Lecture Five

Putting it together

Agenda

- Reiteration of goals
- Exercise from last time
- Functions
- Useful modules
- Actually writing a script

"Le talent est une longue patience..."

Last week's example

- We set a very challenging example last week
    - It'll make for a nice transition into today's topics
    - So let's have a look

Last week's example

```
>>> with open('file1.txt') as f:
...     filelist = [line.strip('\n').split('\t') for line in f]
...
>>> Birdlist = []
>>> Bearlist = []
>>> Beelist = []
```

Last week's example

```
>>> for item in filelist:
...     if item[0] == 'Bear':
...             Bearlist.append(int(item[2]))
...     elif item[0] == 'Bird':
...             Birdlist.append(int(item[2]))
...     elif item[0] == 'Bees':
...             Beelist.append(int(item[2]))
...
```

Last week's example

```
>>> Bearcount = Bearlist[0] + Bearlist[1]
>>> Beescount = Beeslist[0] + Beeslist[1]
>>> Birdcount = Birdlist[0] + Birdlist[1]
>>> OrgCount = {}
```

Last week's example

```
>>> for item in filelist:
...     if item[0] == 'Bear':
...             OrgCount[item[0]]= Bearcount
...     elif item[0] == 'Bird':
...             OrgCount[item[0]]= Birdcount
...     elif item[0] == 'Bees':
...             OrgCount[item[0]]= Beescount
...
```

Last week's example

- This is functional.
- But clunky and inflexible
- Today, we'll talk about some ways to take that code, streamline it a bit and make it more functional and versatile

Last week's example

- We're going to start out talking about functions.

Last week's example

- We're going to start out talking about functions.
- A function is what it sounds like: a chunk of code that does some task

Last week's example

- We're going to start out talking about functions.
- A function is what it sounds like: a chunk of code that does some task
- They are objects that can be *called* by name or assigned to a variable
  - *variable* = *function*()

Last week's example

- When you think about it, there are three main parts in the code from last week

Last week's example

- When you think about it, there are three main parts in the code from last week
    - Opening the file and processing it

Last week's example

- When you think about it, there are three main parts in the code from last week
  - Opening the file and processing it
  - Make our animal:observations dictionary

Last week's example

- When you think about it, there are three main parts in the code from last week
  - Opening the file and processing it
  - Make our animal:observations dictionary
  - Print it out so we can see

Functions

- Functions allow us to make this code more streamlined, modular, and readable.

Functions

- Try to make your function execute one task.

Functions

- Try to make your function execute one task.
  - It's hard to do this
  - Each task should be self contained, yet flexible

Functions

- Try to make your function execute one task.
  - It's hard to do this
  - Each task should be self contained, yet flexible

- Write out the steps you think your code should follow
  - "Open and parse file into a list"
  - "Loop over list and extract x, y, but not z"
  - etc etc...

Functions

- A function is defined by the user with a 'def' statement.

  ```
  def function(parameter list):
      code to be executed
  ```

Functions

- A function is defined by the user with a 'def' statement.

```
def function(parameter list):
    code to be executed
```

- *parameter list* is a comma delimited series of objects you wish to *pass* to the function.

```
def function(file):
    do something with file
```

Functions

- A function definition needs to precede a call to the function

Functions

```
>>> def function():  # function definition
...         print "hurray!"
>>> function()  # function call
"hurray!"
```

The 'return' statement

- Some functions just print something

- But most of the time, you want a function to give you value

The 'return' statement

- Some functions just print something

- But most of the time, you want a function to give you value

- A 'return' statement allows this
    - It also exits the function

The 'return' statement

- General form:

```
def function_name():
    do something
    return value
```

From last week

```
def opener(infile):
    with open(infile) as f:
        return [line.strip('\n').split('\t') for line in f]
```

From last week

```
def opener(infile):
    with open(infile) as f:
        return [line.strip('\n').split('\t') for line in f]
```

- When the function is executed, the data in the list comprehension is held in memory.

From last week

```
def opener(infile):
    with open(infile) as f:
        return [line.strip('\n').split('\t') for line in f]
```

- When the function is executed, the data in the list comprehension is held in memory.

- You can assign it to a variable to access it.

```
>>> file_list = opener(infile)
```

Docstrings

- Functions have a special type of comment called a docstring
  - These are not invisible to Python, like comments
  - They can be accessed with help()

Docstrings

- Functions have a special type of comment called a docstring
    - These are not invisible to Python, like comments
    - They can be accessed with help()

```
>>> def hurray():
...     '''Prints hurray!'''   # Docstring
...     print 'hurray!'
```

Docstrings

- Functions have a special type of comment called a docstring
    - These are not invisible to Python, like comments
    - They can be accessed with help()

```
>>> def hurray():
...      '''Prints hurray!'''   # Docstring
...      print 'hurray!'
>>> help(hurray)
hurray()
    Prints hurray  # Now you know!
```

Organizing Functions

- The hardest part...

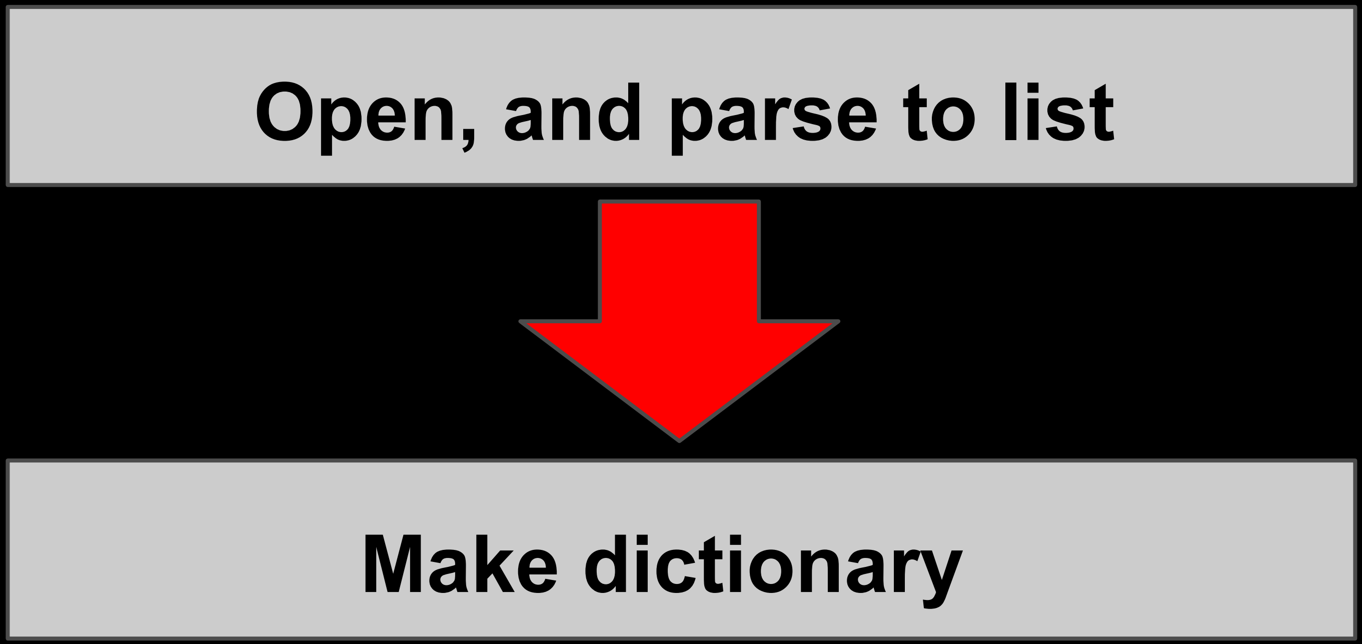
Organizing Functions

- Let's think about our code from last week

Open, and parse to list

Organizing Functions

- Let's think about our code from last week

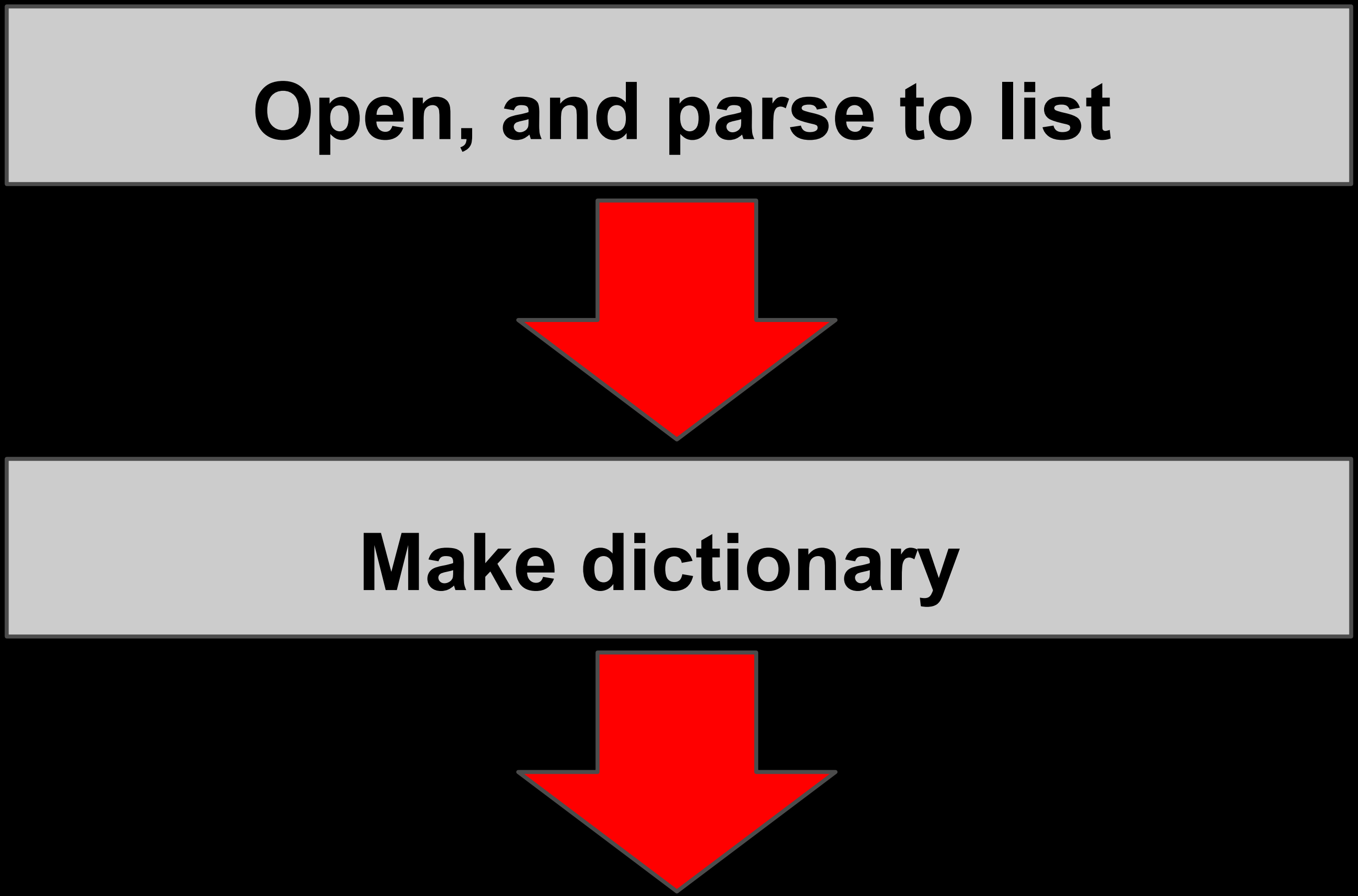


Open, and parse to list

Make dictionary

Organizing Functions

- Let's think about our code from last week



Open, and parse to list

Make dictionary

Organizing Functions

- We want to take output from one function and use it in another.

Organizing Functions

- We want to take output from one function and use it in another.

- How does one function access the data from another?

Organizing Functions

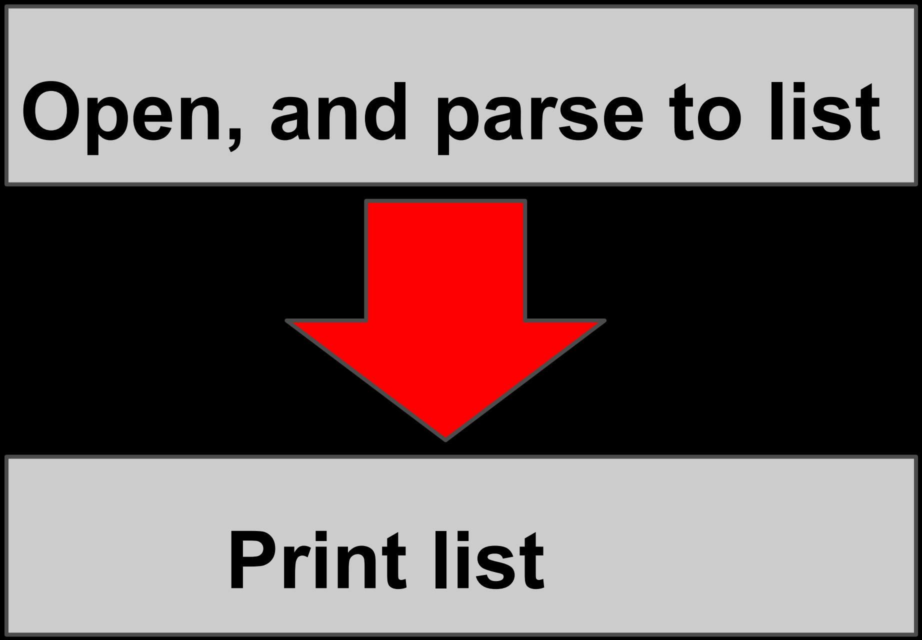
- We want to take output from one function and use it in another.

- How does one function access the data from another?

- What about variables? Can one function access the variables in another?

Organizing Functions

- Simpler example



Open, and parse to list

Print list

Organizing Functions

```
def opener(infile):
    with open(infile) as f:
        my_list=[line.strip('\n').split('\t') for line in f]

def print_list():
    print my_list  # Kosher??
```

Organizing Functions

```
def opener(infile):
    with open(infile) as f:
        my_list=[line.strip('\n').split('\t') for line in f]

def print_list():
    print my_list  # Kosher??
```

Nope! Variables in function have local scope, just like in Unix. 'my_list' has no *meaning* within obs_dict()

Organizing Functions

```
def opener(infile):
    with open(infile) as f:
        my_list=[line.strip('\n').split('\t') for line in f]
        return my_list

def print_list():
    my_list  = opener('my_file.txt')
    print my_list
```

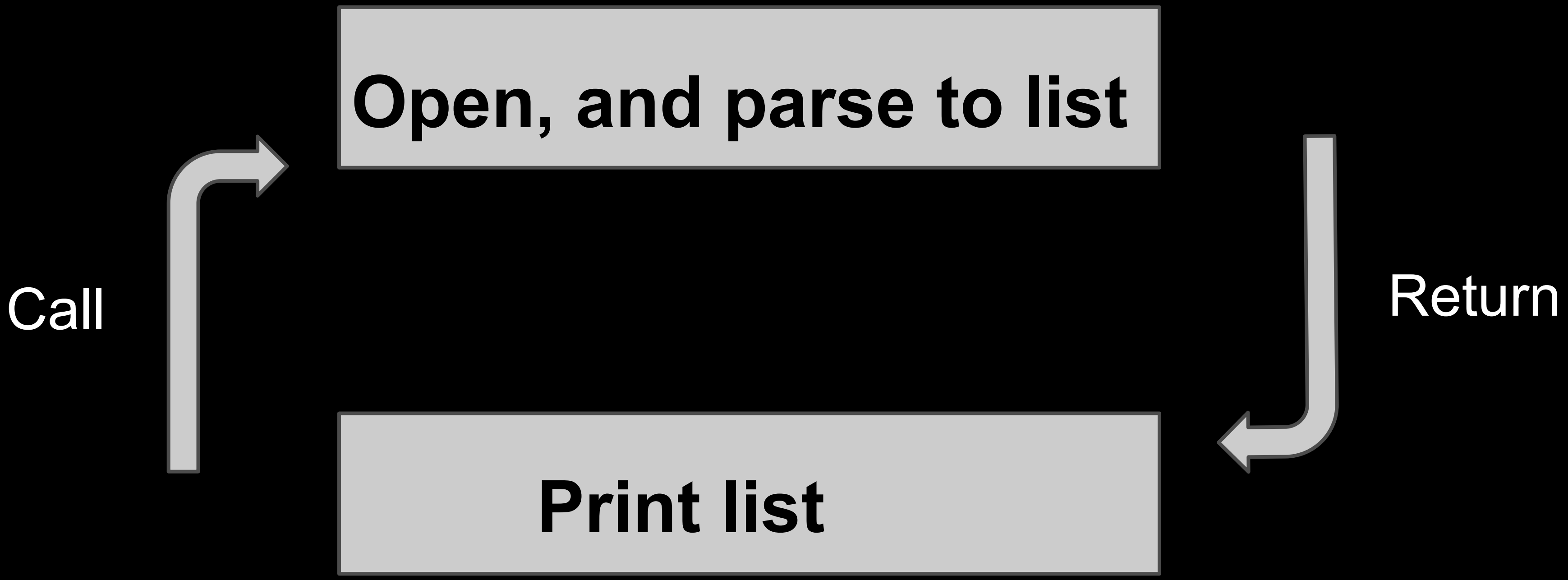
Organizing Functions

- How is this different?

- We put a 'return' statement in opener(), and a *call* to opener() in print_list()

Organizing Functions

- Clear as mud??



Open, and parse to list

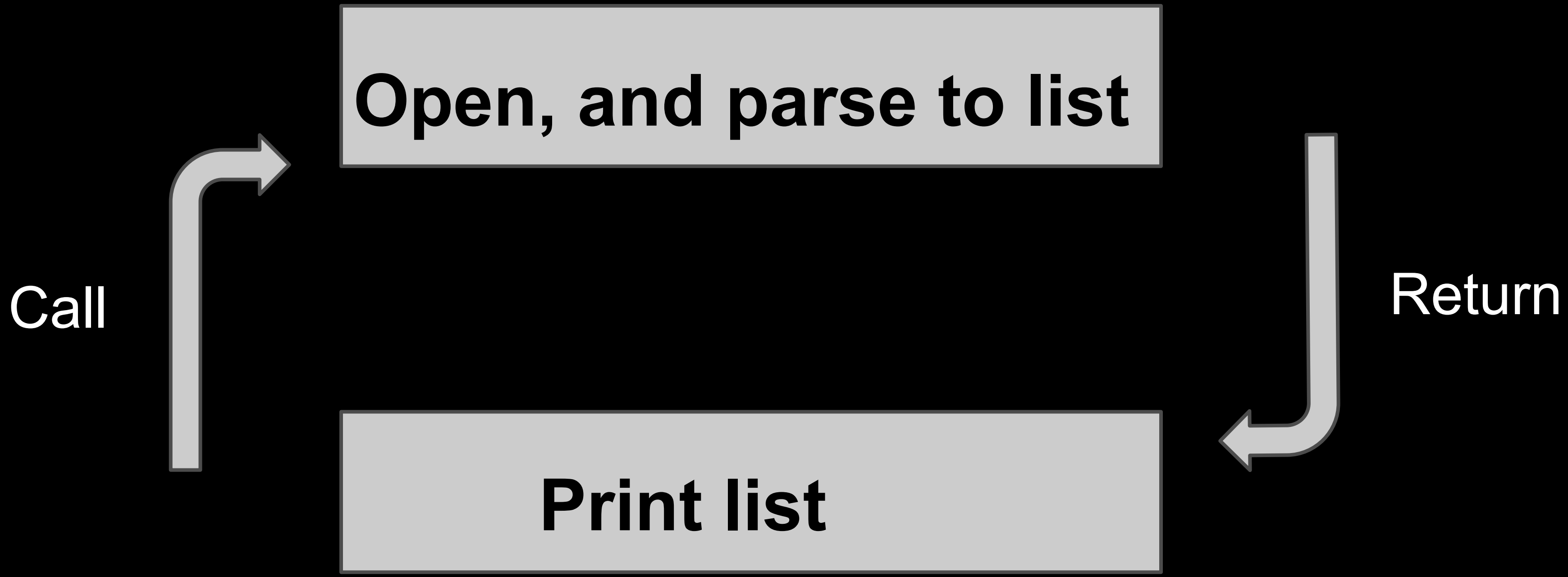
Call

Return

Print list

Organizing Functions

- Clear as mud??



Open, and parse to list

Call

Return

Print list

- Calls: (backward, up),
- Returns: (feed forward, down)

Program Flow

- Ideally, programs are cascading sets of functions that are not hard-coded

Program Flow

- Ideally, programs are cascading sets of functions that are not hard-coded
  - It's pretty easy to make a variable global and not worry about passing the variables around
  - Ideally, your functions should map cleanly to pseudocode. So, thinking from the ground-up in terms of functions can help you start to tackle a monumental task.

Program flow

- Open file and make a list of the contents of each line – strip ‘\n’s and split each line on ‘\t’.
- This is opener() in the functionized script

Program flow

- Loop through lines and add up observations for each animal
  - Dictionary, add it as key with count as value, if the key is in the dict, add count to current value in dictionary.
- obs_dictionary()

Program flow

- Print observations - print organism and count.
- print_obs()

sys.argv

- What a weird name.
  - What's going on here?

sys.argv

- What a weird name.
  - What's going on here?
- Writing scripts that accept input from the command line can be a good way to avoid what is called 'hard coding'

Hard Coding

- Hard coding is a coding method that requires the course code (the original script) to be changed whenever desired output is changed.

Hard Coding

- Example:

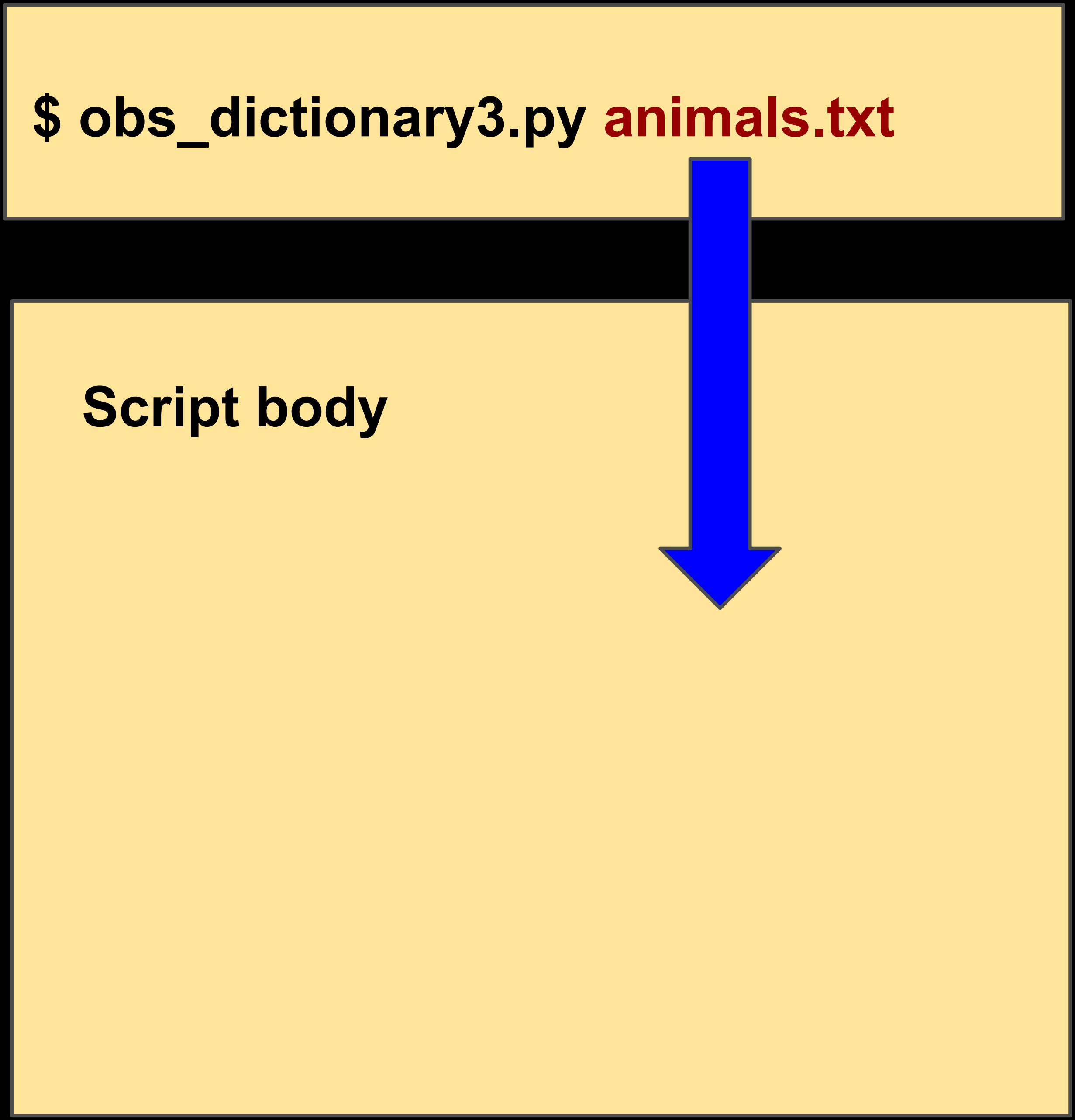
```
>>> with open('animals.txt') as file:
...         file_list = [line.strip("\n") for line in file
```

- We call this hard coding because if you want to perform the strip operation on a different file, you have to alter your script.

Hard Coding

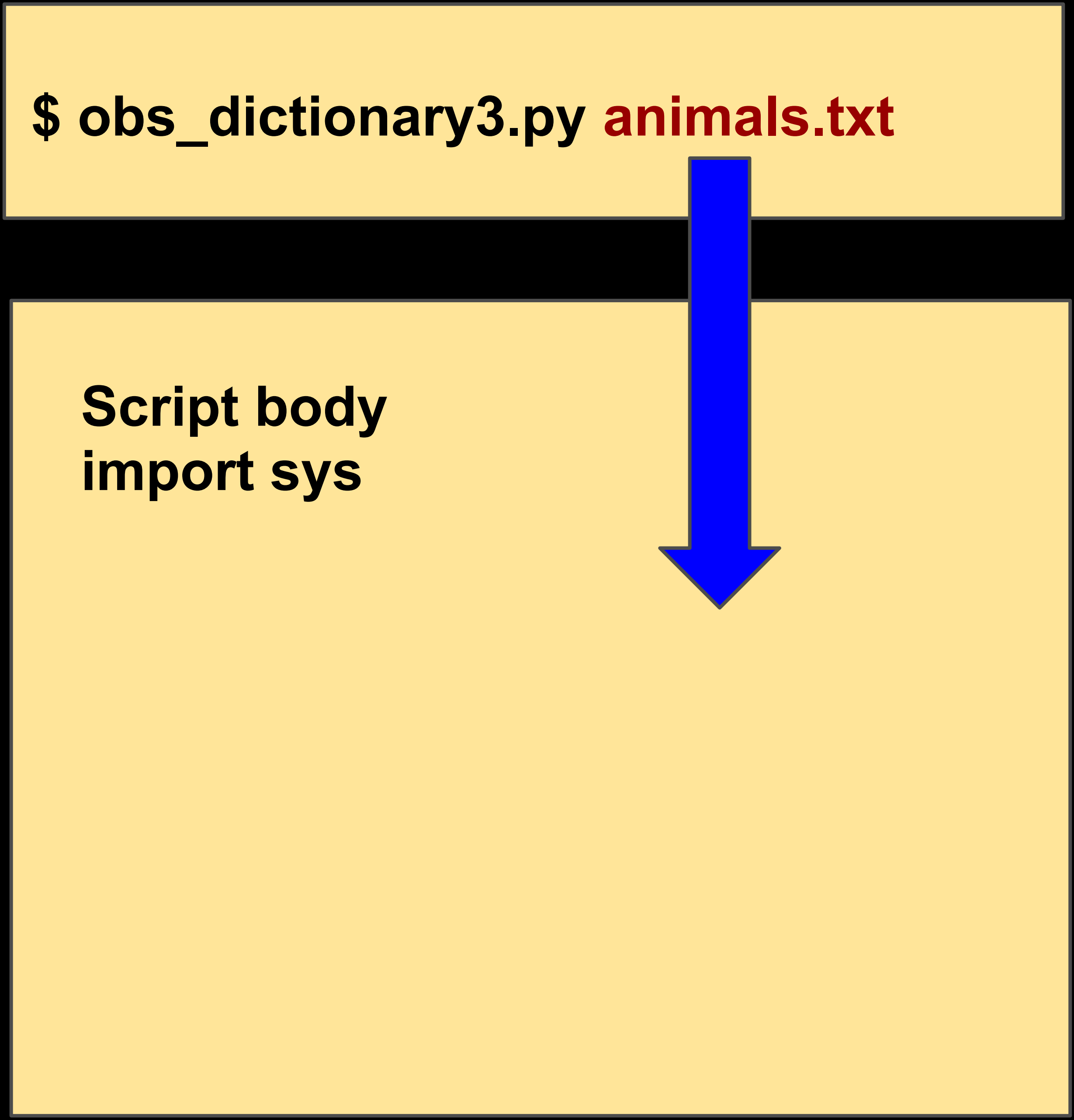
- As we saw last week when you were writing functions, hard coding can work
- But, having applications be flexible to input can make your code more user-friendly and increase your chances of being cited.

```
$ obs_dictionary3.py animals.tx
$ obs_dictionary3.py animals.txt
$ obs_dictionary3.py animals.txt
```



```
$ obs_dictionary3.py animals.txt
```

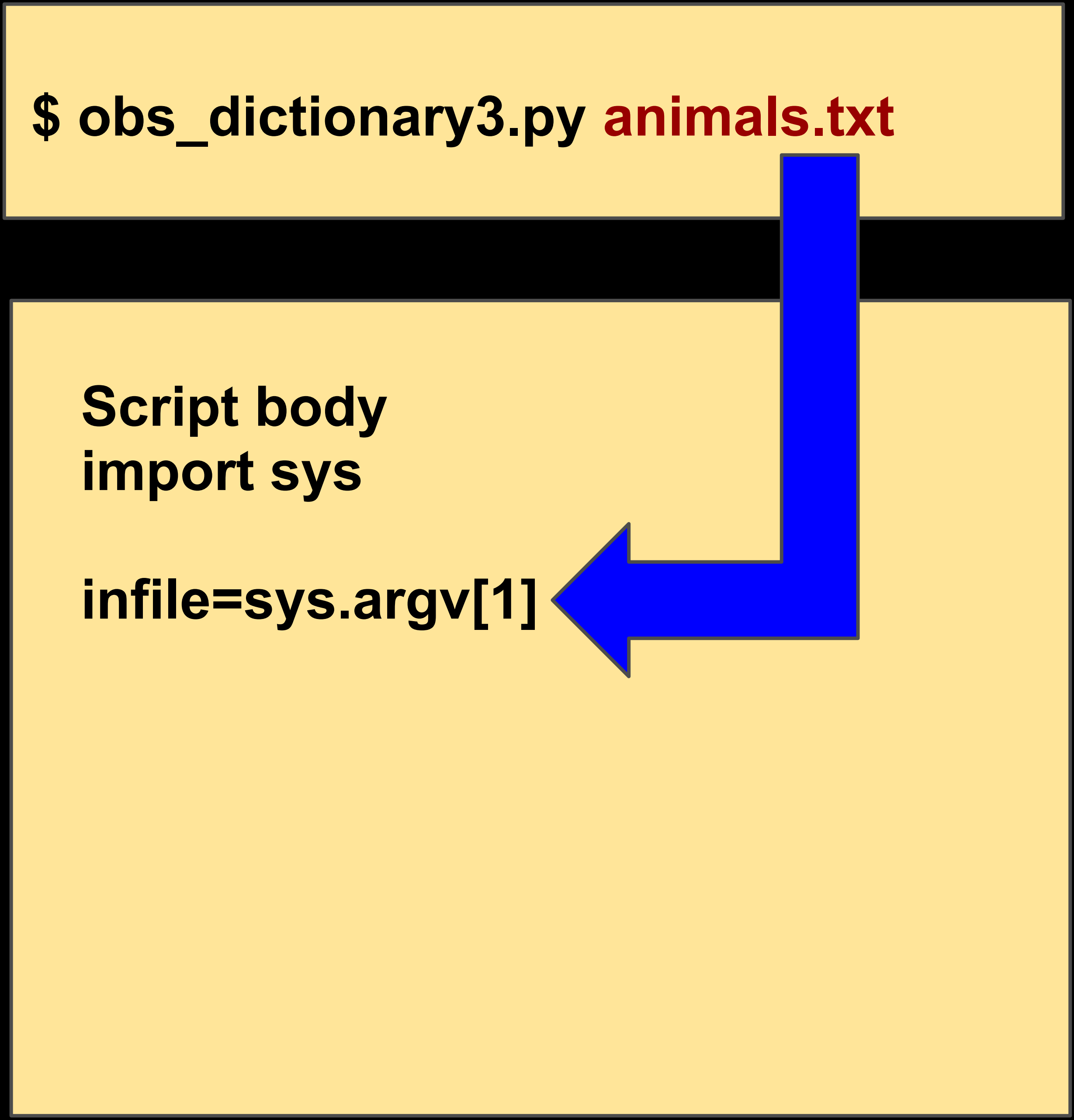
Script body



```
$ obs_dictionary3.py animals.txt
```

Script body

```
import sys
```



```
$ obs_dictionary3.py animals.txt
```

Script body

```
import sys

infile=sys.argv[1]
```

sys.argv

- sys.argv in Python allows the coder to pass input from the command line into the code
- In the "functionized" script of last week's exercise, you will see a line of code that says

```
import sys
infile = sys.argv[1]
```

- This is importing the sys module (more on this in a moment) and setting the variable "infile" as the first argument passed from the

what

- sys.argv takes input from the command line.
    - You can feed the module multiple pieces of information.
    - In this case, as you might have guessed, we want to input a file

```
>>> python obs_counter3.py animals.txt
```

- In this case, the information being passed into the program is the filename animals.txt

what

- In this case, the information being passed into the program is the filename animals.txt
- 'animals.txt' is then passed to this line:

```
>>> infile = sys.argv[1]
```

- in the script body

what

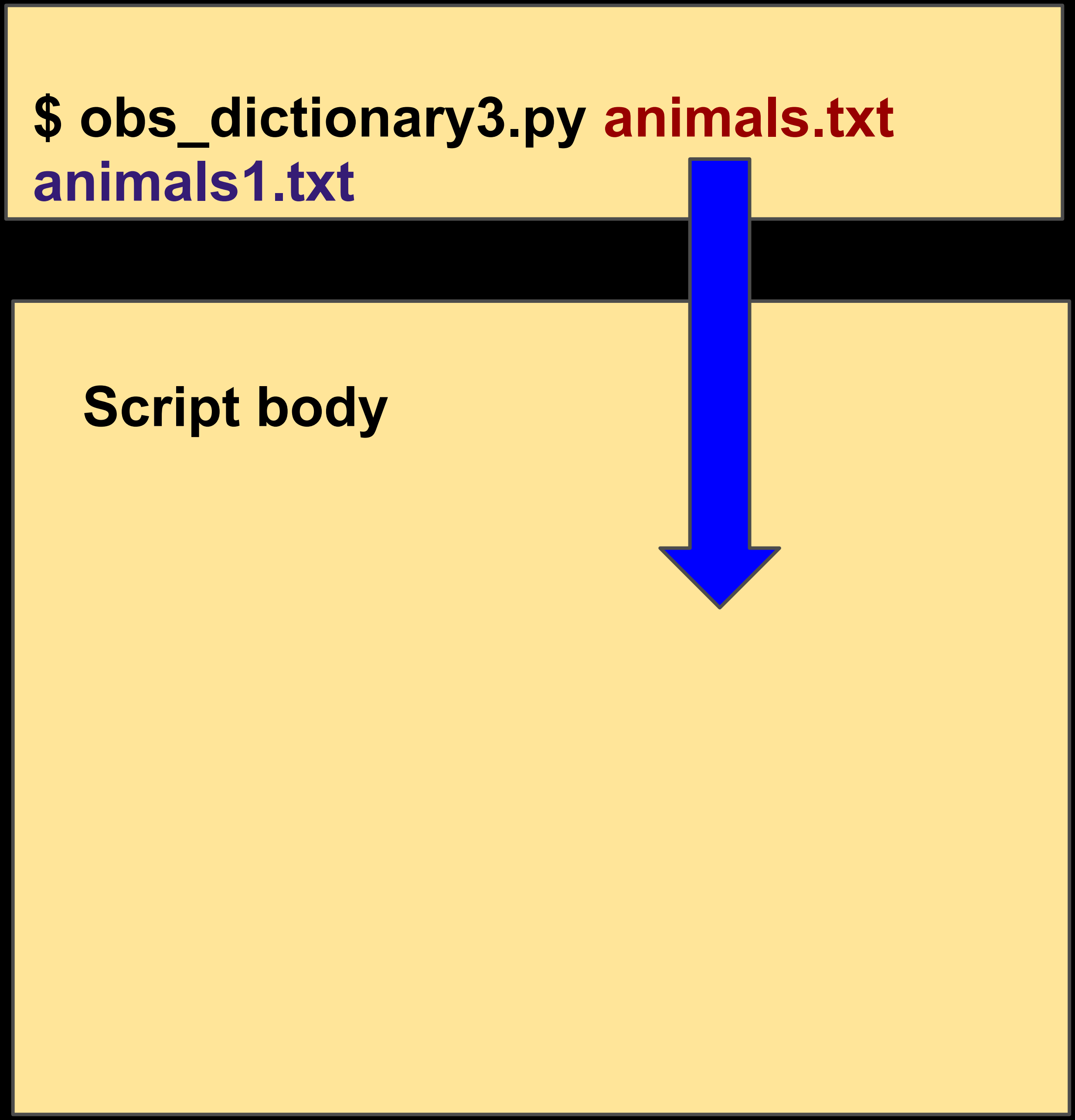
- 'animals.txt' is then passed to this line:

```
>>> infile = sys.argv[1]
```

- in the script body
- This line parses the command line input as the variable infile
- The one means the first argument provided.

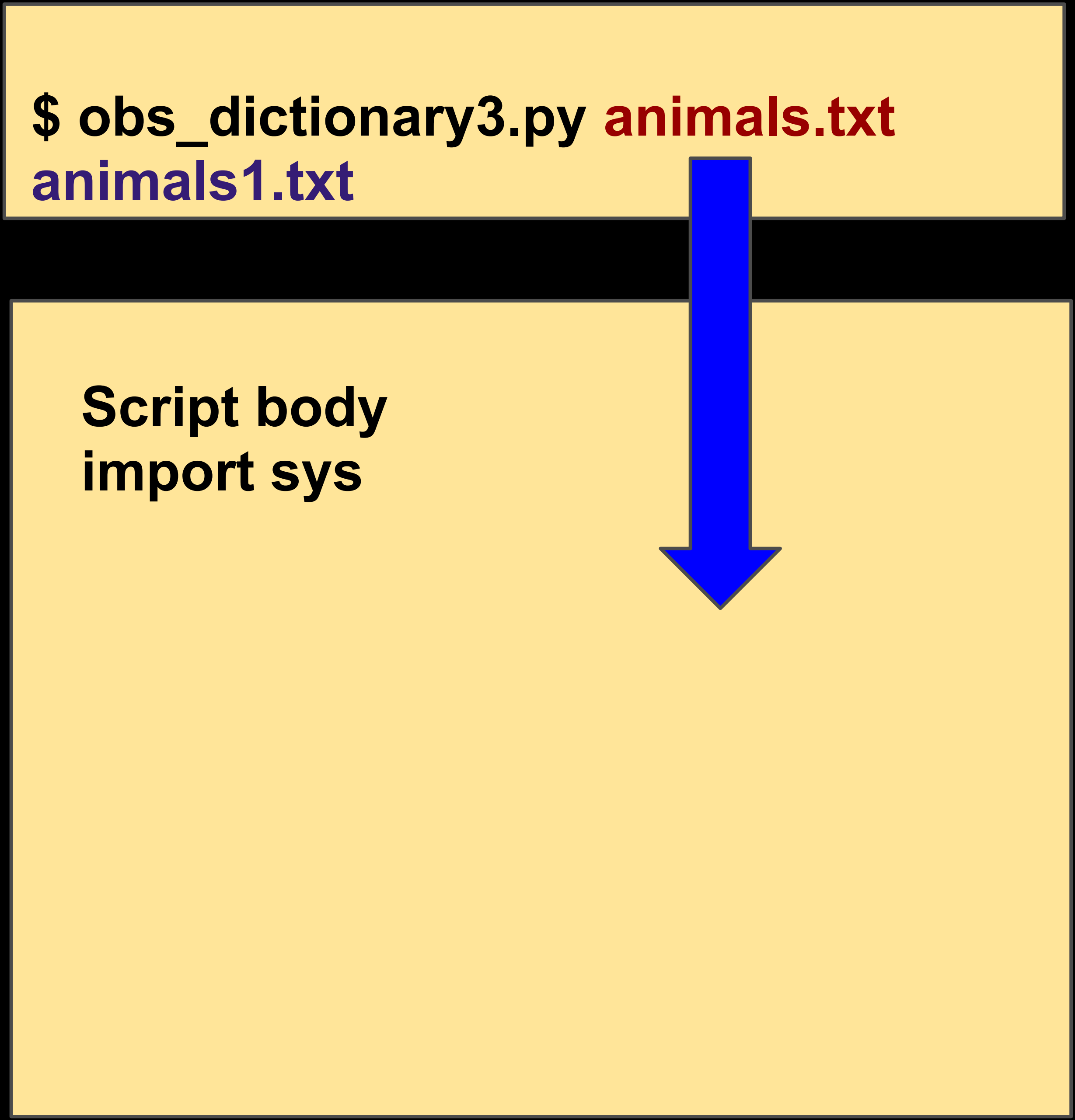
```
$ obs_dictionary3.py animals.txt animals1.txt
$ obs_dictionary3.py animals.txt
animals1.txt
```

```
$ obs_dictionary3.py animals.txt
animals1.txt
```



```
$ obs_dictionary3.py animals.txt animals1.txt
```

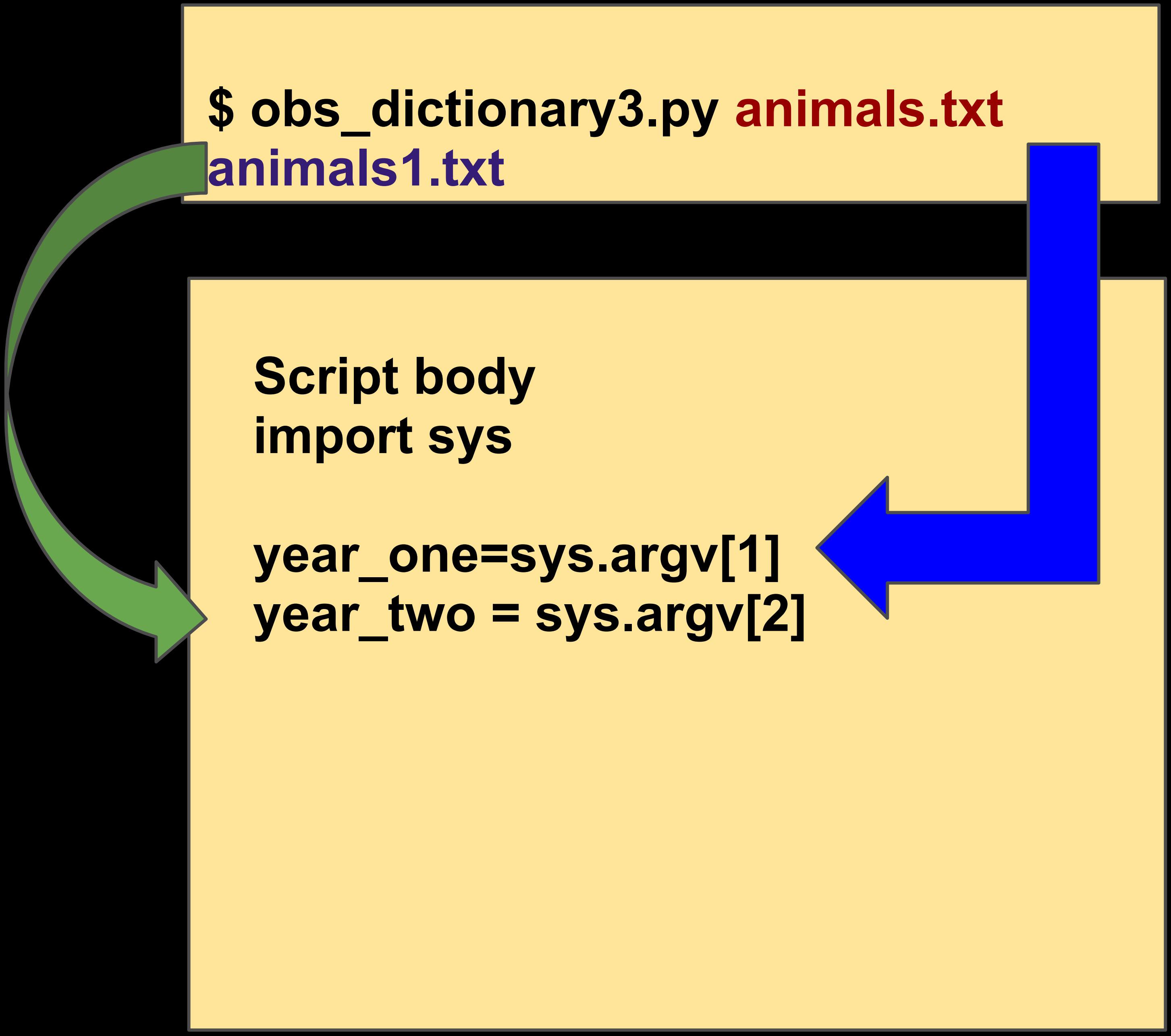
Script body



```
$ obs_dictionary3.py animals.txt animals1.txt
```

Script body

```
import sys
```



```
$ obs_dictionary3.py animals.txt
animals1.txt
```

Script body

```
import sys

year_one=sys.argv[1]
year_two = sys.argv[2]
```

Program Flow

- Ideally, programs are cascading sets of functions that are not hard-coded
  - When you're structuring a program, it's important to think about who will use the program. Why will they use it? How can you make the program more flexible?

Program Flow

- Ideally, programs are cascading sets of functions that are not hard-coded
  - When you're structuring a program, it's important to think about who will use the program. Why will they use it? How can you make the program more flexible?
  - Our opener() function can use sys.argv[]

raw_input()

- We talked about sys_argv[]
- What if you want to have someone input some value for a calculation
- Python has a function for this called raw_input()
- This will take in a value that can be interacted with by a script

raw_input()

-
  ```
  >>> a = raw_input('Please enter a number here: ')
  >>> print a
  ```

raw_input()

-
  ```
  >>> a = raw_input('Please enter a number here: ')
  >>> print a
  ```

```
Please enter a number here:
```

raw_input()

-
  ```
  >>> a = raw_input('Please enter a number here: ')
   >>> print a
  Please enter a number here: 12
  ```

raw_input()

-
  ```
  >>> a = raw_input('Please enter a number here: ')
   >>> print a
  ```

```
Please enter a number here: 12
12
```

raw_input()

- So what happened here?
  - Python read the raw_input call and prompted you to enter some information
  - Python read this information and did what you said to do with it
    - Print, in this case
  - But you could do pretty much any other operation

raw_input()

- What if I had entered a letter?
  - raw_input would have accepted it
  - This is why it's helpful to have text that tells the user what to put in

Wrapping it up and putting a bow on it

- Some further considerations in programming.

Wrapping it up and putting a bow on it

- The shebang
  - If you looked at any of the scripts we posted over the past couple weeks, you might have noticed this line:
  - #! /usr/bin/env python
  - #! is denoting these as the shebang line
  - The rest of the line is invoking Python and telling the interpreter to run commands in the Python subshell
  - This should be the first line in your Python script

Wrapping it up and putting a bow on it

- When do you want to write to a file versus to the standard output?

Wrapping it up and putting a bow on it

- When do you want to write to a file versus to the standard output?
  - Standard out is great for including print statements to do error checking
  - Also for passing output to other programs or scripts

Wrapping it up and putting a bow on it

- When do you want to write to a file versus to the standard output?
    - Standard out is great for including print statements to do error checking
    - Also for passing output to other programs or scripts
    - Writing to a file is great if you need to run part of your script in one location and part in another
        - Generate data file on desktop, Run on TACC
    - Temporal separation of steps.
    - Import to R.

Wrapping it up and putting a bow on it

- When do you want to write to a file versus to the standard output?
  - Standard out is great for including print statements to do error checking
  - Also for passing output to other programs or scripts
  - Writing to a file is great if you need to run part of your script in one location and part in another
    - Generate data file on desktop, Run on TACC
  - Temporal separation of steps.
  - Import to R.
  - Some of this is personal; I output nearly everything to file so I have a constant record of my activities

Modules!

- Python is a popular language
- A lot of people have developed widgets and extensions for use with Python
- Next week Ben will talk about BioPython, which is excellent for sequence manipulation and some tree stuff
- This week we'll talk a little about some common modules for which almost everyone can find some use

os

- os allows you to interact with various operating system functions without leaving the Python environment
    - Do things like get your working directory
    - Change directories
    - Create a temporary file

os

- os.getcwd()
  - This functions prints the current working directory
- os.chdir()
  - Use this function to change directories
  - >>> path = "/filepath/to/location"
  - >>> os.chdir(path)

os

- Why would I do this?
- Why not just switch to UNIX and do it?

os

- Why would I do this?
- Why not just switch to UNIX and do it?
- If you're processing a lot of files that are in a directory structure

os

- os.tmpfile()
  - This sounds not useful, but actually can be
  - Creates a temporary file that persists for the duration of the script.
  - This is nice if you're doing something with lots of variables or a high-memory operation.

csv

- Let's say you have some data from a colleague. It's in a spreadsheet.
- Lots of people have data that's in spreadsheets.
- Some of them have big, kind hearts and wrote an interpreter for spreadsheet data

csv

```
>>>csv.reader(filename, dialect)
```

- This reads in the file and takes care of any meta characters (line endings, etc) that might trip you up
- Assumes a csv format, but for dialect, Excel can be subbed in, if the spreadsheet is Excel

csv

- Likewise, there is a writer function
- csv.writer(filename) writes out data in csv format
- We'll talk about databasing later in this course, but a csv file can be a very handy way to send data to a colleague and doesn't have a lot of the wonky formatting issues of . xls