**SECTION1:INTRODUCTION**

**CHAPTER1**

How to Get Started With Python?

**In this tutorial, you will learn to install and run Python on your computer. Once we do that, we will also write our first Python program.**

### Table of Contents

* [The Easiest Way to Run Python](https://www.programiz.com/python-programming/first-program#run-thonny)
* [Install Python](https://www.programiz.com/python-programming/first-program#install-python)
  + [Run Python in Immediate Mode](https://www.programiz.com/python-programming/first-program#immediate-mode)
  + [Run Python in IDE](https://www.programiz.com/python-programming/first-program#ide)
* [Your First Python Program](https://www.programiz.com/python-programming/first-program#first-program)

Python is a cross-platform programming language, meaning, it runs on multiple platforms like Windows, MacOS, Linux and has even been ported to the Java and .NET virtual machines. It is free and open source.

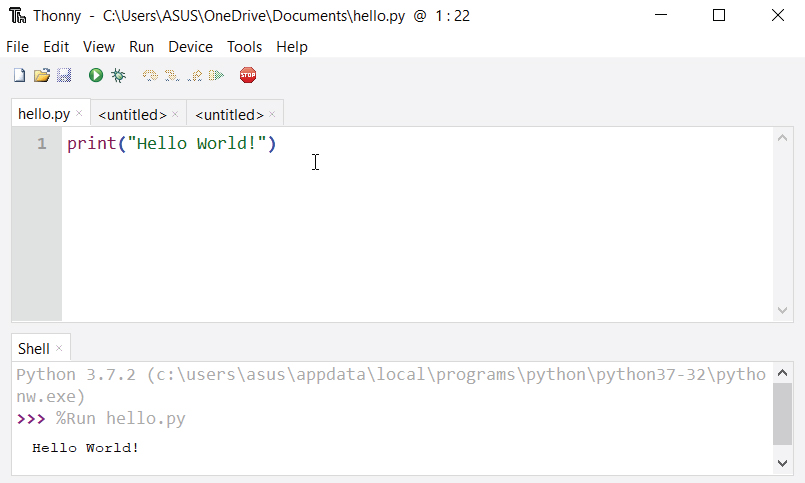
Even though most of today’s Linux and Mac have Python preinstalled in it, the version might be out-of-date. So, it is always a good idea to install the most current version.

## The Easiest Way to Run Python

The easiest way to run Python is by using **Thonny IDE**.

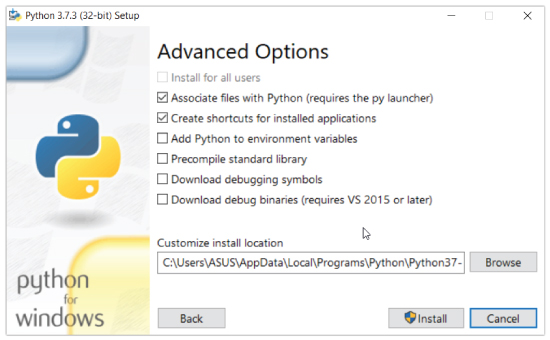
The Thonny IDE comes with the latest version of Python bundled in it. So you don't have to install Python separately.

Follow the following steps to run Python on your computer.

1. Download [Thonny IDE](https://thonny.org/" \o "Download Thonny IDE).
2. Run the installer to install Thonny on your computer.
3. Go to **File** > **New**. Then save the file with .py extension. For example, hello.py, example.py etc.  
   You can give any name to the file. However, the file name should end with **.py**
4. Write Python code in the file and save it.  
   
5. Then Go to **Run** > **Run current script** or simply click **F5** to run it.

## Install Python Separately

If you don't want to use Thonny, here's how you can install and run Python on your computer.

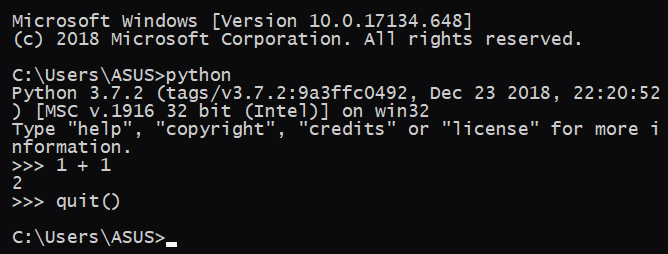
1. Download the [latest version of Python](https://www.python.org/downloads/).
2. Run the installer file and follow the steps to install Python  
   During the install process, check **Add Python to environment variables**. This will add Python to environment variables and you are able to run Python from any part of the computer.  
   Also, you can choose the path where Python is installed.  
   

Once you finish the installation process, you can run Python.

### 1. Run Python in Immediate mode

Once Python is installed, typing python in the command line will invoke the interpreter in immediate mode. We can directly type in Python code and press enter to get the output.

Try typing in 1 + 1 and press enter. We get 2 as the output. This prompt can be used as a calculator. To exit this mode type quit() and press enter.



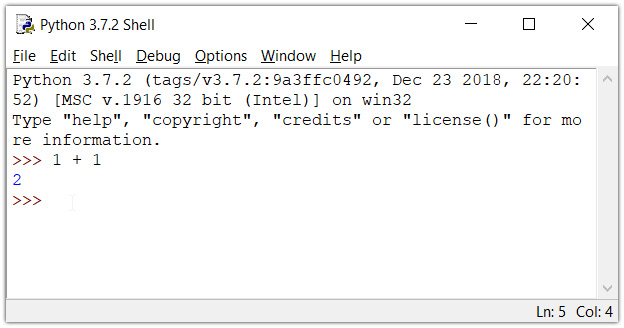
### 2. Run Python in the Integrated Development Environment (IDE)

We can use any text editing software to write a Python script file.

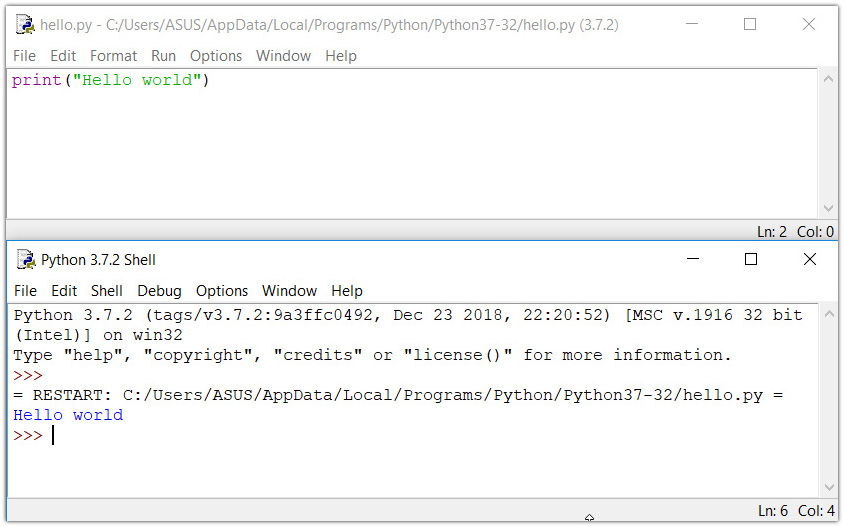
We just need to save it with the .py extension. But using an IDE can make our life a lot easier. IDE is a piece of software that provides useful features like code hinting, syntax highlighting and checking, file explorers etc. to the programmer for application development.

By the way, when you install Python, an IDE named **IDLE** is also installed. You can use it to run Python on your computer. It's a decent IDE for beginners.

When you open IDLE, an interactive Python Shell is opened.

  
Now you can create a new file and save it with **.py** extension. For example, **hello.py**

Write Python code in the file, save it. To run the file go to **Run** > **Run Module** or simply click **F5**.



## Your first Python Program

Now that we have Python up and running, we can write our first Python program.

Let's create a very simple program called "Hello World!".  A "Hello, World!" is a simple program that outputs Hello, World! on the screen. Since it's a very simple program, it's often used to introduce a new programming language to a newbie.

Type the following code in any text editor or an IDE and save it as helloWorld.py

print("Hello world!")

Then, run the file. You will get the following output.

Hello world!

Congratulations! You just wrote your first program in Python.

As we can see, it was pretty easy. This is the beauty of Python programming language.

CHAPTER2

# Python Keywords and Identifiers

**In this tutorial, you will learn about keywords (reserved words in Python) and identifiers (names given to variables, functions, etc.).**

## Python Keywords

Keywords are the reserved words in Python.

We cannot use a keyword as a [variable name](https://www.programiz.com/python-programming/variables-datatypes), [function](https://www.programiz.com/python-programming/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 3.7. This number can vary slightly in the course of time.

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 is 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 |  |
| Keywords in Python | | | | |

Looking at all the keywords at once and trying to figure out what they mean might be overwhelming.

If you want to have an overview, here is the complete [list of all the keywords](https://www.programiz.com/python-programming/keyword-list) with examples.

## Python Identifiers

An identifier is a name given to entities like class, functions, variables, etc. It helps to differentiate one entity from another.

### Rules for writing identifiers

1. 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 myClass, var\_1 and print\_this\_to\_screen, all are valid example.
2. An identifier cannot start with a digit. 1variable is invalid, but variable1 is perfectly fine.
3. Keywords cannot be used as identifiers.
   1. >>> global = 1
   2. File "<interactive input>", line 1
   3. global = 1
   4. ^
   5. SyntaxError: invalid syntax
4. We cannot use special symbols like **!**, **@**, **#**, **$**, **%** etc. in our identifier.
   1. >>> a@ = 0
   2. File "<interactive input>", line 1
   3. a@ = 0
   4. ^
   5. SyntaxError: invalid syntax
5. Identifier can be of any length.

### Things to Remember

Python is a case-sensitive language. This means, Variable and variable are not the same. Always name identifiers that make sense.

While, c = 10 is valid. Writing count = 10 would make more sense and it would be easier to figure out what it does even when you look at your code after a long gap.

Multiple words can be separated using an underscore, this\_is\_a\_long\_variable.

**CHAPTER3**

Python Statement, Indentation and Comments

**In this article, you will learn about Python statements, why indentation is important and use of comments in programming.**

## Python Statement

Instructions that a Python interpreter can execute are called statements. For example, a = 1is an assignment statement. if statement, for statement, while statement etc. are other kinds of statements which will be discussed later.

### Multi-line statement

In Python, end of a statement is marked by a newline character. But we can make a statement extend over multiple lines with the line continuation character (\). For example:

1. a = 1 + 2 + 3 + \
2. 4 + 5 + 6 + \
3. 7 + 8 + 9

This is explicit line continuation. In Python, line continuation is implied inside parentheses ( ), brackets [ ] and braces { }. For instance, we can implement the above multi-line statement as

1. a = (1 + 2 + 3 +
2. 4 + 5 + 6 +
3. 7 + 8 + 9)

Here, the surrounding parentheses ( ) do the line continuation implicitly. Same is the case with [ ] and { }. For example:

1. colors = ['red',
2. 'blue',
3. 'green']

We could also put multiple statements in a single line using semicolons, as follows

1. a = 1; b = 2; c = 3

## Python Indentation

Most of the programming languages like C, C++, Java use braces { } to define a block of code. Python uses indentation.

A code block (body of a [function](https://www.programiz.com/python-programming/function), [loop](https://www.programiz.com/python-programming/for-loop) etc.) starts with indentation and ends with the first unindented line. The amount of indentation is up to you, but it must be consistent throughout that block.

Generally four whitespaces are used for indentation and is preferred over tabs. Here is an example.

for i in range(1,11):

print(i)

if i == 5:

break

The enforcement of indentation in Python makes the code look neat and clean. This results into Python programs that look similar and consistent.

Indentation can be ignored in line continuation. But it's a good idea to always indent. It makes the code more readable. For example:

1. if True:
2. print('Hello')
3. a = 5

and

1. if True: print('Hello'); a = 5

both are valid and do the same thing. But the former style is clearer.

Incorrect indentation will result into IndentationError.

Comments are very important while writing a program. It describes what's going on inside a program so that a person looking at the source code does not have a hard time figuring it out. You might forget the key details of the program you just wrote in a month's time. So taking time to explain these concepts in form of comments is always fruitful.

In Python, we use the hash (#) symbol to start writing a comment.

It extends up to the newline character. Comments are for programmers for better understanding of a program. Python Interpreter ignores comment.

1. #This is a comment
2. #print out Hello
3. print('Hello')

### Multi-line comments

If we have comments that extend multiple lines, one way of doing it is to use hash (#) in the beginning of each line. For example:

1. #This is a long comment
2. #and it extends
3. #to multiple lines

Another way of doing this is to use triple quotes, either ''' or """.

These triple quotes are generally used for multi-line strings. But they can be used as multi-line comment as well. Unless they are not docstrings, they do not generate any extra code.

1. """This is also a
2. perfect example of
3. multi-line comments"""

### Docstring in Python

Docstring is short for documentation string.

It is a [string](https://www.programiz.com/python-programming/string) that occurs as the first statement in a module, function, class, or method definition. We must write what a function/class does in the docstring.

Triple quotes are used while writing docstrings. For example:

def double(num):

"""Function to double the value"""

return 2\*num

Docstring is available to us as the attribute \_\_doc\_\_ of the function. Issue the following code in shell once you run the above program.

1. >>> print(double.\_\_doc\_\_)
2. Function to double the value

CHAPTER4

# Python Variables, Constants and Literals

**In this article, you will learn about Python variables, constants, literals and their use cases.**

## Python Variables

A variable is a named location used to store data in the memory. It is helpful to think of variables as a container that holds data which can be changed later throughout programming. For example,

1. number = 10

Here, we have created a named number. We have assigned value 10 to the variable.

You can think variable as a bag to store books in it and those books can be replaced at any time.

1. number = 10
2. number = 1.1

Initially, the value of number was 10. Later it's changed to 1.1.

Note: In Python, we don't assign values to the variables, whereas Python gives the reference of the object (value) to the variable.

### Assigning a value to a Variable in Python

As you can see from the above example, you can use the assignment operator = to assign a value to a variable.

**Example 1: Declaring and assigning a value to a variable**

website = "apple.com"

print(website)

When you run the program, the output will be:

apple.com

n the above program, we assigned a value apple.com to the variable website. Then we print the value assigned to website i.e. apple.com

Note : Python is a [type inferred](https://en.wikipedia.org/wiki/Type_inference) language; it can automatically know apple.com is a string and declare website as a string.

**Example 2: Changing the value of a variable**

website = "apple.com"

print(website)

# assigning a new variable to website

website = "programiz.com"

print(website)

When you run the program, the output will be:

apple.com

programiz.com

In the above program, we have assigned apple.com to the website variable initially. Then, it's value is changed to programiz.com.

a, b, c = 5, 3.2, "Hello"

print (a)

print (b)

print (c)

If we want to assign the same value to multiple variables at once, we can do this as

x = y = z = "same"

print (x)

print (y)

print (z)

The second program assigns the same string to all the three variables x, y and z.

## Constants

A constant is a type of variable whose value cannot be changed. It is helpful to think of constants as containers that hold information which cannot be changed later.

Non technically, you can think of constant as a bag to store some books and those books cannot be replaced once placed inside the bag.

### Assigning value to a constant in Python

In Python, constants are usually declared and assigned on a module. Here, the module means a new file containing variables, functions etc which is imported to main file. Inside the module, constants are written in all capital letters and underscores separating the words.

#### Example 3: Declaring and assigning value to a constant

Create a constant.py

1. PI = 3.14
2. GRAVITY = 9.8

Create a main.py

1. import constant
2. print(constant.PI)
3. print(constant.GRAVITY)
4. When you run the program, the output will be:
5. 3.14
6. 9.8

In the above program, we create a constant.py module file. Then, we assign the constant value to PI and GRAVITY. After that, we create a main.py file and import the constant module. Finally, we print the constant value.

Note: In reality, we don't use constants in Python. The globals or constants module is used throughout the Python programs.

## Rules and Naming convention for variables and constants

1. Create a name that makes sense. Suppose, vowel makes more sense than v.
2. Use camelCase notation to declare a variable. It starts with lowercase letter. For example:
3. myName
4. myAge

myAddress

1. Use capital letters where possible to declare a constant. For example:
2. PI
3. G
4. MASS

TEMP

1. Never use special symbols like !, @, #, $, %, etc.
2. Don't start name with a digit.
3. Constants are put into Python modules and meant not be changed.
4. Constant and variable names should have combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (\_). For example:
5. snake\_case
6. MACRO\_CASE
7. camelCase

CapWords

## Literals

Literal is a raw data given in a variable or constant. In Python, there are various types of literals they are as follows:

### Numeric Literals

Numeric Literals are immutable (unchangeable). Numeric literals can belong to 3 different numerical types Integer, Float and Complex.

#### Example 4: How to use Numeric literals in Python?

a = 0b1010 #Binary Literals

b = 100 #Decimal Literal

c = 0o310 #Octal Literal

d = 0x12c #Hexadecimal Literal

#Float Literal

float\_1 = 10.5

float\_2 = 1.5e2

#Complex Literal

x = 3.14j

print(a, b, c, d)

print(float\_1, float\_2)

print(x, x.imag, x.real)

When you run the program, the output will be:

10 100 200 300

10.5 150.0

3.14j 3.14 0.0

In the above program,

* We assigned integer literals into different variables. Here, a is binary literal, b is a decimal literal, c is an octal literal and d is a hexadecimal literal.
* When we print the variables, all the literals are converted into decimal values.
* 10.5 and 1.5e2 are floating point literals. 1.5e2 is expressed with exponential and is equivalent to 1.5 \* 102.
* We assigned a complex literal i.e 3.14j in variable x. Then we use imaginary literal (x.imag) and real literal (x.real) to create imaginary and real part of complex number.

To learn more about Numeric Literals, refer [Python Numbers](https://www.programiz.com/python-programming/numbers).

### String literals

A string literal is a sequence of characters surrounded by quotes. We can use both single, double or triple quotes for a string. And, a character literal is a single character surrounded by single or double quotes.

#### Example 7: How to use string literals in Python?

strings = "This is Python"

char = "C"

multiline\_str = """This is a multiline string with more than one line code."""

unicode = u"\u00dcnic\u00f6de"

raw\_str = r"raw \n string"

print(strings)

print(char)

print(multiline\_str)

print(unicode)

print(raw\_str)

When you run the program, the output will be:

This is Python

C

This is a multiline string with more than one line code.

Ünicöde

raw \n string

When you run the program, the output will be:

This is Python

C

This is a multiline string with more than one line code.

Ünicöde

raw \n string

In the above program, This is Python is a string literal and C is a character literal. The value with triple-quote """ assigned in the multiline\_str is multi-line string literal. The u"\u00dcnic\u00f6de" is a unicode literal which supports characters other than English and r"raw \n string" is a raw string literal

### Boolean literals

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

#### Example 8: How to use boolean literals in Python?

x = (1 == True)

y = (1 == False)

a = True + 4

b = False + 10

print("x is", x)

print("y is", y)

print("a:", a)

print("b:", b)

When you run the program, the output will be:

x is True

y is False

a: 5

b: 10

n the above program, we use boolean literal True and False. In Python, True represents the value as 1 and False as 0. The value of x is True because 1 is equal to True. And, the value of y is False because 1 is not equal to False.

Similarly, we can use the True and False in numeric expressions as the value. The value of a is 5 because we add True which has value of 1 with 4. Similarly, b is 10 because we add the False having value of 0 with 10.

### Special literals

Python contains one special literal i.e. None. We use it to specify to that field that is not created.

#### Example 9: How to use special literals in Python?

drink = "Available"

food = None

def menu(x):

if x == drink:

print(drink)

else:

print(food)

menu(drink)

menu(food)

When you run the program, the output will be:

Available

None

In the above program, we define a menu function. Inside menu, when we set parameter as drink then, it displays Available. And, when the parameter is food, it displays None.

### Literal Collections

There are four different literal collections List literals, Tuple literals, Dict literals, and Set literals.

#### Example 10: How to use literals collections in Python?

fruits = ["apple", "mango", "orange"] #list

numbers = (1, 2, 3) #tuple

alphabets = {'a':'apple', 'b':'ball', 'c':'cat'} #dictionary

vowels = {'a', 'e', 'i' , 'o', 'u'} #set

print(fruits)

print(numbers)

print(alphabets)

print(vowels)

When you run the program, the output will be:

['apple', 'mango', 'orange']

(1, 2, 3)

{'a': 'apple', 'b': 'ball', 'c': 'cat'}

{'e', 'a', 'o', 'i', 'u'}

In the above program, we created a list of fruits, tuple of numbers, dictionary dict having values with keys desginated to each value and set of vowels.

To learn more about literal collections, refer [Python Data Types](https://www.programiz.com/python-programming/variables-datatypes).

CHAPTER 5

Python Data Types

**In this tutorial, you will learn about different data types you can use in Python.**

## Data types in Python

Every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.

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

### Python Numbers

Integers, floating point numbers and complex numbers falls under [Python numbers](https://www.programiz.com/python-programming/numbers) category. They are defined as int, float and complex class in Python.

We can use the type() function to know which class a variable or a value belongs to and the isinstance() function to check if an object belongs to a particular class.

a = 5

print(a, "is of type", type(a))

a = 2.0

print(a, "is of type", type(a))

a = 1+2j

print(a, "is complex number?", isinstance(1+2j,complex))

A floating point number is accurate up to 15 decimal places. Integer and floating points are separated by decimal points. 1 is integer, 1.0 is floating point number.

Complex numbers are written in the form, x + yj, where x is the real part and y is the imaginary part. Here are some examples.

1. >>> a = 1234567890123456789
2. >>> a
3. 1234567890123456789
4. >>> b = 0.1234567890123456789
5. >>> b
6. 0.12345678901234568
7. >>> c = 1+2j
8. >>> c
9. (1+2j)

Notice that the float variable b got truncated.

### Python List

[List](https://www.programiz.com/python-programming/list) is an ordered sequence of items. It is one of the most used datatype 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 [ ].

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

We can use the slicing operator [ ] to extract an item or a range of items from a list. Index starts form 0 in Python.

a = [5,10,15,20,25,30,35,40]

# a[2] = 15

print("a[2] = ", a[2])

# a[0:3] = [5, 10, 15]

print("a[0:3] = ", a[0:3])

# a[5:] = [30, 35, 40]

print("a[5:] = ", a[5:])

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

1. >>> a = [1,2,3]
2. >>> a[2]=4
3. >>> a
4. [1, 2, 4]

### Python Tuple

[Tuple](https://www.programiz.com/python-programming/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.

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

We can use the slicing operator [] to extract items but we cannot change its value.

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

# t[1] = 'program'

print("t[1] = ", t[1])

# t[0:3] = (5, 'program', (1+3j))

print("t[0:3] = ", t[0:3])

# Generates error

# Tuples are immutable

t[0] = 10

### Python Strings

[String](https://www.programiz.com/python-programming/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 """.

1. >>> s = "This is a string"
2. >>> s = '''a multiline

Like list and tuple, slicing operator [ ] can be used with string. Strings are immutable.

s = 'Hello world!'

# s[4] = 'o'

print("s[4] = ", s[4])

# s[6:11] = 'world'

print("s[6:11] = ", s[6:11])

# Generates error

# Strings are immutable in Python

s[5] ='d'

### Python Set

[Set](https://www.programiz.com/python-programming/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.

a = {5,2,3,1,4}

# printing set variable

print("a = ", a)

# data type of variable a

print(type(a))

We can perform set operations like union, intersection on two sets. Set have unique values. They eliminate duplicates.

1. >>> a = {1,2,2,3,3,3}
2. >>> a
3. {1, 2, 3}

Since, set are unordered collection, indexing has no meaning. Hence the slicing operator [] does not work.

1. >>> a = {1,2,3}
2. >>> a[1]
3. Traceback (most recent call last):
4. File "<string>", line 301, in runcode
5. File "<interactive input>", line 1, in <module>
6. TypeError: 'set' object does not support indexing

### Python Dictionary

[Dictionary](https://www.programiz.com/python-programming/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.

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

d = {1:'value','key':2}

print(type(d))

print("d[1] = ", d[1]);

print("d['key'] = ", d['key']);

# Generates error

print("d[2] = ", d[2]);

### Conversion between data types

We can convert between different data types by using different type conversion functions like int(), float(), str() etc.

1. >>> float(5)
2. 5.0

Conversion from float to int will truncate the value (make it closer to zero).

1. >>> int(10.6)
2. 10
3. >>> int(-10.6)
4. -10

Conversion to and from string must contain compatible values.

1. >>> float('2.5')
2. 2.5
3. >>> str(25)
4. '25'
5. >>> int('1p')
6. Traceback (most recent call last):
7. File "<string>", line 301, in runcode
8. File "<interactive input>", line 1, in <module>
9. ValueError: invalid literal for int() with base 10: '1p'

We can even convert one sequence to another.

1. >>> set([1,2,3])
2. {1, 2, 3}
3. >>> tuple({5,6,7})
4. (5, 6, 7)
5. >>> list('hello')
6. ['h', 'e', 'l', 'l', 'o']

To convert to dictionary, each element must be a pair

1. >>> dict([[1,2],[3,4]])
2. {1: 2, 3: 4}
3. >>> dict([(3,26),(4,44)])
4. {3: 26, 4: 44}

**CHAPTER 6**

Python Type Conversion and Type Casting

**In this article you will learn about the Type conversion and uses of type conversion.**

## Type Conversion:

The process of converting the value of one data type (integer, string, float, etc.) to another data type is called type conversion. Python has two types of type conversion.

1. Implicit Type Conversion
2. Explicit Type Conversion

## Implicit Type Conversion:

In Implicit type conversion, Python automatically converts one data type to another data type. This process doesn't need any user involvement.

Let's see an example where Python promotes conversion of lower datatype (integer) to higher data type (float) to avoid data loss.

### Example 1: Converting integer to float

**num\_int = 123**

**num\_flo = 1.23**

**num\_new = num\_int + num\_flo**

**print("datatype of num\_int:",type(num\_int))**

**print("datatype of num\_flo:",type(num\_flo))**

**print("Value of num\_new:",num\_new)**

**print("datatype of num\_new:",type(num\_new))**

When we run the above program, the output will be

datatype of num\_int: <class 'int'>

datatype of num\_flo: <class 'float'>

Value of num\_new: 124.23

datatype of num\_new: <class 'float'>

In the above program,

* We add two variables num\_int and num\_flo, storing the value in num\_new.
* We will look at the data type of all three objects respectively.
* In the output we can see the datatype of num\_int is an integer, datatype of num\_flo is a float.
* Also, we can see the num\_new has float data type because Python always converts smaller data type to larger data type to avoid the loss of data.
* Now, let's try adding a string and an integer, and see how Python treats it.

### Example 2: Addition of string(higher) data type and integer(lower) datatype

**num\_int = 123**

**num\_str = "456"**

**print("Data type of num\_int:",type(num\_int))**

**print("Data type of num\_str:",type(num\_str))**

**print(num\_int+num\_str)**

When we run the above program, the output will be

Data type of num\_int: <class 'int'>

Data type of num\_str: <class 'str'>

Traceback (most recent call last):

File "python", line 7, in <module>

TypeError: unsupported operand type(s) for +: 'int' and 'str'

n the above program,

* We add two variable num\_int and num\_str.
* As we can see from the output, we got typeerror. Python is not able use Implicit Conversion in such condition.
* However Python has the solution for this type of situation which is know as Explicit Conversion.

## Explicit Type Conversion:

In Explicit Type Conversion, users convert the data type of an object to required data type. We use the predefined functions like int(), float(), str(), etc to perform explicit type conversion.

This type conversion is also called typecasting because the user casts (change) the data type of the objects.

Syntax :

(required\_datatype)(expression)

Typecasting can be done by assigning the required data type function to the expression.

### Example 3: Addition of string and integer using explicit conversion

num\_int = 123

num\_str = "456"

print("Data type of num\_int:",type(num\_int))

print("Data type of num\_str before Type Casting:",type(num\_str))

num\_str = int(num\_str)

print("Data type of num\_str after Type Casting:",type(num\_str))

num\_sum = num\_int + num\_str

print("Sum of num\_int and num\_str:",num\_sum)

print("Data type of the sum:",type(num\_sum))

n above program,

* We add num\_str and num\_int variable.
* We converted num\_str from string(higher) to integer(lower) type using int() function to perform the addition.
* After converting num\_str to a integer value Python is able to add these two variable.
* We got the num\_sum value and data type to be integer.

## Key Points to Remember:

1. Type Conversion is the conversion of object from one data type to another data type.
2. Implicit Type Conversion is automatically performed by the Python interpreter.
3. Python avoids the loss of data in Implicit Type Conversion.
4. Explicit Type Conversion is also called Type Casting, the data types of object are converted using predefined function by user.
5. In Type Casting loss of data may occur as we enforce the object to specific data type.

**CHAPTER 7**

Python Input, Output and Import

**This tutorial focuses on two built-in functions print() and input() to perform I/O task in Python. Also, you will learn to import modules and use them in your program.**

Python provides numerous [built-in functions](https://www.programiz.com/python-programming/built-in-function) that are readily available to us at the Python prompt.

Some of the functions like input() and print() are widely used for standard input and output operations respectively. Let us see the output section first.

## Python Output Using print() function

We use the print() function to output data to the standard output device (screen).

We can also [output data to a file](https://www.programiz.com/python-programming/file-operation), but this will be discussed later. An example use is given below.

print('This sentence is output to the screen')

# Output: This sentence is output to the screen

a = 5

print('The value of a is', a)

# Output: The value of a is 5

In the second print() statement, we can notice that a space was added between the [string](https://www.programiz.com/python-programming/string" \o "Python Strings)and the value of variable a.This is by default, but we can change it.

The actual syntax of the print() function is

print(\*objects, sep=' ', end='\n', file=sys.stdout, flush=False)

Here, objects is the value(s) to be printed.

The sep separator is used between the values. It defaults into a space character.

After all values are printed, end is printed. It defaults into a new line.

The file is the object where the values are printed and its default value is sys.stdout(screen). Here are an example to illustrate this.

print(1,2,3,4)

# Output: 1 2 3 4

print(1,2,3,4,sep='\*')

# Output: 1\*2\*3\*4

print(1,2,3,4,sep='#',end='&')

# Output: 1#2#3#4&

### Output formatting

Sometimes we would like to format our output to make it look attractive. This can be done by using the str.format() method. This method is visible to any string object.

1. >>> x = 5; y = 10
2. >>> print('The value of x is {} and y is {}'.format(x,y))
3. The value of x is 5 and y is 10

Here the curly braces {} are used as placeholders. We can specify the order in which it is printed by using numbers (tuple index).

# Output: I love bread and butter

print('I love {1} and {0}'.format('bread','butter'))

# Output: I love butter and bread