Python for scientific research

Input, output and the filesystem

Bram Kuijper

University of Exeter, Penryn Campus, UK

February 11, 2020

What we've done so far

1. Declare variables using built-in data types and execute operations on them
2. Use flow control commands to dictate the order in which commands are run and when
3. Encapsulate programs into reusable functions, modules and packages
4. Work with textual data and pattern matching
5. **Next** working with files & the file system

Working with the filesystem: key modules

- Work with *names* of directories and files (pathnames)

Working with the filesystem: key modules

- Work with *names* of directories and files (pathnames)
  - Newer, object-oriented `Pathlib` module (Python 3.5+)

Working with the filesystem: key modules

- Work with *names* of directories and files (pathnames)
  - Newer, object-oriented `Pathlib` module (Python 3.5+)
  - This course: conventional `os.path` module

Working with the filesystem: key modules

- Work with *names* of directories and files (pathnames)
  - Newer, object-oriented `Pathlib` module (Python 3.5+)
  - This course: conventional `os.path` module
- Access the filesystem (disks, file descriptors, directories, etc):

Working with the filesystem: key modules

- Work with *names* of directories and files (pathnames)
  - Newer, object-oriented `Pathlib` module (Python 3.5+)
  - This course: conventional `os.path` module
- Access the filesystem (disks, file descriptors, directories, etc):
  - Basic filesystem tasks: the `os` module (user access rights, removing directories, etc)

Working with the filesystem: key modules

- Work with *names* of directories and files (pathnames)
  - Newer, object-oriented `Pathlib` module (Python 3.5+)
  - This course: conventional `os.path` module
- Access the filesystem (disks, file descriptors, directories, etc):
  - Basic filesystem tasks: the `os` module (user access rights, removing directories, etc)
  - High-level operations: the `shutil` module (copying files, removing directory trees, etc)

Working with the filesystem: key modules

- Work with *names* of directories and files (pathnames)
  - Newer, object-oriented `Pathlib` module (Python 3.5+)
  - This course: conventional `os.path` module
- Access the filesystem (disks, file descriptors, directories, etc):
  - Basic filesystem tasks: the `os` module (user access rights, removing directories, etc)
  - High-level operations: the `shutil` module (copying files, removing directory trees, etc)
- Reading and writing files
  - The `open()` command

Working with the filesystem: key modules

- Work with *names* of directories and files (pathnames)
  - Newer, object-oriented `Pathlib` module (Python 3.5+)
  - This course: conventional `os.path` module
- Access the filesystem (disks, file descriptors, directories, etc):
  - Basic filesystem tasks: the `os` module (user access rights, removing directories, etc)
  - High-level operations: the `shutil` module (copying files, removing directory trees, etc)
- Reading and writing files
  - The `open()` command

Working with filesystems: key functions

- Some important functions of the `os` module:

Working with filesystems: key functions

- Some important functions of the `os` module:
  - `os.getcwd()` obtains current working directory

Working with filesystems: key functions

- Some important functions of the `os` module:
  - `os.getcwd()` obtains current working directory
  - `os.chdir(dir)` changes current working directory to `dir`

Working with filesystems: key functions

- Some important functions of the `os` module:
  - `os.getcwd()` obtains current working directory
  - `os.chdir(dir)` changes current working directory to `dir`
  - `os.rmdir(dir)` removes directory `dir`

Working with filesystems: key functions

- Some important functions of the `os` module:
  - `os.getcwd()` obtains current working directory
  - `os.chdir(dir)` changes current working directory to `dir`
  - `os.rmdir(dir)` removes directory `dir`
  - `os.mkdir(dir)` makes directory `dir`

Working with filesystems: key functions

- Some important functions of the `os` module:
  - `os.getcwd()` obtains current working directory
  - `os.chdir(dir)` changes current working directory to `dir`
  - `os.rmdir(dir)` removes directory `dir`
  - `os.mkdir(dir)` makes directory `dir`
  - `os.listdir(dir=".")` list files in directory `dir`

Working with filesystems: key functions

- Some important functions of the `os` module:
  - `os.getcwd()` obtains current working directory
  - `os.chdir(dir)` changes current working directory to `dir`
  - `os.rmdir(dir)` removes directory `dir`
  - `os.mkdir(dir)` makes directory `dir`
  - `os.listdir(dir=".")` list files in directory `dir`
  - `os.walk(dir)` loop through all subdirectories and files in directory `dir`

Working with filesystems: key functions

- Some important functions of the `os` module:
  - `os.getcwd()` obtains current working directory
  - `os.chdir(dir)` changes current working directory to `dir`
  - `os.rmdir(dir)` removes directory `dir`
  - `os.mkdir(dir)` makes directory `dir`
  - `os.listdir(dir=".")` list files in directory `dir`
  - `os.walk(dir)` loop through all subdirectories and files in directory `dir`

Working with filesystems: key functions II

- Some important functions of the `os.path` module:

Working with filesystems: key functions II

- Some important functions of the `os.path` module:
  - `os.path.exists(path)` checks whether path exists

Working with filesystems: key functions II

- Some important functions of the `os.path` module:
  - `os.path.exists(path)` checks whether path exists
  - `os.path.expanduser("~")` provides home directory name of current user

Working with filesystems: key functions II

- Some important functions of the `os.path` module:
  - `os.path.exists(path)` checks whether path exists
  - `os.path.expanduser("~")` provides home directory name of current user
  - `os.path.join(path1,path2)` concatenates path1 and path2

Working with filesystems: key functions II

- Some important functions of the `os.path` module:
  - `os.path.exists(path)` checks whether path exists
  - `os.path.expanduser("~")` provides home directory name of current user
  - `os.path.join(path1,path2)` concatenates path1 and path2
  - `os.path.split(path)` splits path `a/b/c.txt` into `(a/b, c.txt)`

Working with filesystems: key functions II

- Some important functions of the `os.path` module:
  - `os.path.exists(path)` checks whether path exists
  - `os.path.expanduser("~")` provides home directory name of current user
  - `os.path.join(path1,path2)` concatenates path1 and path2
  - `os.path.split(path)` splits path `a/b/c.txt` into `(a/b, c.txt)`
  - `os.path.splitext(path)` splits path `a/b/c.ext` into `(a/b/c, ext)`

Working with filesystems: key functions II

- Some important functions of the `os.path` module:
  - `os.path.exists(path)` checks whether path exists
  - `os.path.expanduser("~")` provides home directory name of current user
  - `os.path.join(path1,path2)` concatenates path1 and path2
  - `os.path.split(path)` splits path `a/b/c.txt` into `(a/b, c.txt)`
  - `os.path.splitext(path)` splits path `a/b/c.ext` into `(a/b/c, ext)`

Working with paths 1

- Task: what is the current working directory?
  - Access the filesystem using the `os` module:

Working with paths 1

- Task: what is the current working directory?
  - Access the filesystem using the `os` module:

```
1 import os # import modules
```

Working with paths 1

- Task: what is the current working directory?
  - Access the filesystem using the `os` module:

```
1 import os # import modules
2
3 wdir = os.getcwd() # returns pathname (as string)
```

Working with paths 1

- Task: what is the current working directory?
  - Access the filesystem using the `os` module:

```
1 import os # import modules
2
3 wdir = os.getcwd() # returns pathname (as string)
4 print(wdir) # e.g., "C:\"
```

Working with paths 2

- Task: change the current working directory to the home directory:

Working with paths 2

- Task: change the current working directory to the home directory:
  - Access the filesystem using the `os` module
  - Manipulate path names using the `os.path` module

Working with paths 2

- Task: change the current working directory to the home directory:
  - Access the filesystem using the `os` module
  - Manipulate path names using the `os.path` module

```
1 import os, os.path # import modules
```

Working with paths 2

- Task: change the current working directory to the home directory:
  - Access the filesystem using the `os` module
  - Manipulate path names using the `os.path` module

```
1 import os, os.path # import modules
2
3 # Get homedir through expanduser()
4 # "~" denotes homedir in Unix (but this works on
      Windows too)
5 home_dir = os.path.expanduser("~")
```

Working with paths 2

- Task: change the current working directory to the home directory:
  - Access the filesystem using the `os` module
  - Manipulate path names using the `os.path` module

```
1 import os, os.path # import modules
2
3 # Get homedir through expanduser()
4 # "~" denotes homedir in Unix (but this works on
      Windows too)
5 home_dir = os.path.expanduser("~")
6 print(home_dir) # "C:\Users\foo323"
```

Working with paths 2

- Task: change the current working directory to the home directory:
  - Access the filesystem using the `os` module
  - Manipulate path names using the `os.path` module

```
1 import os, os.path # import modules
2
3 # Get homedir through expanduser()
4 # "~" denotes homedir in Unix (but this works on
      Windows too)
5 home_dir = os.path.expanduser("~")
6 print(home_dir) # "C:\Users\foo323"
7
8 # change the directory to the homedir
9 os.chdir(home_dir)
```

Working with paths 2

- Task: change the current working directory to the home directory:
  - Access the filesystem using the `os` module
  - Manipulate path names using the `os.path` module

```
1  import os, os.path # import modules
2
3  # Get homedir through expanduser()
4  # "~" denotes homedir in Unix (but this works on
       Windows too)
5  home_dir = os.path.expanduser("~")
6  print(home_dir) # "C:\Users\foo323"
7
8  # change the directory to the homedir
9  os.chdir(home_dir)
10
11 # check the result
12 print(os.getcwd()) # "C:\Users\foo323"
```

Finding files

- Task: in your home directory, find all directories (non-nested) which contain whitespace

```
1 import re, os, os.path # import the modules
```

Finding files

- Task: in your home directory, find all directories (non-nested) which contain whitespace

```
1 import re, os, os.path # import the modules
2
3 # go to home dir
4 home_dir = os.path.expanduser("~")
```

Finding files

- Task: in your home directory, find all directories (non-nested) which contain whitespace

```
1 import re, os, os.path # import the modules
2
3 # go to home dir
4 home_dir = os.path.expanduser("~")
5 os.chdir(home_dir)
```

Finding files

- Task: in your home directory, find all directories (non-nested) which contain whitespace

```
1 import re, os, os.path # import the modules
2
3 # go to home dir
4 home_dir = os.path.expanduser("~")
5 os.chdir(home_dir)
6
7 # list all files in home dir
8 file_list = os.listdir() # ["Desktop", "Downloads",
      ...]
```

Finding files

- Task: in your home directory, find all directories (non-nested) which contain whitespace

```
1  import re, os, os.path # import the modules
2
3  # go to home dir
4  home_dir = os.path.expanduser("~")
5  os.chdir(home_dir)
6
7  # list all files in home dir
8  file_list = os.listdir() # ["Desktop", "Downloads",
       ...]
9
10 # declare list containing dirs with spaces
11 dirs_with_spaces = []
```

Finding files

- Task: in your home directory, find all directories (non-nested) which contain whitespace

```
1  import re, os, os.path # import the modules
2
3  # go to home dir
4  home_dir = os.path.expanduser("~")
5  os.chdir(home_dir)
6
7  # list all files in home dir
8  file_list = os.listdir() # ["Desktop", "Downloads",
       ...]
9
10 # declare list containing dirs with spaces
11 dirs_with_spaces = []
12
13 for file_i in file_list:
```

Finding files

- Task: in your home directory, find all directories (non-nested) which contain whitespace

```
1  import re, os, os.path # import the modules
2
3  # go to home dir
4  home_dir = os.path.expanduser("~")
5  os.chdir(home_dir)
6
7  # list all files in home dir
8  file_list = os.listdir() # ["Desktop", "Downloads",
       ...]
9
10 # declare list containing dirs with spaces
11 dirs_with_spaces = []
12
13 for file_i in file_list:
14     # check for white space in filename and whether
           file is directory
15     if re.search(r"\s",file_i) is not None and os.path.
           isdir(file_i):
```

Finding files

- Task: in your home directory, find all directories (non-nested) which contain whitespace

```
1  import re, os, os.path # import the modules
2
3  # go to home dir
4  home_dir = os.path.expanduser("~")
5  os.chdir(home_dir)
6
7  # list all files in home dir
8  file_list = os.listdir() # ["Desktop", "Downloads",
       ...]
9
10 # declare list containing dirs with spaces
11 dirs_with_spaces = []
12
13 for file_i in file_list:
14     # check for white space in filename and whether
           file is directory
15     if re.search(r"\s",file_i) is not None and os.path.
           isdir(file_i):
16         dirs_with_spaces += [file_i] # append
```

Finding files

- Task: in your home directory, find all directories (non-nested) which contain whitespace

```
1  import re, os, os.path # import the modules
2
3  # go to home dir
4  home_dir = os.path.expanduser("~")
5  os.chdir(home_dir)
6
7  # list all files in home dir
8  file_list = os.listdir() # ["Desktop", "Downloads",
       ...]
9
10 # declare list containing dirs with spaces
11 dirs_with_spaces = []
12
13 for file_i in file_list:
14     # check for white space in filename and whether
           file is directory
15     if re.search(r"\s",file_i) is not None and os.path.
           isdir(file_i):
16         dirs_with_spaces += [file_i] # append
```

Finding files 2

- Task: list all files *anywhere* within the home directory which have a size larger than 50 kB

Finding files 2

- Task: list all files *anywhere* within the home directory which have a size larger than 50 kB
- Iterate over all files in any subdirectory within the home directory, using os.walk():

Finding files 2

- Task: list all files *anywhere* within the home directory which have a size larger than 50 kB
- Iterate over all files in any subdirectory within the home directory, using os.walk():
  - os.walk() walks the directory tree, returning a tuple for each directory with `(parent_dir, subdirectories, files)`

Finding files using `os.walk()`

- Task: list all files *anywhere* within the home directory which have a size larger than 1 Mb

Finding files using `os.walk()`

- Task: list all files *anywhere* within the home directory which have a size larger than 1 Mb
- Say, we have the following directory tree

```
C:
├── Users
├── Public
│   └── foo323
│       ├── Desktop
│       ├── Downloads
│       │   ├── citation.txt (5 Kb)
│       │   └── boring_paper.pdf (5 Mb)
│       ├── Scripts
│       ├── groceries.docx (10 Kb)
│       └── manuscript.docx (2 Mb)
```

Finding files using `os.walk()`

- Task: list all files *anywhere* within the home directory which have a size larger than 1 Mb
- Say, we have the following directory tree

```
C:
├── Users
├── Public
│   └── foo323
│       ├── Desktop
│       ├── Downloads
│       │   ├── citation.txt (5 Kb)
│       │   └── boring_paper.pdf (5 Mb)
│       ├── Scripts
│       ├── groceries.docx (10 Kb)
│       └── manuscript.docx (2 Mb)
```

- We should return `Downloads/boring_paper.pdf` and `manuscript.docx`

Finding files using `os.walk()`

- To illustrate how os.walk() works, we iterate over the home directory and then exit the loop through `break`:

```
1 import os
2
3 # get home directory
4 homedir = os.path.expanduser("~")
```

Finding files using `os.walk()`

- To illustrate how os.walk() works, we iterate over the home directory and then exit the loop through `break`:

```
1 import os
2
3 # get home directory
4 homedir = os.path.expanduser("~")
5
6 # iterate over all files nested in the home directory
7 for parent_dir, subdirs, files in os.walk(homedir):
```

Finding files using `os.walk()`

- To illustrate how os.walk() works, we iterate over the home directory and then exit the loop through `break`:

```
1 import os
2
3 # get home directory
4 homedir = os.path.expanduser("~")
5
6 # iterate over all files nested in the home directory
7 for parent_dir, subdirs, files in os.walk(homedir):
8     print(parent_dir) # C:\Users\Public\foo323
```

Finding files using `os.walk()`

- To illustrate how os.walk() works, we iterate over the home directory and then exit the loop through `break`:

```
1 import os
2
3 # get home directory
4 homedir = os.path.expanduser("~")
5
6 # iterate over all files nested in the home directory
7 for parent_dir, subdirs, files in os.walk(homedir):
8     print(parent_dir) # C:\Users\Public\foo323
9     print(subdirs ) # ['Desktop', 'Downloads', 'Scripts
          ']
```

Finding files using `os.walk()`

- To illustrate how os.walk() works, we iterate over the home directory and then exit the loop through `break`:

```
1  import os
2
3  # get home directory
4  homedir = os.path.expanduser("~")
5
6  # iterate over all files nested in the home directory
7  for parent_dir, subdirs, files in os.walk(homedir):
8      print(parent_dir) # C:\Users\Public\foo323
9      print(subdirs ) # ['Desktop', 'Downloads', 'Scripts
           ']
10     print(files) # ['groceries.docx', 'manuscript.docx
           ']
```

Finding files using `os.walk()`

- To illustrate how os.walk() works, we iterate over the home directory and then exit the loop through `break`:

```
1  import os
2
3  # get home directory
4  homedir = os.path.expanduser("~")
5
6  # iterate over all files nested in the home directory
7  for parent_dir, subdirs, files in os.walk(homedir):
8      print(parent_dir) # C:\Users\Public\foo323
9      print(subdirs ) # ['Desktop', 'Downloads', 'Scripts
           ']
10     print(files) # ['groceries.docx', 'manuscript.docx
           ']
11
12     # quit after the first iteration
13     break
```

Finding files using `os.walk()`

- To illustrate how os.walk() works, we iterate over the home directory and then exit the loop through `break`:

```
1 import os, os.path
2
3 # get home directory
4 homedir = os.path.expanduser("~")
```

Finding files using `os.walk()`

- To illustrate how os.walk() works, we iterate over the home directory and then exit the loop through `break`:

```
1  import os, os.path
2
3  # get home directory
4  homedir = os.path.expanduser("~")
5
6  # make a list to store the files
7  files_larger_1mb = []
8
9  # iterate over all files nested in the home directory
10 for parent_dir, subdirs, files in os.walk(homedir):
```

Finding files using `os.walk()`

- To illustrate how os.walk() works, we iterate over the home directory and then exit the loop through `break`:

```
1  import os, os.path
2
3  # get home directory
4  homedir = os.path.expanduser("~")
5
6  # make a list to store the files
7  files_larger_1mb = []
8
9  # iterate over all files nested in the home directory
10 for parent_dir, subdirs, files in os.walk(homedir):
11
12     for file in files:
```

Finding files using `os.walk()`

- To illustrate how os.walk() works, we iterate over the home directory and then exit the loop through `break`:

```
1  import os, os.path
2
3  # get home directory
4  homedir = os.path.expanduser("~")
5
6  # make a list to store the files
7  files_larger_1mb = []
8
9  # iterate over all files nested in the home directory
10 for parent_dir, subdirs, files in os.walk(homedir):
11
12     for file in files:
13
14         # get the full path name
15         full_path = os.path.join(parent_dir, file)
```

Finding files using `os.walk()`

- To illustrate how os.walk() works, we iterate over the home directory and then exit the loop through `break`:

```
1  import os, os.path
2
3  # get home directory
4  homedir = os.path.expanduser("~")
5
6  # make a list to store the files
7  files_larger_1mb = []
8
9  # iterate over all files nested in the home directory
10 for parent_dir, subdirs, files in os.walk(homedir):
11
12     for file in files:
13
14         # get the full path name
15         full_path = os.path.join(parent_dir, file)
16
17         if os.path.exists(full_path):
```

Finding files using `os.walk()`

- To illustrate how os.walk() works, we iterate over the home directory and then exit the loop through `break`:

```
1  import os, os.path
2
3  # get home directory
4  homedir = os.path.expanduser("~")
5
6  # make a list to store the files
7  files_larger_1mb = []
8
9  # iterate over all files nested in the home directory
10 for parent_dir, subdirs, files in os.walk(homedir):
11
12     for file in files:
13
14         # get the full path name
15         full_path = os.path.join(parent_dir, file)
16
17         if os.path.exists(full_path):
18             size = os.path.getsize(full_path)
```

Finding files using `os.walk()`

- To illustrate how os.walk() works, we iterate over the home directory and then exit the loop through `break`:

```
1  import os, os.path
2
3  # get home directory
4  homedir = os.path.expanduser("~")
5
6  # make a list to store the files
7  files_larger_1mb = []
8
9  # iterate over all files nested in the home directory
10 for parent_dir, subdirs, files in os.walk(homedir):
11
12     for file in files:
13
14         # get the full path name
15         full_path = os.path.join(parent_dir, file)
16
17         if os.path.exists(full_path):
18             size = os.path.getsize(full_path)
19
20             if size / 1024 > 1000:
21                 files_larger_1mb += [full_path]
```

Finding files using `os.walk()`

- To illustrate how os.walk() works, we iterate over the home directory and then exit the loop through `break`:

```
1  import os, os.path
2
3  # get home directory
4  homedir = os.path.expanduser("~")
5
6  # make a list to store the files
7  files_larger_1mb = []
8
9  # iterate over all files nested in the home directory
10 for parent_dir, subdirs, files in os.walk(homedir):
11
12     for file in files:
13
14         # get the full path name
15         full_path = os.path.join(parent_dir, file)
16
17         if os.path.exists(full_path):
18             size = os.path.getsize(full_path)
19
20             if size / 1024 > 1000:
21                 files_larger_1mb += [full_path]
22
23 # print folder contents
24 print(files_larger_1mb)
```

Copying files using `shutil`

- Task: make a file and then copy it to another file using `shutil`

```
1 import shutil
2
3 # store a filename
4 filename = "new_file.txt"
```

Copying files using `shutil`

- Task: make a file and then copy it to another file using `shutil`

```
1 import shutil
2
3 # store a filename
4 filename = "new_file.txt"
5
6 f = open("new_file.txt","w")
```

Copying files using `shutil`

- Task: make a file and then copy it to another file using `shutil`

```
1 import shutil
2
3 # store a filename
4 filename = "new_file.txt"
5
6 f = open("new_file.txt","w")
7 f.write("some text")
```

Copying files using `shutil`

- Task: make a file and then copy it to another file using `shutil`

```
1 import shutil
2
3 # store a filename
4 filename = "new_file.txt"
5
6 f = open("new_file.txt","w")
7 f.write("some text")
8 f.close()
```

Copying files using `shutil`

- Task: make a file and then copy it to another file using `shutil`

```
1  import shutil
2
3  # store a filename
4  filename = "new_file.txt"
5
6  f = open("new_file.txt","w")
7  f.write("some text")
8  f.close()
9
10 # now copy using shutil
11 shutil.copy(filename, "another_new_file.txt")
```

Copying files using `shutil`

- Task: make a file and then copy it to another file using `shutil`

```
1  import shutil
2
3  # store a filename
4  filename = "new_file.txt"
5
6  f = open("new_file.txt","w")
7  f.write("some text")
8  f.close()
9
10 # now copy using shutil
11 shutil.copy(filename, "another_new_file.txt")
12
13 # list all the files in the current directory
14 # to see whether copied file exists
15 print(os.listdir("."))
```

Copying files using `shutil`

- Task: make a file and then copy it to another file using `shutil`

```
1  import shutil
2
3  # store a filename
4  filename = "new_file.txt"
5
6  f = open("new_file.txt","w")
7  f.write("some text")
8  f.close()
9
10 # now copy using shutil
11 shutil.copy(filename, "another_new_file.txt")
12
13 # list all the files in the current directory
14 # to see whether copied file exists
15 print(os.listdir("."))
```

Writing files

- open a file for writing, using the `open()` command

Writing files

- open a file for writing, using the `open()` command

```
1 # the 'w' flag reflects that we write to a file
2 # any existing contents will be overwritten
3 the_file_object = open("my_first_file","w")
```

Writing files

- open a file for writing, using the `open()` command

```
1 # the 'w' flag reflects that we write to a file
2 # any existing contents will be overwritten
3 the_file_object = open("my_first_file","w")
4
5 # write a string to the file
6 the_file_object.write("Hello world!")
```

Writing files

- open a file for writing, using the `open()` command

```
1 # the 'w' flag reflects that we write to a file
2 # any existing contents will be overwritten
3 the_file_object = open("my_first_file","w")
4
5 # write a string to the file
6 the_file_object.write("Hello world!")
7
8 # always close the file
9 the_file_object.close()
```

Reading files

- open a file for reading, using the `read()` command

Reading files

- open a file for reading, using the `read()` command

```
1 # the 'r' flag reflects that we read from a file
2 the_file_object = open("my_first_file","r")
```

Reading files

- open a file for reading, using the `read()` command

```
1 # the 'r' flag reflects that we read from a file
2 the_file_object = open("my_first_file","r")
3
4 # get the file contents as a string
5 file_contents = the_file_object.read()
```

Reading files

- open a file for reading, using the `read()` command

```
1 # the 'r' flag reflects that we read from a file
2 the_file_object = open("my_first_file","r")
3
4 # get the file contents as a string
5 file_contents = the_file_object.read()
6
7 # always close the file
8 the_file_object.close()
```

Reading files

- open a file for reading, using the `read()` command

```
1  # the 'r' flag reflects that we read from a file
2  the_file_object = open("my_first_file","r")
3
4  # get the file contents as a string
5  file_contents = the_file_object.read()
6
7  # always close the file
8  the_file_object.close()
9
10 # process file output
11 print(file_contents) # Hello World!
```

Other file operations

- File operations can be specified by the flag provided to `open()` function

Other file operations

- File operations can be specified by the flag provided to `open()` function

Flag	File operation
`"w"`	Write to a file, file will be truncated first
`"r"`	Reading from a file
`"r+"`	Reading and writing to a file
`"a"`	Append to a file
`"a+"`	Read from and write (by appending) to a file
`"x"`	Exclusive creation, fails if file exists

Appending to a file

- Append text to a previously opened file using the `a+` flag

```
1 # first write something to a new file
2 myfile = open("myfile","w")
```

Appending to a file

- Append text to a previously opened file using the `a+` flag

```
1 # first write something to a new file
2 myfile = open("myfile","w")
3 myfile.write("I wrote this to a file!")
4 myfile.close()
5
6 myfile = open("myfile","a+")
```

Appending to a file

- Append text to a previously opened file using the `a+` flag

```
1  # first write something to a new file
2  myfile = open("myfile","w")
3  myfile.write("I wrote this to a file!")
4  myfile.close()
5
6  myfile = open("myfile","a+")
7  myfile.write("\nAnd now I also wrote this!")
8
9  # get the file contents as a string
10 file_contents = myfile.read()
```

Appending to a file

- Append text to a previously opened file using the `a+` flag

```
1  # first write something to a new file
2  myfile = open("myfile","w")
3  myfile.write("I wrote this to a file!")
4  myfile.close()
5
6  myfile = open("myfile","a+")
7  myfile.write("\nAnd now I also wrote this!")
8
9  # get the file contents as a string
10 file_contents = myfile.read()
11
12 # always close the file
13 myfile.close()
```

Appending to a file

- Append text to a previously opened file using the a+ flag

```
1  # first write something to a new file
2  myfile = open("myfile","w")
3  myfile.write("I wrote this to a file!")
4  myfile.close()
5
6  myfile = open("myfile","a+")
7  myfile.write("\nAnd now I also wrote this!")
8
9  # get the file contents as a string
10 file_contents = myfile.read()
11
12 # always close the file
13 myfile.close()
14
15 # process file output
16 print(file_contents) # Nothing!
```

Appending to a file

- Append text to a previously opened file using the `a+` flag

```
1  # first write something to a new file
2  myfile = open("myfile","w")
3  myfile.write("I wrote this to a file!")
4  myfile.close()
5
6  myfile = open("myfile","a+")
7  myfile.write("\nAnd now I also wrote this!")
8
9  # get the file contents as a string
10 file_contents = myfile.read()
11
12 # always close the file
13 myfile.close()
14
15 # process file output
16 print(file_contents) # Nothing!
```

- No output because the internal file pointer used by `file.read()` is at the end of the file!

Appending to a file

- Solution: use the `file.seek()` command to change the internal file pointer position

Appending to a file

- Solution: use the `file.seek()` command to change the internal file pointer position

```
1 # first write something to a new file
2 myfile = open("myfile","w")
```

Appending to a file

- Solution: use the `file.seek()` command to change the internal file pointer position

```
1 # first write something to a new file
2 myfile = open("myfile","w")
3 myfile.write("I wrote this to a file!")
4 myfile.close()
5
6 myfile = open("myfile","a+")
```

Appending to a file

- Solution: use the `file.seek()` command to change the internal file pointer position

```
1  # first write something to a new file
2  myfile = open("myfile","w")
3  myfile.write("I wrote this to a file!")
4  myfile.close()
5
6  myfile = open("myfile","a+")
7  myfile.write("\nAnd now I also wrote this!")
8
9  # move the file pointer to the 0th byte of the file
10 myfile.seek(0)
```

Appending to a file

- Solution: use the `file.seek()` command to change the internal file pointer position

```
1  # first write something to a new file
2  myfile = open("myfile","w")
3  myfile.write("I wrote this to a file!")
4  myfile.close()
5
6  myfile = open("myfile","a+")
7  myfile.write("\nAnd now I also wrote this!")
8
9  # move the file pointer to the 0th byte of the file
10 myfile.seek(0)
11
12 # get the file contents as a string
13 file_contents = myfile.read()
14
15 # always close the file
16 myfile.close()
```

Appending to a file

- Solution: use the `file.seek()` command to change the internal file pointer position

```
1  # first write something to a new file
2  myfile = open("myfile","w")
3  myfile.write("I wrote this to a file!")
4  myfile.close()
5
6  myfile = open("myfile","a+")
7  myfile.write("\nAnd now I also wrote this!")
8
9  # move the file pointer to the 0th byte of the file
10 myfile.seek(0)
11
12 # get the file contents as a string
13 file_contents = myfile.read()
14
15 # always close the file
16 myfile.close()
17
18 print(file_contents) # I wrote this to a file!
19                      # And now I also wrote this!
```

Appending to a file

- Solution: use the `file.seek()` command to change the internal file pointer position

```
1  # first write something to a new file
2  myfile = open("myfile","w")
3  myfile.write("I wrote this to a file!")
4  myfile.close()
5
6  myfile = open("myfile","a+")
7  myfile.write("\nAnd now I also wrote this!")
8
9  # move the file pointer to the 0th byte of the file
10 myfile.seek(0)
11
12 # get the file contents as a string
13 file_contents = myfile.read()
14
15 # always close the file
16 myfile.close()
17
18 print(file_contents) # I wrote this to a file!
19                      # And now I also wrote this!
```