

# Files

* Traditional definition: A computer file is a resource for storing information, which is available to a computer program and is usually based on some kind of durable storage
* In Python, these files can be accessed using the file object discussed in the next few slides
* File objects can be used to access not only normal disk files, but also any other type of "file" that uses that abstraction

# Opening a File

* To open a file use the open() or file() functions
* Syntax:

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

* + **file\_name:** The file\_name argument is a string value that contains the name of the file that you want to access.
  + **access\_mode:** The access\_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc.
  + **buffering:** If the buffering value is 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).

|  |  |
| --- | --- |
| **File Mode** | **Operation** |
| r | Open for read |
| rU or U | Open for read with universal NEWLINE support (PEP 278) |
| w | Open for write (truncate if necessary) |
| a | Open for append (always works from EOF, create if necessary) |
| r+ | Open for read an write |
| w+ | Open for read and write (see w above) |
| a+ | Open for read and write (see a above) |
| rb | Open for binary read |
| wb | Open for binary write (see w above) |
| ab | Open for binary append (see a above) |
| rb+ | Open for binary read and write (see r+ above) |
| wb+ | Open for binary read and write (see w+ above) |
| ab+ | Open for binary read and write (see a+ above) |

# Access Modes

# Closing a File

* 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



•

file

•

Syntax:

**file\_object.close()**

It is a good practice to use the close() method to close a

# Built-in Methods: Input

* The **read()** method is used to read bytes directly into a string, reading at most the number of bytes indicated Syntax: **file\_object.read([size])**
* The **readline()** method reads one line of the open file (reads all bytes until a line-terminating character like NEWLINE is



encountered)

Syntax:

**file\_object.readline([size])**

•

The

**readlines()**

reads all (remaining) lines and returns them as

a list of strings

Syntax:

**file\_object.readlines([sizeint])**

# Built-in Methods: Output

* The write() takes a string that can consist of one or more lines of text data or a block of bytes and writes the data to the file Syntax: **file\_object.write(str)**
* The writelines() takes a string that can consist of one or more lines of text data or a block of bytes and writes the data to the



file

Syntax:

**file\_object.writelines(sequence)**

|  |  |
| --- | --- |
| **File Object Method** | **Operation** |
| file.next() | Returns the next line in the file [similar to file.readline()] or raises StopIteration if no more lines are available |
| file.xreadlines() | Meant for iteration, returns lines in file read as chunks in a more efficient way than readlines() |
| file.seek(off, whence=0) | Moves to a location within file, off bytes offset from whence (0 == beginning of file, 1 == current location, or 2 == end of file) |
| file.tell() | Returns current location within file |
| file.truncate(size=file.tell()) | Truncates file to at most size bytes, the default being the current file location |

# Other Important Functions

|  |  |
| --- | --- |
| **Method** | **Returns** |
| s.lstrip()  s.lstrip(chars) | A new version of s in which white space (blanks, tabs, and newlines) is removed from the left (the front) of s. If provided, characters in the string chars are removed instead of white space. |
| s.rstrip()  s.rstrip(chars) | Same as lstrip except characters are removed from the right (the end) of s. |
| s.strip()  s.strip(chars) | Similar to lstrip and rstrip, except characters are removed from the front and end of s. |

# Character Stripping Methods

# Built-in Attributes

|  |  |
| --- | --- |
| **File Object Attribute** | **Description** |
| file.closed | TRue if file is closed and False otherwise |
| file.encoding | Encoding that this file useswhen Unicode strings are written to file, they will be converted to byte strings using file.encoding; a value of None indicates that the system default encoding for converting Unicode strings should be used |
| file.mode | Access mode with which file was opened |
| file.name | Name of file |
| file.newlines | None if no line separators have been r  ead, a string consisting of  one type of line separator, or a tuple containing a  ll types of line  termination characters read so far |
| file.softspace | 0 if space explicitly required with print, 1 otherwise; rarely used by the programmergenerally for internal use only |

# Standard Files

* There are generally three standard files that are made available to you when your program starts
* Python makes these file handles available to you from the **sys** module
* Once you import sys, you have access to these files as **sys.stdin**, **sys.stdout** and **sys.stderr**
* The print statement normally outputs to **sys.stdout** while the **input()** built-in function receives its input from **sys.stdin**