it can start with characters or the underscore (For example, _rollnum).
- The underscore can be used in between the characters to separate the words of a variable name (For example, reg_no).
- The variable names should be meaningful, which easily explains its purpose.
- Python is case sensitive. So, it treats uppercase and lowercase letters differently.

Assignments

Assigning means to store values in variables using a token '='. Here, the symbol '=' acts as an assignment operator. The 'equal to' sign (=) is used to assign values to the variables. The operand to the left of '=' operator is the variable name and the operand to the right of '=' operator is the value to be stored.

Syntax: <variable> = <value>

For example:

```
m = 15                      : 15 is stored in the variable m which is of integer type.
n = 45.24                   : 45.24 is stored in the variable n which is of float type.
ch = 'k'                    : The letter 'k' is stored as a string character in a variable ch.
wd = "Computer Science"     : A word is stored in the variable wd.
```

Multiple Assignment

Python language also facilitates assigning a single value to several variables simultaneously. The values may be of mixed data types viz. integer, floating point numbers, string, etc.

- Assigning same integer value to different variables

  For example,

  a= b = c = 15

  Here, an integer value 15 is assigned to three different variables and is stored in the memory locations assigned to the variables a, b and c.

- Assigning multiple objects to multiple variables.

  For example,

  m, n, wd = 146, 2020, "Computer"

  Here, two integers with values 146 and 2020 are assigned to the variables m and n respectively and one string with the value "Computer" is assigned to the variable wd.



Chapter at a Glance

- The basic structure of a Python program consists of the components such as comments, import statements, variables, data types, operators, control structures, etc.
- Comments are non-executable statements. They are used to explain the purpose of the code or to make notes for other programmers.
- The import statements are used to include modules or libraries into the program to accomplish tasks.
- Variables are used to store data in memory to perform various arithmetical tasks and also for later use.
- Python supports several built-in data types including integers, floats, strings, boolean, lists, tuple, etc. to perform various tasks during programming.
- Operators are the symbols or tokens to perform various mathematical or logical operations on data.
- Indentation is the fundamental aspect of Python programming that defines the structure and flow of code.
- Python data types are referred to as the categorisation of data items used in coding. It includes numeric, non-numeric, sequences and dictionary.
- In Python, a string is defined as a collection of one or more characters enclosed within single, double or triple quotes.
- All whole numbers (positive or negative numbers) without a decimal point are said to be *int* type numbers.
- All fractional numbers are said to be *float* type numbers. They are also called floating point numbers.
- Complex number is the combination of a real number and an imaginary number.
- When the data type gets converted into another data type after the user's intervention, the type conversion is known as *explicit type conversion or type casting.*
- Type casting is applicable in converting the data type from a higher data type to a lower data type or vice-versa.
- In Python, the Boolean constants are represented as True or False.
- A list is defined as the ordered sequence of one or more data items of the same or different data types.
- A tuple is also an ordered set of elements containing one or more data items of the same or different types.
- In Python, a set is a built-in data type that represents an unordered collection of unique elements.
- Unlike list and tuple, a dictionary is an unordered set of elements. The collection of data in a dictionary is referred to as keys along with their values.