**Files I/O**

# Printing to the Screen

print() function converts the expressions you pass into a string and writes the result to standard output

#!/usr/bin/python3

print ("Python is really a great language,", "isn't it?")

Python is really a great language, isn't it?

# Reading Keyboard Input

To read data from standard input:

|  |  |  |
| --- | --- | --- |
| Functions | Python2 | Python3 |
| raw\_input() | ✓ | 🗶 |
| input() | ✓ | ✓ |

# input() function

Read data from keyboard as string, irrespective of whether it is enclosed with quotes ('' or "" ) or not.

The input([prompt]) function is equivalent to raw\_input, except that it assumes that the input is a valid Python expression and returns the evaluated result to you.

# Opening and Closing Files

## open() function

Creates a file object, which would be utilized to call other support methods associated with it.

**Syntax**

file object = open(file\_name [, access\_mode][, buffering])

or

**file\_name:**

* path name of the file

**access\_mode:**

* determines the mode in which the file has to be opened, i.e., read, write, append, etc.
* optional parameter and the default file access mode is read (r).

**buffering:**

* If set to 0, no buffering takes place.
* If the buffering value is 1, line buffering is performed while accessing a file.
* If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size.
* If negative, the buffer size is the system default(default behavior).

## Different modes

### For text files

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Mode** | **Description** |
| 1 | **r** | * Opens a file for reading only. * File pointer is placed at the beginning of the file. * **Default mode.** |
| 2 | **r+** | * Opens a file for both reading and writing. * The file pointer placed at the beginning of the file. |
| 3 | **w** | * Opens a file for writing only. * Overwrites the file if the file exists. * If the file does not exist, creates a new file for writing. |
| 4 | **w+** | * Opens a file for both writing and reading. * Overwrites the existing file if the file exists. * If the file does not exist, creates a new file for reading and writing. |
| 5 | **a** | * Opens a file for appending. * The file pointer is at the end of the file if the file exists. * If the file does not exist, it creates a new file for writing. |
| 6 | **a+** | * Opens a file for both appending and reading. * The file pointer is at the end of the file if the file exists * If the file does not exist, it creates a new file for reading and writing. |

### For binary files

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Mode** | **Description** |
| 1 | **rb** | * Opens a file for reading only in binary format. * The file pointer is placed at the beginning of the file. * **Default mode.** |
| 2 | **rb+** | * Opens a file for both reading and writing in binary format. * The file pointer placed at the beginning of the file. |
| 3 | **wb** | * Opens a file for writing only in binary format. * Overwrites the file if the file exists. * If the file does not exist, creates a new file for writing. |
| 4 | **wb+** | * Opens a file for both writing and reading in binary format. * Overwrites the existing file if the file exists. * If the file does not exist, creates a new file for reading and writing. |
| 5 | **ab** | * Opens a file for appending in binary format. * The file pointer is at the end of the file if the file exists. * If the file does not exist, it creates a new file for writing. |
| 6 | **ab+** | * Opens a file for both appending and reading in binary format. * The file pointer is at the end of the file if the file exists. * If the file does not exist, it creates a new file for reading and writing. |

## File Object Attributes

Once a file is opened and you have one file object, you can get various information related to that file.

List of all the attributes related to a file object:

|  |  |  |
| --- | --- | --- |
| S. No. | Attribute | Description |
| 1 | **file.closed** | Returns true if file is closed, false otherwise. |
| 2 | **file.mode** | Returns access mode with which file was opened. |
| 3 | **file.name** | Returns name of the file. |

## close() Method

The close() method of a file object flushes any unwritten information and closes the file object, after which no more writing can be done.

Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the close() method to close a file.

**Syntax**

fileObject.close();

#!/usr/bin/python3

# Open a file

fo = open("foo.txt", "wb")

print("Name of the file: ", fo.name)

print("Closed or not: ", fo.closed)

print("Opening mode: ", fo.mode)

fo.close()

Output:

Name of the file: foo.txt

Closed or not: False

Opening mode: wb

# Reading and Writing Files

## read() Method

Reads a string from an open file. Python strings can have binary data, apart from text data.

**Syntax**

fileObject.read([count])

**count:** number of bytes to be read from the opened file. (optional argument)

This method starts reading from the beginning of the file and if count is missing, then it tries to read as much as possible, maybe until the end of file.

## write() Method

Writes any string to an open file. Python strings can have binary data and not just text.

The write() method does not add a newline character ('\n') to the end of the string.

**Syntax**

fileObject.write(string)

**string:** content to be written into the opened file

# File Positions

## tell() method

Tells you the current position within the file.

In other words, the next read or write will occur at that many bytes from the beginning of the file.

## seek() method

Changes the current file position.

**Syntax**

seek(offset[, from])

**offset:** indicates the number of bytes to be moved.

**from:** specifies the reference position from where the bytes are to be moved.

|  |  |
| --- | --- |
| from | Reference position |
| 0 | beginning of the file |
| 1 | current position |
| 2 | end of the file |

# Storing Data (json)

The JSON (JavaScript Object Notation) module allows you to dump simple Python data structures into a file and load the data from that file the next time the program runs. You can also use json to share data between different Python programs.

The JSON data format is not specific to Python, so you can share data you store in the JSON format with people who work in many other programming languages.

#!/usr/bin/python3

import json

numbers = [2, 3, 5, 7, 11, 13]

filename = 'numbers.json'

with open(filename, 'w') as f\_obj:

json.dump(numbers, f\_obj)

f\_obj.close()

with open(filename) as f\_obj:

numbers = json.load(f\_obj)

print(numbers)

f\_obj.close()

Output:

[2, 3, 5, 7, 11, 13] (contents of numbers.json file)

[2, 3, 5, 7, 11, 13]

# END