

# Modules and File Handles

Eric Franzosa (franzosa@hsph.harvard.edu)

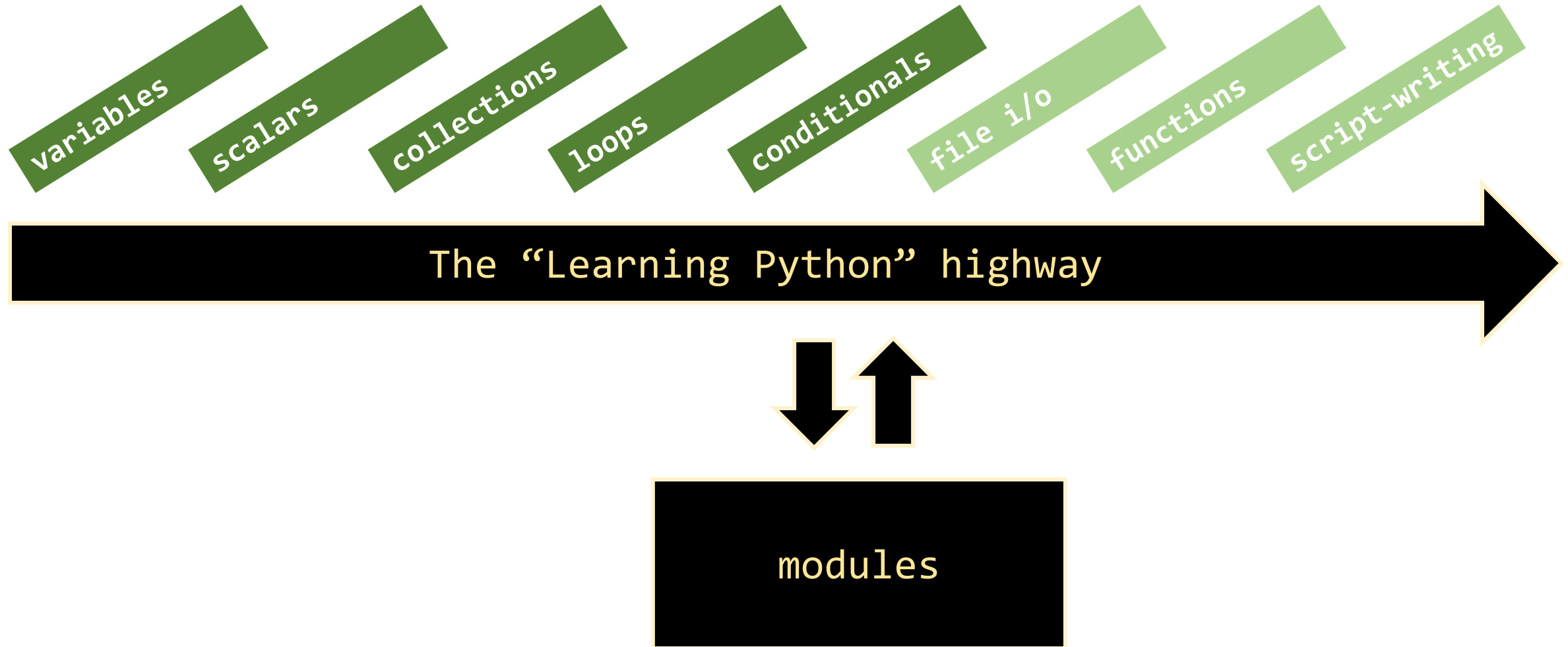
<http://franzosa.net/bst273>

# Overview

- Working with modules
- File I/O (input/output) and file handles
- Practice
- Revisit operator precedence (time permitting)

# Modules

# A little detour...



# What is a module?

- A **module** (or **package**) is Python code designed to be re-used in other code
  - Sometimes referred to as “Libraries” in other languages
- Contributes to our “Second Rule of Programming” from Day 1:
  - Be appropriately lazy
  - Don’t reinvent the wheel
- Also helps with organizing code related to a particular domain

# What is a module?

- A module can be as small as a single Python script
  - In fact, every Python script can be used as a module (more on that shortly...)
- Some modules come bundled with Python
  - The Python “Standard Library”
- Other (more specialized) modules can be installed separately
  - For example, the Python “scientific stack”: [scipy](#), [numpy](#), [pandas](#), and [matplotlib](#)
  - Anaconda installed a bunch of these for us automatically
- You can write your own modules
  - <https://bitbucket.org/franzosa/zopy>

# Using modules

- We use modules by **importing** them into our code
- We got a sneak peak of this earlier -- does anyone remember when?

```
import time
for i in range( 5 ):
    i2 = i ** 2
    print( "i=", i, "i2=", i2 )
    time.sleep( 1 )
```

- **time** is a built-in module for dealing with matters related to, well, time
- <https://docs.python.org/3/library/time.html>
  - We'll look at module documentation shortly

# Using modules

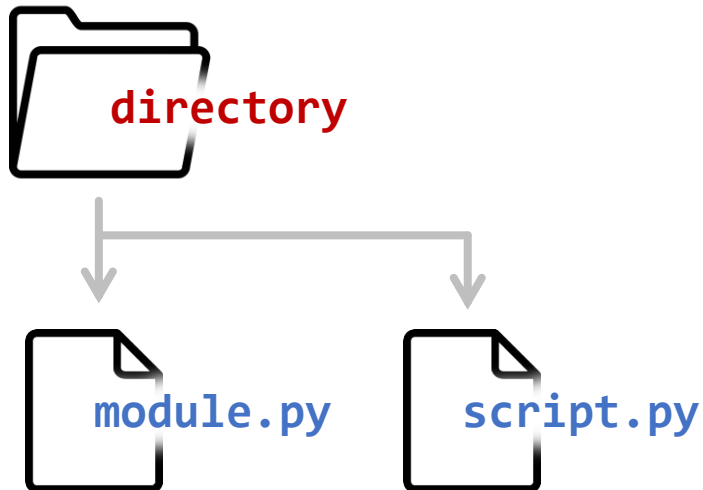
- Modules contain the same sort of elements as other Python code
- Modules can contain variables
  - `math.pi` *contains the value of pi (to many decimal places)*
  - `string.uppercase` *contains the uppercase English alphabet*
- Modules can contain functions
  - `time.sleep` *pauses computation for N seconds*
  - `math.sqrt` *returns the square root of a number*
- Modules can contain classes defining other data types
  - `collections.Counter` *a special dictionary for counting*
  - `Bio.Seq` *special strings for representing biological sequences*



# Module concepts

# Module concepts

- The following example assumes I have two Python files in the same folder
  - `script.py` is a new script I am working on
  - `module.py` is some existing code that I want to re-use



# Module concepts

`module.py` (open in Atom)

```
# an approximation of pi
pi = 3.14

# area of a circle
def area( r ):
    return pi * r ** 2

# first few primes
primes = [2, 3, 5, 7, 11]
```

`script.py` (open in Atom)

```
print( pi )
```

(a terminal)

```
$ python script.py

NameError: name 'pi' is
not defined
```

# Module concepts

`module.py` (open in Atom)

```
# an approximation of pi
pi = 3.14

# area of a circle
def area( r ):
    return pi * r ** 2

# first few primes
primes = [2, 3, 5, 7, 11]
```

`script.py` (open in Atom)

```
import module

print( pi )
```

Module “import”  
statements are among the  
first lines in a Python file

(a terminal)

```
$ python script.py

NameError: name 'pi' is
not defined
```

# Module concepts

`module.py` (open in Atom)

```
# an approximation of pi
pi = 3.14

# area of a circle
def area( r ):
    return pi * r ** 2

# first few primes
primes = [2, 3, 5, 7, 11]
```

`script.py` (open in Atom)

```
import module

print( module.pi )
```

“module” is a **namespace**:  
a collection of previously  
defined objects  
(variables, functions, etc.)

We request individual  
objects using “.” syntax

(a terminal)

```
$ python script.py

3.14
```

# Module concepts

`module.py` (open in Atom)

```
# an approximation of pi
pi = 3.14

# area of a circle
def area( r ):
    return pi * r ** 2

# first few primes
primes = [2, 3, 5, 7, 11]
```

`script.py` (open in Atom)

```
import module
import math

print( module.pi )
print( math.pi )
```

“math” is a separate namespace: using namespaces helps to avoid “collisions” (redefining a second variable with the same name)

(a terminal)

```
$ python script.py

3.14
3.14159265359
```

# Module concepts

`module.py` (open in Atom)

```
# an approximation of pi
pi = 3.14

# area of a circle
def area( r ):
    return pi * r ** 2

# first few primes
primes = [2, 3, 5, 7, 11]
```

`script.py` (open in Atom)

```
import module
import math

pi = 3

print( module.pi )
print( math.pi )
print( pi )
```

The running script has its own namespace (called “\_\_main\_\_”). Here, `pi` is also defined in that namespace.

(a terminal)

```
$ python script.py

3.14
3.14159265359
3
```

# Module concepts

`module.py` (open in Atom)

```
# an approximation of pi
pi = 3.14

# area of a circle
def area( r ):
    return pi * r ** 2

# first few primes
primes = [2, 3, 5, 7, 11]
```

`script.py` (open in Atom)

```
from module import area, pi

print( pi )
print( area( 2 ) )
```

We can also import *specific* variables/functions from a module into the main namespace as a comma-separated list.

(a terminal)

```
$ python script.py

3.14
12.56
```



# Module concepts

`module.py` (open in Atom)

```
# an approximation of pi
pi = 3.14

# area of a circle
def area( r ):
    return pi * r ** 2

# first few primes
primes = [2, 3, 5, 7, 11]
```

`script.py` (open in Atom)

```
from module import area, pi

pi = 3

print( pi )
print( area( 2 ) )
```

Once we import a name,  
we can overwrite it (this is  
often something you want  
to avoid).

(Tricky!) Note that `area`  
still uses `module.pi`.

(a terminal)

```
$ python script.py

3
12.56
```

# Module concepts

`module.py` (open in Atom)

```
# an approximation of pi
pi = 3.14

# area of a circle
def area( r ):
    return pi * r ** 2

# first few primes
primes = [2, 3, 5, 7, 11]
```

`script.py` (open in Atom)

```
from module import primes

primes.append( 13 )
print( primes )

from module import primes

print( primes )
```

Nothing we do in this box  
actually changes `module.py`  
(the original is always  
loaded as a fresh copy).

(a terminal)

```
$ python script.py

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

# Module concepts

`module.py` (open in Atom)

```
# an approximation of pi
pi = 3.14

# area of a circle
def area( r ):
    return pi * r ** 2

# first few primes
primes = [2, 3, 5, 7, 11]
```

`script.py` (open in Atom)

```
from module import *
```

Imports everything from  
module into the main  
namespace. You will see  
code that does this, but it is  
sloppy!

(a terminal)

```
$ python script.py
```

# Module concepts

`module.py` (open in Atom)

```
# an approximation of pi
pi = 3.14

# area of a circle
def area( r ):
    return pi * r ** 2

# first few primes
primes = [2, 3, 5, 7, 11]
```

`script.py` (open in Atom)

```
import module as m

print( m.pi )
```

If a module has a particularly long name, we can use “import...as” to give it an alias in our code.

Some “famous” modules have common nicknames, for example:

```
import numpy as np
```

(a terminal)

```
$ python script.py

3.14
```

# Module concepts

`module.py` (open in Atom)

```
# an approximation of pi
pi = 3.14

# area of a circle
def area( r ):
    return pi * r ** 2

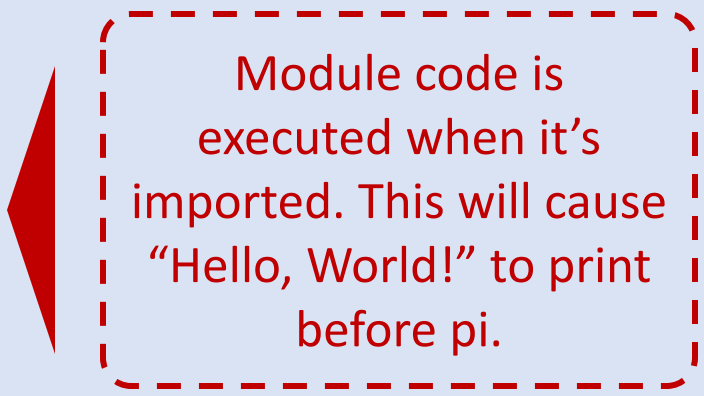
# first few primes
primes = [2, 3, 5, 7, 11]

# say hello
print( "Hello, World!" )
```

`script.py` (open in Atom)

```
import module

print( module.pi )
```



Module code is executed when it's imported. This will cause "Hello, World!" to print before pi.

(a terminal)

```
$ python script.py

"Hello, World!"
3.14
```

# Module concepts

module.py (open in Atom)

```
# an approximation of pi
pi = 3.14

# area of a circle
def area( r ):
    return pi * r ** 2

# first few primes
primes = [2, 3, 5, 7, 11]

# say hello in script mode
if __name__ == "__main__":
    print( "Hello, World!" )
```

script.py (open in Atom)

```
import module

print( module.pi )
```

We can use a special conditional to indicate that some code should only be run when the module is run as a script!

*(We'll come back to this next week)*

(a terminal)

```
$ python script.py

3.14
```

# Module concepts

`module.py` (open in Atom)

```
# an approximation of pi
pi = 3.14

# area of a circle
def area( r ):
    return pi * r ** 2

# first few primes
primes = [2, 3, 5, 7, 11]

# say hello in script mode
if __name__ == "__main__":
    print( "Hello, World!" )
```

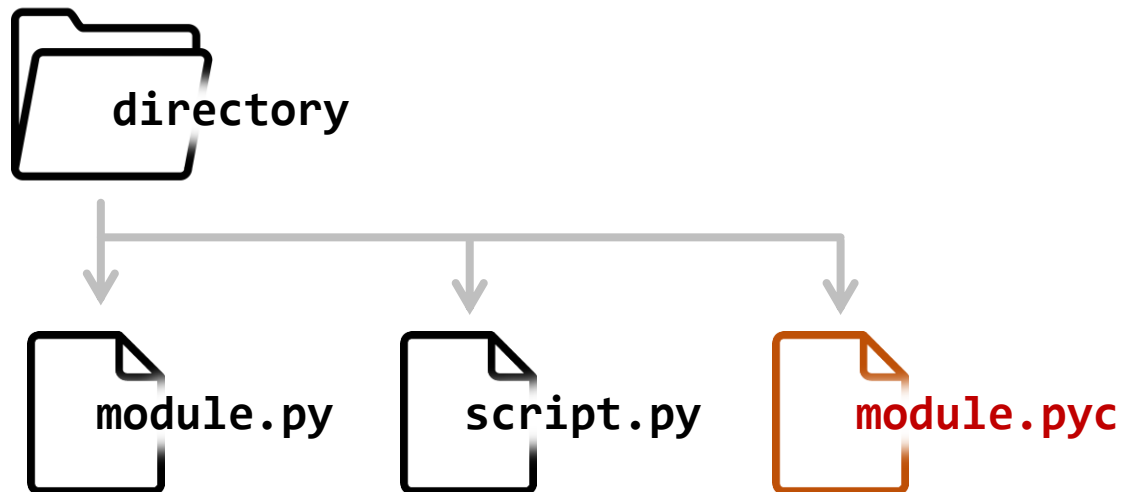
This module is just a  
Python script and can  
also be executed.

(a terminal)

```
$ python module.py
"Hello, World!"
```

# Module concepts

- You'll find a new file in the working directory called `module.pyc`
- This is “compiled” Python code
- Compiled code can be acted on more directly by the computer
- Rather than compiling “on the fly” (as with script-code), module code might not change as often, so we save time by keeping a compiled version





# Module concepts

- Not all modules live in your working directory
- Most of them live in the folder for your Python installation
  - e.g. `C:\Users\Franzosa\Anaconda3`
- New packages will be installed there too
  - e.g. `$ conda install scipy`
- You can also tell Python to look in other places for modules
  - Via the `$PYTHONPATH` environment variable
- The Python Standard Library + the Anaconda install from day 1 have all the modules you're likely to need for this course

# Module concepts

- Complex modules are organized in a nested structure
  - `scipy.stats.wilcoxon` is a function in the module `stats` in the module `scipy`
  - In reality, `scipy` would be a folder, `stats` would be a file within that folder, and `wilcoxon` would be a function in that file
- Modules may also be bundled with useful datasets
  - e.g. genetic code translation tables in the `biopython` module
- Python modules may also be bundled with compiled C code
  - For operations that need to be done very fast

Module help

# Module help

- Modules contain documentation that we can access directly, similar to using `man` on the command line.
- Within the Python interpreter (or a Jupyter Notebook), import a module, then execute `help( module_name )`

```
$ python
Python 3.6.5 | Anaconda, Inc.
Type "help", "copyright", "credits" or "license" for more information.
>>> import math
>>> help( math )
```

# Module help

- Modules contain documentation that we can access directly, similar to using `man` on the command line.
- Within the Python interpreter (or a Jupyter Notebook), import a module, then execute `help( module_name )`

```
Help on built-in module math:
```

```
NAME
```

```
    math
```

```
DESCRIPTION
```

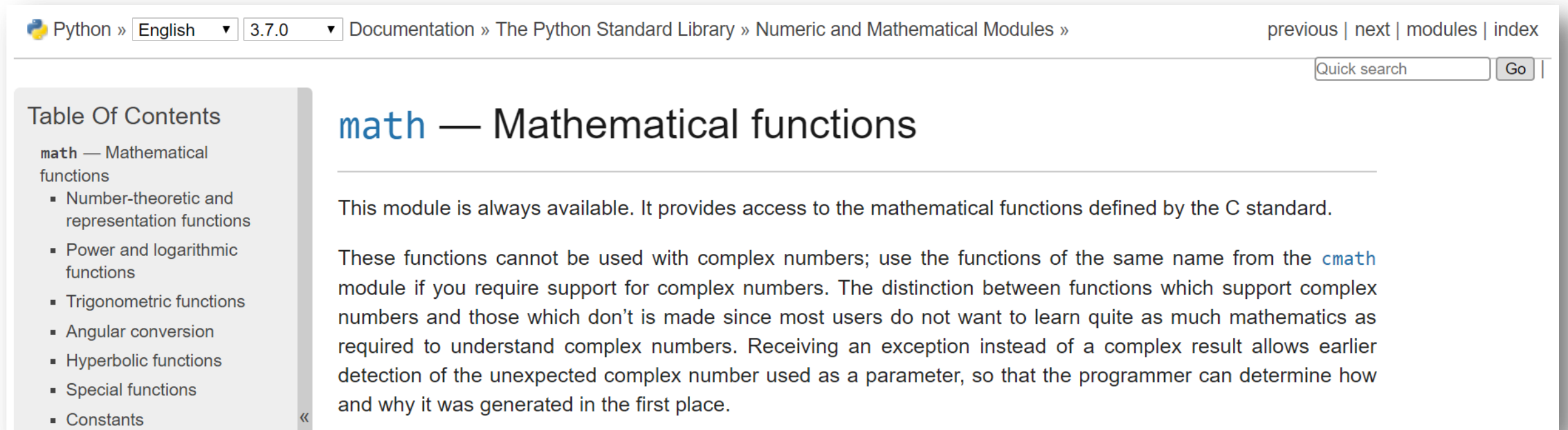
```
    This module is always available. It provides access to the  
    mathematical functions defined by the C standard.
```

```
FUNCTIONS
```

```
    acos(...)
```

# Module help

- Unlike **man** pages, I rarely consult **help( )** from the command line
- It is much more convenient to read about modules online:
  - <https://docs.python.org/3/library/math.html>
  - *(or Google “python module math” like I did)*



The screenshot shows the Python documentation website for the `math` module. The breadcrumb navigation at the top reads: Python » English » 3.7.0 » Documentation » The Python Standard Library » Numeric and Mathematical Modules ». On the right, there are links for 'previous', 'next', 'modules', and 'index', along with a 'Quick search' input field and a 'Go' button. On the left, a 'Table Of Contents' sidebar lists the contents of the `math` module: `math` — Mathematical functions, Number-theoretic and representation functions, Power and logarithmic functions, Trigonometric functions, Angular conversion, Hyperbolic functions, Special functions, and Constants. The main content area is titled '`math` — Mathematical functions' and contains two paragraphs: 'This module is always available. It provides access to the mathematical functions defined by the C standard.' and 'These functions cannot be used with complex numbers; use the functions of the same name from the `cmath` module if you require support for complex numbers. The distinction between functions which support complex numbers and those which don't is made since most users do not want to learn quite as much mathematics as required to understand complex numbers. Receiving an exception instead of a complex result allows earlier detection of the unexpected complex number used as a parameter, so that the programmer can determine how and why it was generated in the first place.'

# Module help

- Visit module help pages to 1) find a function that performs a data transformation that you need and 2) determine “how to use it.”
- “How to use it” = “What arguments does it take?” + “What does it return?”
- For example:

```
math.log(x[, base])
```

With one argument, return the natural logarithm of  $x$  (to base  $e$ ).

With two arguments, return the logarithm of  $x$  to the given *base*, calculated as `log(x)/log(base)`.

- In function help, `[ , ...]` indicates an optional positional argument.
  - *We'll get into the nuances of function arguments next week.*

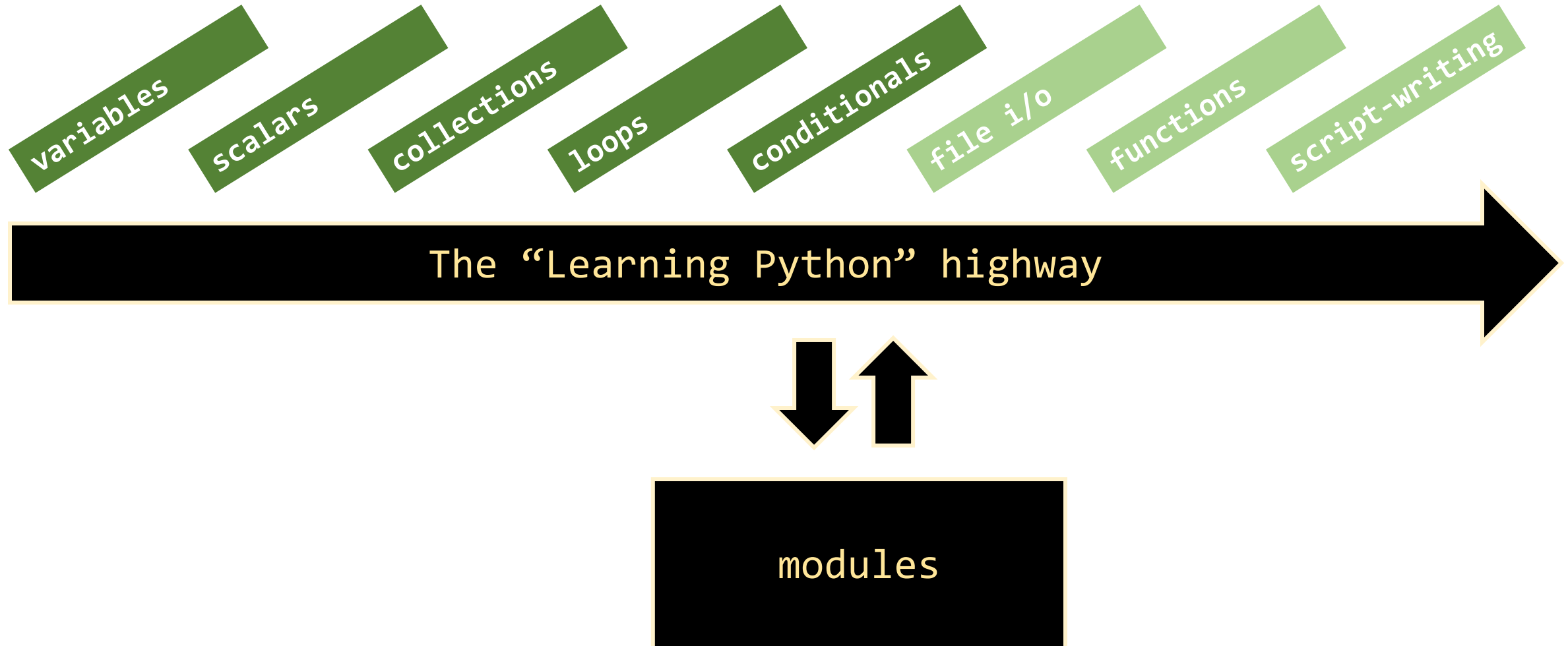
# Module help

- Starting next week, we will devote large chunks of individual lectures to exploring important Python modules
  - `argparse` for command-line interfaces
  - `re` for pattern matching
  - `numpy` for fast numerical computation
  - `subprocess` for calling other programs from Python
- You may end up needing/wanting to use other modules for final projects
- Unlike Python grammar (`for`, `if/else`, etc.), modules and their most useful components are a vocabulary you build up slowly over time



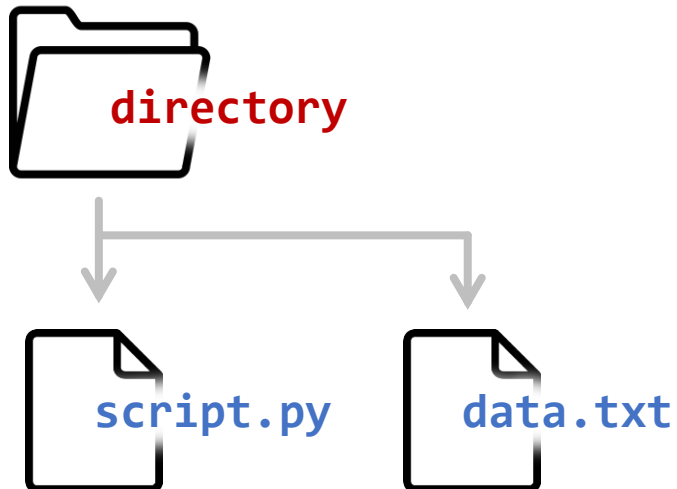
File I/O

# A little detour...



# File I/O and the `sys` module

- Python uses a special `file` data type to read data into a program
  - For when you get bored of looking through lists of numbers...
  - I'll refer to `files` as “file handles” to differentiate them from “files on your computer”
- We create `files` using the `open` function
  - `Open` takes a path to a file as a mandatory argument
- Let's look at an example



# File I/O and the sys module

`data.txt` (*open in Atom*)

```
Come gather 'round people  
Wherever you roam  
And admit that the waters  
Around you have grown  
And accept it that soon  
You'll be drenched to the bone.  
If your time to you  
Is worth savin'  
Then you better start swimmin'  
Or you'll sink like a stone  
For the times they are a-changin'.  
Come writers and critics  
Who prophesize with your pen  
And keep your eyes wide  
The chance won't come again  
And don't speak too soon  
For the wheel's still in spin  
And there's no tellin' who  
That it's namin'.  
For the loser now  
Will be later to win  
For the times they are a-changin'.
```

`script.py` (*open in Atom*)

```
fh = open( "data.txt" )
```

*open returns a file object  
that we store in a variable.  
I use "fh" as a convention  
(for "file handle").*

(*a terminal*)

```
$ python script.py
```

*No output yet*

# File I/O and the sys module

`data.txt` (*open in Atom*)

```
Come gather 'round people
Wherever you roam
And admit that the waters
Around you have grown
And accept it that soon
You'll be drenched to the bone.
If your time to you
Is worth savin'
Then you better start swimmin'
Or you'll sink like a stone
For the times they are a-changin'.
Come writers and critics
Who prophesize with your pen
And keep your eyes wide
The chance won't come again
And don't speak too soon
For the wheel's still in spin
And there's no tellin' who
That it's namin'.
For the loser now
Will be later to win
For the times they are a-changin'.
```

`script.py` (*open in Atom*)

```
fh = open( "data.txt" )
for line in fh:
    print( line )
```

Each iteration of the for loop gives us the next line of the file (stored here in the aptly named `line` variable).

(*a terminal*)

```
$ python script.py

Come gather 'round people

Wherever you roam

And admit that the waters

Around you have grown

And accept it that soon

You'll be drenched to the bone.

If your time to you

Is worth savin'

Then you better start swimmin'

...
```

# File I/O and the sys module

`data.txt` (*open in Atom*)

```
Come gather 'round people
Wherever you roam
And admit that the waters
Around you have grown
And accept it that soon
You'll be drenched to the bone.
If your time to you
Is worth savin'
Then you better start swimmin'
Or you'll sink like a stone
For the times they are a-changin'.
Come writers and critics
Who prophesize with your pen
And keep your eyes wide
The chance won't come again
And don't speak too soon
For the wheel's still in spin
And there's no tellin' who
That it's namin'.
For the loser now
Will be later to win
For the times they are a-changin'.
```

`script.py` (*open in Atom*)

```
fh = open( "data.txt" )
for line in fh:
    print( line )
```

Each line of the file ends  
with a newline (`\n`)  
character.

`print( )` adds its own  
newline by default,  
resulting in double spacing.

(*a terminal*)

```
$ python script.py

Come gather 'round people

Wherever you roam

And admit that the waters

Around you have grown

And accept it that soon

You'll be drenched to the bone.

If your time to you

Is worth savin'

Then you better start swimmin'

...
```

# File I/O and the sys module

`data.txt` (*open in Atom*)

```
Come gather 'round people
Wherever you roam
And admit that the waters
Around you have grown
And accept it that soon
You'll be drenched to the bone.
If your time to you
Is worth savin'
Then you better start swimmin'
Or you'll sink like a stone
For the times they are a-changin'.
Come writers and critics
Who prophesize with your pen
And keep your eyes wide
The chance won't come again
And don't speak too soon
For the wheel's still in spin
And there's no tellin' who
That it's namin'.
For the loser now
Will be later to win
For the times they are a-changin'.
```

`script.py` (*open in Atom*)

```
fh = open( "data.txt" )
for line in fh:
    line = line.strip( )
    print( line )
```

The string method  
`.strip( )` will remove  
white space (including  
newlines) at the beginning  
and end of a string, solving  
this problem.

(*a terminal*)

```
$ python script.py

Come gather 'round people
Wherever you roam
And admit that the waters
Around you have grown
And accept it that soon
You'll be drenched to the bone.
If your time to you
Is worth savin'
Then you better start swimmin'
...
```

# File I/O and the sys module

`data.txt` (*open in Atom*)

```
Come gather 'round people
Wherever you roam
And admit that the waters
Around you have grown
And accept it that soon
You'll be drenched to the bone.
If your time to you
Is worth savin'
Then you better start swimmin'
Or you'll sink like a stone
For the times they are a-changin'.
Come writers and critics
Who prophesize with your pen
And keep your eyes wide
The chance won't come again
And don't speak too soon
For the wheel's still in spin
And there's no tellin' who
That it's namin'.
For the loser now
Will be later to win
For the times they are a-changin'.
```

`script.py` (*open in Atom*)

```
fh = open( "data.txt" )
for line in fh:
    print( line, end="" )
```

Another solution

(*a terminal*)

```
$ python script.py

Come gather 'round people
Wherever you roam
And admit that the waters
Around you have grown
And accept it that soon
You'll be drenched to the bone.
If your time to you
Is worth savin'
Then you better start swimmin'
...
```



# File I/O and the sys module

`data.txt` (open in Atom)

```
Come gather 'round people
Wherever you roam
And admit that the waters
Around you have grown
And accept it that soon
You'll be drenched to the bone.
If your time to you
Is worth savin'
Then you better start swimmin'
Or you'll sink like a stone
For the times they are a-changin'.
Come writers and critics
Who prophesize with your pen
And keep your eyes wide
The chance won't come again
And don't speak too soon
For the wheel's still in spin
And there's no tellin' who
That it's namin'.
For the loser now
Will be later to win
For the times they are a-changin'.
```

`script.py` (open in Atom)

```
fh = open( "data.txt" )
for line in fh:
    line = line.strip( )
    print( line )
fh.close( )
```

It is a good convention to  
close a file when we are  
done with it.

(a terminal)

```
$ python script.py

Come gather 'round people
Wherever you roam
And admit that the waters
Around you have grown
And accept it that soon
You'll be drenched to the bone.
If your time to you
Is worth savin'
Then you better start swimmin'
...
```

# File I/O and the sys module

`data.txt` (open in Atom)

```
Come gather 'round people
Wherever you roam
And admit that the waters
Around you have grown
And accept it that soon
You'll be drenched to the bone.
If your time to you
Is worth savin'
Then you better start swimmin'
Or you'll sink like a stone
For the times they are a-changin'.
Come writers and critics
Who prophesize with your pen
And keep your eyes wide
The chance won't come again
And don't speak too soon
For the wheel's still in spin
And there's no tellin' who
That it's namin'.
For the loser now
Will be later to win
For the times they are a-changin'.
```

`script.py` (open in Atom)

```
import sys

print( sys.argv )

fh = open( "data.txt" )
for line in fh:
    line = line.strip( )
    print( line )
fh.close( )
```

The `sys` module (among other things) provides us with a list of arguments passed on the command line: `sys.argv`

(a terminal)

```
$ python script.py x y z
['script.py', 'x', 'y', 'z']
```

The first element of `sys.argv`, in other words `sys.argv[0]`, is the name of the script that is running

# File I/O and the sys module



`data.txt` (*open in Atom*)

```
Come gather 'round people
Wherever you roam
And admit that the waters
Around you have grown
And accept it that soon
You'll be drenched to the bone.
If your time to you
Is worth savin'
Then you better start swimmin'
Or you'll sink like a stone
For the times they are a-changin'.
Come writers and critics
Who prophesize with your pen
And keep your eyes wide
The chance won't come again
And don't speak too soon
For the wheel's still in spin
And there's no tellin' who
That it's namin'.
For the loser now
Will be later to win
For the times they are a-changin'.
```

`script.py` (*open in Atom*)

```
import sys

fh = open( sys.argv[1] )
for line in fh:
    line = line.strip( )
    print( line )
fh.close( )
```

This is a better way to pass a  
file into a program than  
“hard-coding” the path.

**Can you guess why?**

(*a terminal*)

```
$ python script.py data.txt

Come gather 'round people
Wherever you roam
And admit that the waters
Around you have grown
And accept it that soon
You'll be drenched to the bone.
If your time to you
Is worth savin'
Then you better start swimmin'
...
```

# File I/O and the sys module

`data.txt` (open in Atom)

```
Come gather 'round people
Wherever you roam
And admit that the waters
Around you have grown
And accept it that soon
You'll be drenched to the bone.
If your time to you
Is worth savin'
Then you better start swimmin'
Or you'll sink like a stone
For the times they are a-changin'.
Come writers and critics
Who prophesize with your pen
And keep your eyes wide
The chance won't come again
And don't speak too soon
For the wheel's still in spin
And there's no tellin' who
That it's namin'.
For the loser now
Will be later to win
For the times they are a-changin'.
```

`script.py` (open in Atom)

```
import sys

for line in sys.stdin:
    line = line.strip( )
    print( line )
```

`sys.stdin` is a file handle  
that returns information  
piped into the script. We  
read it just like a normal file.

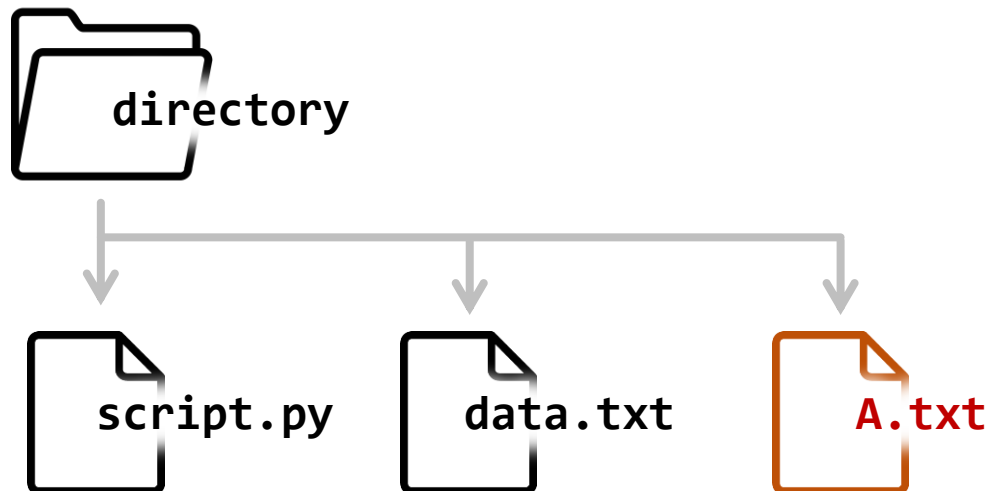
(a terminal)

```
$ cat data.txt | python script.py

Come gather 'round people
Wherever you roam
And admit that the waters
Around you have grown
And accept it that soon
You'll be drenched to the bone.
If your time to you
Is worth savin'
Then you better start swimmin'
...
```

# File I/O and the `sys` module

- We can also use `open( )` to create file handles for writing new files
- The syntax for this is `open( file_name, "w" )`
  - *!!! Warning: this will overwrite file\_name if it already exists !!!*
- We can write lines to such file handles using `print( )`
  - By default, `print( )` writes to `sys.stdout`, i.e. the terminal itself



# File I/O and the sys module

*data.txt (open in Atom)*

```
Come gather 'round people
Wherever you roam
And admit that the waters
Around you have grown
And accept it that soon
You'll be drenched to the bone.
If your time to you
Is worth savin'
Then you better start swimmin'
Or you'll sink like a stone
For the times they are a-changin'.
Come writers and critics
Who prophesize with your pen
And keep your eyes wide
The chance won't come again
And don't speak too soon
For the wheel's still in spin
And there's no tellin' who
That it's namin'.
For the loser now
Will be later to win
For the times they are a-changin'.
```

*script.py (open in Atom)*

```
import sys

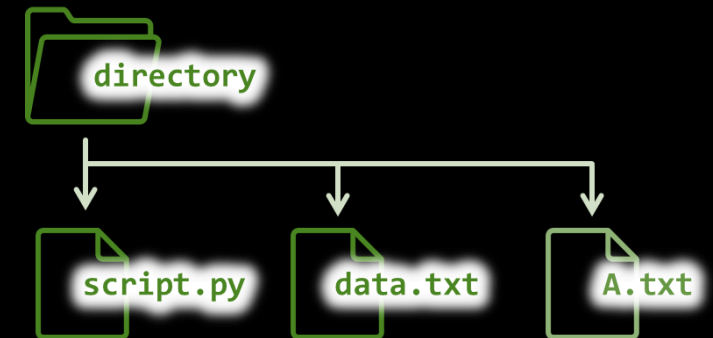
fh = open( "A.txt", "w" )
for line in sys.stdin:
    line = line.strip( )
    if line[0] == "A":
        print( line, file=fh )
fh.close( )
```

We use the optional "file=" argument of print to tell the function where to put the data.

*(a terminal)*

```
$ cat data.txt | python script.py
$ cat A.txt
```

```
And admit that the waters
Around you have grown
And accept it that soon
And keep your eyes wide
And don't speak too soon
And there's no tellin' who
```



# In perspective

- This is very powerful
- We now have the ability to read a data file into a program, operate on that data, and then write some results as a new file.
- We can use this to make programs that behave like command-line tools:
  - Solve problems on their own...
  - ...or as part of command-line chains
- Actually, that sounds like a good idea...

Activity: `grep.py`



# grep.py

- Write a Python script called `grep.py`
- The script should take two positional, command-line arguments:
  - `$ python grep.py pattern file_name`
  - The 1<sup>st</sup> argument (*pattern*) is a string to search for
  - The 2<sup>nd</sup> argument (*file\_name*) is a path to a file you want to search
- The script should print (to the screen) lines from the file that contain the matching pattern
- An example execution might look like:
  - `$ python grep.py "e" data.txt`

# grep.py



- Compose your script in a plain-text editor (such as Atom)
- Be careful about copying and pasting code directly from the slides
  - The text sometimes has extra formatting embedded that can cause errors
  - Better to retype the commands you want (good for muscle memory too!)
- Other suggestions:
  - Use `sys.argv[1]` and `sys.argv[2]` to access the pattern and file name
  - Use the `in` operator to see if a pattern occurs in a line
  - The code from the slide with the star (like the one above) is a good starting point
- Alternatively...
  - Download my `grep( )` function from Canvas and import it into your script
  - You'll still use `sys.argv[1]` and `sys.argv[2]` to get the pattern and file name
  - Canvas also contains a `data.txt` file to experiment with

(a quick word about...)

# Operator Precedence

# Python operator precedence

Higher precedence / Tighter binding

( )

\*\*

\*, /, //, %

+, -

in, <, <=, >, >=, !=, ==

not

and

or

Lower precedence / Weaker binding

Tied precedence (same row)  
evaluated left to right

`(1+1)**2 < 4 or 1 + 1 * 2 == 3 and not "i" in "team"`

`2**2 < 4 or 1 + 1 * 2 == 3 and not "i" in "team"`

`4 < 4 or 1 + 1 * 2 == 3 and not "i" in "team"`

`4 < 4 or 1 + 2 == 3 and not "i" in "team"`

`4 < 4 or 3 == 3 and not "i" in "team"`

`False or True and not False`

`False or True and True`

`False or True`

True

# Python operator precedence

Higher precedence / Tighter binding

( )

\*\*

\*, /, //, %

+, -

in, <, <=, >, >=, !=, ==

not

and

or

Lower precedence / Weaker binding

Tied precedence (same row)  
evaluated left to right

When in doubt, put  
parentheses around  
expressions that you want  
to evaluate earlier!

True