

Python for scientific research

Input, output and the filesystem

Bram Kuijper

University of Exeter, Penryn Campus, UK

February 24, 2020



Researcher
Development



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

Filesystem: introduction

- Nearly every program will **read and write files** (also called file IO; file input and output)

Filesystem: introduction

- Nearly every program will **read and write files** (also called file IO; file input and output)
- The file IO workhorse is the **open()** function, which opens a file and returns a file object

Filesystem: introduction

- Nearly every program will **read and write files** (also called file IO; file input and output)
- The file IO workhorse is the **open()** function, which opens a file and returns a file object
- Next, there are several utilities to find out information about files and directories contained in modules like **os**, **os.path** and **shutil**

Using `open()`: writing files

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

Using `open()`: writing files

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

```
1 # mode="w" keyword argument: we write to a file
2 # any existing contents will be overwritten
3 file_obj = open(file="myfile.txt",mode="w")
```

Using `open()`: writing files

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

```
1 # mode="w" keyword argument: we write to a file
2 # any existing contents will be overwritten
3 file_obj = open(file="myfile.txt",mode="w")
4
5 # write a string to the file
6 file_obj.write("Hello world!")
```


Using `open()`: writing files

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

```
1 # mode="w" keyword argument: we write to a file
2 # any existing contents will be overwritten
3 file_obj = open(file="myfile.txt",mode="w")
4
5 # write a string to the file
6 file_obj.write("Hello world!")
7
8 # always close the file
9 file_obj.close()
```

Using `open()`: writing files

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

```
1 # mode="w" keyword argument: we write to a file
2 # any existing contents will be overwritten
3 file_obj = open(file="myfile.txt",mode="w")
4
5 # write a string to the file
6 file_obj.write("Hello world!")
7
8 # always close the file
9 file_obj.close()
```

- The above listing is bad practice, however: any error in lines 4 - 8 will cause python to quit before `file_obj.close()` is called, leaving `file_obj` open

Using `open()`: writing files II

- Instead, we need to use a `with` statement

Using `open()`: writing files II

- Instead, we need to use a `with` statement

```
1 with open(file="myfile.txt",mode="w") as file_obj:
```

Using `open()`: writing files II

- Instead, we need to use a `with` statement

```
1 with open(file="myfile.txt",mode="w") as file_obj:  
2     file_obj.write("Hello world!")
```

Using `open()`: writing files II

- Instead, we need to use a `with` statement

```
1 with open(file="myfile.txt",mode="w") as file_obj:  
2     file_obj.write("Hello world!")
```

- Line 1: the variable `file_obj` gets assigned the return value of `open()`

Using `open()`: writing files II

- Instead, we need to use a `with` statement

```
1 with open(file="myfile.txt",mode="w") as file_obj:  
2     file_obj.write("Hello world!")
```

- Line 1: the variable `file_obj` gets assigned the return value of `open()`
- Line 2: we perform operations on `file_obj`

Using `open()`: writing files II

- Instead, we need to use a `with` statement

```
1 with open(file="myfile.txt",mode="w") as file_obj:  
2     file_obj.write("Hello world!")
```

- Line 1: the variable `file_obj` gets assigned the return value of `open()`
- Line 2: we perform operations on `file_obj`
- In case line 2 finishes or ends in an error, a hidden `__exit().__` function is called on `file_obj`, closing it

Using `open()`: writing files II

- Instead, we need to use a `with` statement

```
1 with open(file="myfile.txt",mode="w") as file_obj:  
2     file_obj.write("Hello world!")
```

- Line 1: the variable `file_obj` gets assigned the return value of `open()`
- Line 2: we perform operations on `file_obj`
- In case line 2 finishes or ends in an error, a hidden `__exit()` function is called on `file_obj`, closing it
- Bottom line: when using `with`, `file_obj` will not remain open

Reading files, using `open()`

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

Reading files, using open()

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

```
1 # mode="r" keyword argument: we read from a file
2 with open(file="myfile.txt",mode="r") as file_obj:
3     # get the file contents as a string
4     file_contents = file_obj.read()
```

Reading files, using open()

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

```
1 # mode="r" keyword argument: we read from a file
2 with open(file="myfile.txt",mode="r") as file_obj:
3     # get the file contents as a string
4     file_contents = file_obj.read()
5
6 # process file output
7 print(file_contents) # Hello World!
```

Other modes of the `open()` function

- Different file opening modes can be specified by mode keyword of the `open()` function

Other modes of the `open()` function

- Different file opening modes can be specified by mode keyword of the `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 (no truncation)
"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 `mode="a+":`

```
1 # first write something to a new file
2 with open(file="myfile.txt",mode="w") as file_obj:
```

Appending to a file

- Append text to a previously opened file using `mode="a+"`:

```
1 # first write something to a new file
2 with open(file="myfile.txt",mode="w") as file_obj:
3     file_obj.write("I wrote this to a file!")
4
5 with open(file="myfile",mode="a+") as file_obj:
```


Appending to a file

- Append text to a previously opened file using `mode="a+"`:

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

Appending to a file

- Append text to a previously opened file using `mode="a+"`:

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

Appending to a file

- Append text to a previously opened file using `mode="a+"`:

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


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

Position of file pointer

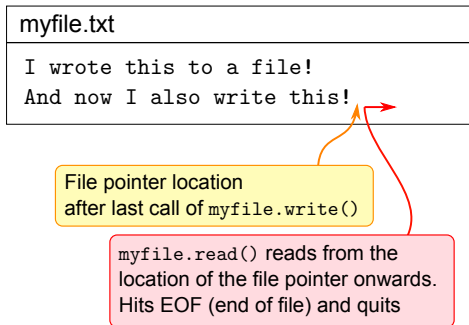
myfile.txt

I wrote this to a file!
And now I also write this!

File pointer location
after last call of `myfile.write()`



Position of file pointer



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 with open(file="myfile.txt",mode="w") as file_obj:
```

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 with open(file="myfile.txt",mode="w") as file_obj:
3     file_obj.write("I wrote this to a file!")
4
5 # open file again, now to append and read:
6 with open(file="myfile.txt",mode="a+") as file_obj:
```


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 with open(file="myfile.txt",mode="w") as file_obj:
3     file_obj.write("I wrote this to a file!")
4
5 # open file again, now to append and read:
6 with open(file="myfile.txt",mode="a+") as file_obj:
7     file_obj.write("\nAnd now I also wrote this!")
8
9     # move the file pointer to the 0th byte (start)
10    # of the file:
11    file_obj.seek(0)
12
13    # get the file contents as a string
14    file_contents = file_obj.read()
```

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 with open(file="myfile.txt",mode="w") as file_obj:
3     file_obj.write("I wrote this to a file!")
4
5 # open file again, now to append and read:
6 with open(file="myfile.txt",mode="a+") as file_obj:
7     file_obj.write("\nAnd now I also wrote this!")
8
9     # move the file pointer to the 0th byte (start)
10    # of the file:
11    file_obj.seek(0)
12
13    # get the file contents as a string
14    file_contents = file_obj.read()
15
16 # process file output
17 print(file_contents) # output: I 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 with open(file="myfile.txt",mode="w") as file_obj:
3     file_obj.write("I wrote this to a file!")
4
5 # open file again, now to append and read:
6 with open(file="myfile.txt",mode="a+") as file_obj:
7     file_obj.write("\nAnd now I also wrote this!")
8
9     # move the file pointer to the 0th byte (start)
10    # of the file:
11    file_obj.seek(0)
12
13    # get the file contents as a string
14    file_contents = file_obj.read()
15
16 # process file output
17 print(file_contents) # output: I wrote this...
```

Working with the filesystem: key modules

- Reading and writing files
 - The `open()` command

Working with the filesystem: key modules

- Reading and writing files
 - The `open()` command
- Work with *names* of directories and files (pathnames)

Working with the filesystem: key modules

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

Working with the filesystem: key modules

- Reading and writing files
 - The `open()` command
- 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

- Reading and writing files
 - The `open()` command
- 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

- Reading and writing files
 - The `open()` command
- 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

- Reading and writing files
 - The `open()` command
- 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

- Reading and writing files
 - The `open()` command
- 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 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)`

- 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 subdirectories (non-nested) which contain whitespace

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

Finding files

- Task: in your home directory, find all subdirectories (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(path="~")
```

Finding files

- Task: in your home directory, find all subdirectories (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(path="~")
5 os.chdir(path=home_dir)
```


Finding files

- Task: in your home directory, find all subdirectories (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(path="~")
5 os.chdir(path=home_dir)
6
7 # list all files in home dir
8 flist = os.listdir() #["Desktop", "Downloads",...]
```

Finding files

- Task: in your home directory, find all subdirectories (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(path="~")
5 os.chdir(path=home_dir)
6
7 # list all files in home dir
8 flist = 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 subdirectories (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(path="~")
5 os.chdir(path=home_dir)
6
7 # list all files in home dir
8 flist = os.listdir() #["Desktop", "Downloads",...]
9
10 # declare list containing dirs with spaces
11 dirs_with_spaces = []
12
13 for file_i in flist:
```

Finding files

- Task: in your home directory, find all subdirectories (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(path="~")
5 os.chdir(path=home_dir)
6
7 # list all files in home dir
8 flist = os.listdir() #["Desktop", "Downloads",...]
9
10 # declare list containing dirs with spaces
11 dirs_with_spaces = []
12
13 for file_i in flist:
14     # check for white space in filename and whether
15     # file is directory
16     if re.search(pattern=r"\s",string=file_i) != None
17         and os.path.isdir(file_i):
```

Finding files

- Task: in your home directory, find all subdirectories (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(path=~")
5 os.chdir(path=home_dir)
6
7 # list all files in home dir
8 flist = os.listdir() #["Desktop", "Downloads",...]
9
10 # declare list containing dirs with spaces
11 dirs_with_spaces = []
12
13 for file_i in flist:
14     # check for white space in filename and whether
15     # file is directory
16     if re.search(pattern=r"\s",string=file_i) != None
17         and os.path.isdir(file_i):
18         dirs_with_spaces.append(file_i)
```

Finding files

- Task: in your home directory, find all subdirectories (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(path="~")
5 os.chdir(path=home_dir)
6
7 # list all files in home dir
8 flist = os.listdir() #["Desktop", "Downloads",...]
9
10 # declare list containing dirs with spaces
11 dirs_with_spaces = []
12
13 for file_i in flist:
14     # check for white space in filename and whether
15     # file is directory
16     if re.search(pattern=r"\s",string=file_i) != None
17         and os.path.isdir(file_i):
18         dirs_with_spaces.append(file_i)
```

- 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()`:

- 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(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
2
3 # get home directory
4 homedir = os.path.expanduser(path="~")
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(path="~")
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(path="~")
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(path="~")
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         ']'
11     print(files) # ['groceries.docx', 'manuscript.docx
12         ']
```

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(path="~")
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                    '']
11     print(files) # ['groceries.docx', 'manuscript.docx',
12                  '']
13
14 # quit after the first iteration
15 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 with open(file="new_file.txt",mode="w") as f:
```

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 with open(file="new_file.txt", mode="w") as f:
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 with open(file="new_file.txt",mode="w") as f:
7     f.write("some text")
8
9 # now copy using shutil
10 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 with open(file="new_file.txt",mode="w") as f:
7     f.write("some text")
8
9 # now copy using shutil
10 shutil.copy(filename, "another_new_file.txt")
11
12 # list all the files in the current directory
13 # to see whether copied file exists
14 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 with open(file="new_file.txt",mode="w") as f:
7     f.write("some text")
8
9 # now copy using shutil
10 shutil.copy(filename, "another_new_file.txt")
11
12 # list all the files in the current directory
13 # to see whether copied file exists
14 print(os.listdir("."))
```