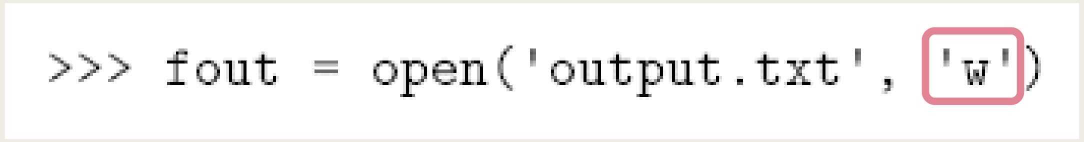
LECTURE 14: FILES

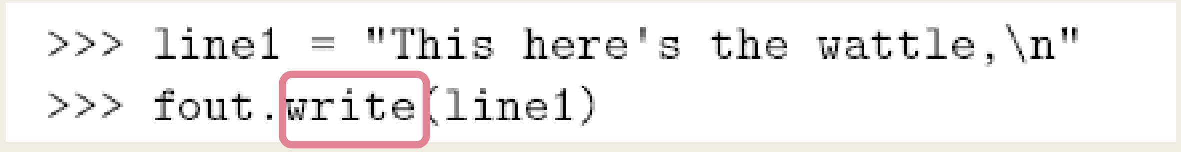
Hung-Yu Wei

Reading and writing from a file

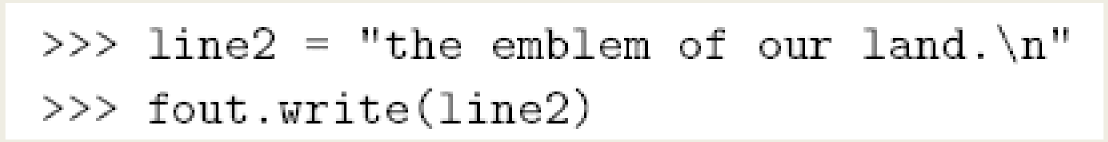
- Review: Reading from file
  fin = open('words.txt')
- Writing to a file
  - *w: write mode*
    - If the file exists, it will clear old data !



```
>>> fout = open('output.txt', 'w')
```

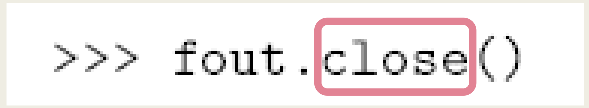


```
>>> line1 = "This here's the wattle,\n"
>>> fout.write(line1)
```



```
>>> line2 = "the emblem of our land.\n"
>>> fout.write(line2)
```

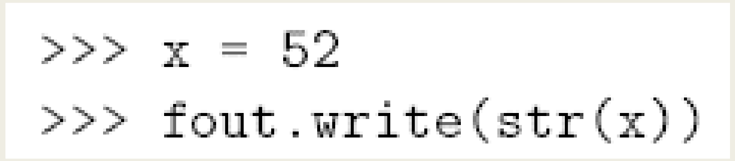
  - *write()*
  - *Close it at the end*
    - close()



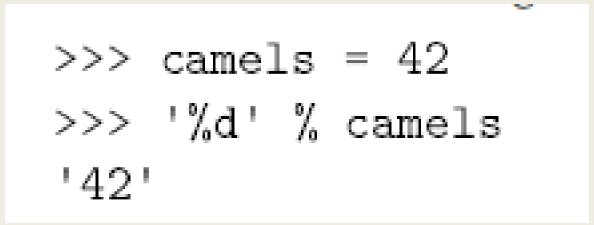
```
>>> fout.close()
```

Formatting

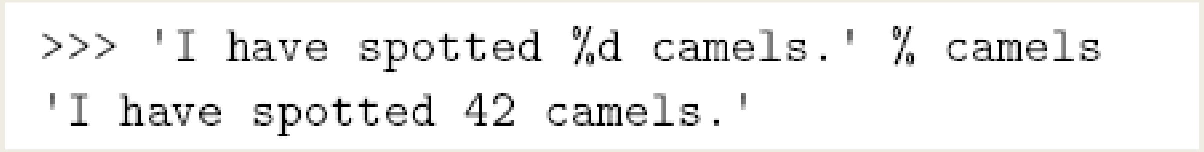
- write()
  - *It takes string format*
- Convert to a string
  - *str()*
- Formatting with %
  - *%d integer*
  - *%g floating point*
  - *%s string*



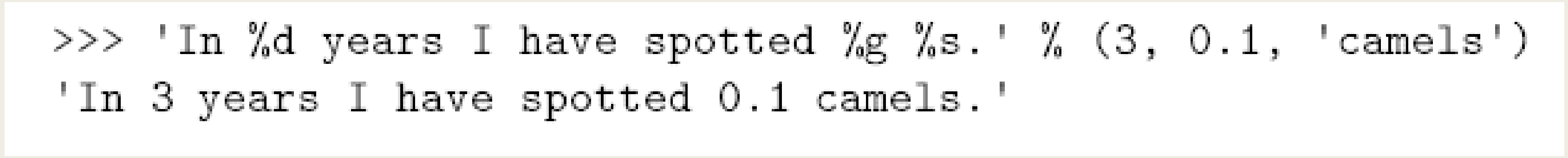
```
>>> x = 52
>>> fout.write(str(x))
```



```
>>> camels = 42
>>> '%d' % camels
'42'
```



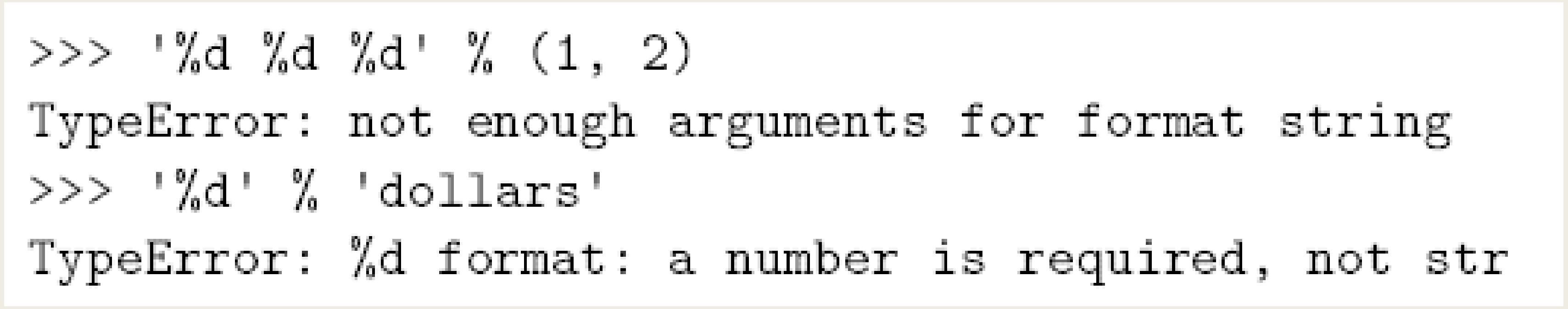
```
>>> 'I have spotted %d camels.' % camels
'I have spotted 42 camels.'
```



```
>>> 'In %d years I have spotted %g %s.' % (3, 0.1, 'camels')
'In 3 years I have spotted 0.1 camels.'
```

Formatting errors

- Matching the number of elements
- Matching the type



```
>>> '%d %d %d' % (1, 2)
TypeError: not enough arguments for format string
>>> '%d' % 'dollars'
TypeError: %d format: a number is required, not str
```

File names and paths

- import os
  - *getcwd()*
    - Return the current directory
  - *abspath()*
    - Get the absolute path to a file
  - *exists()*
    - Whether it exists
  - *isdir()*
    - Is it a directory?
  - *isfile()*
    - Is it a file?
  - *listdir()*
    - Return the list of files in a directory

```
>>> import os
>>> cwd = os.getcwd()
>>> cwd
'/home/dinsdale'
```

```
>>> os.path.abspath('memo.txt')
'/home/dinsdale/memo.txt'
```

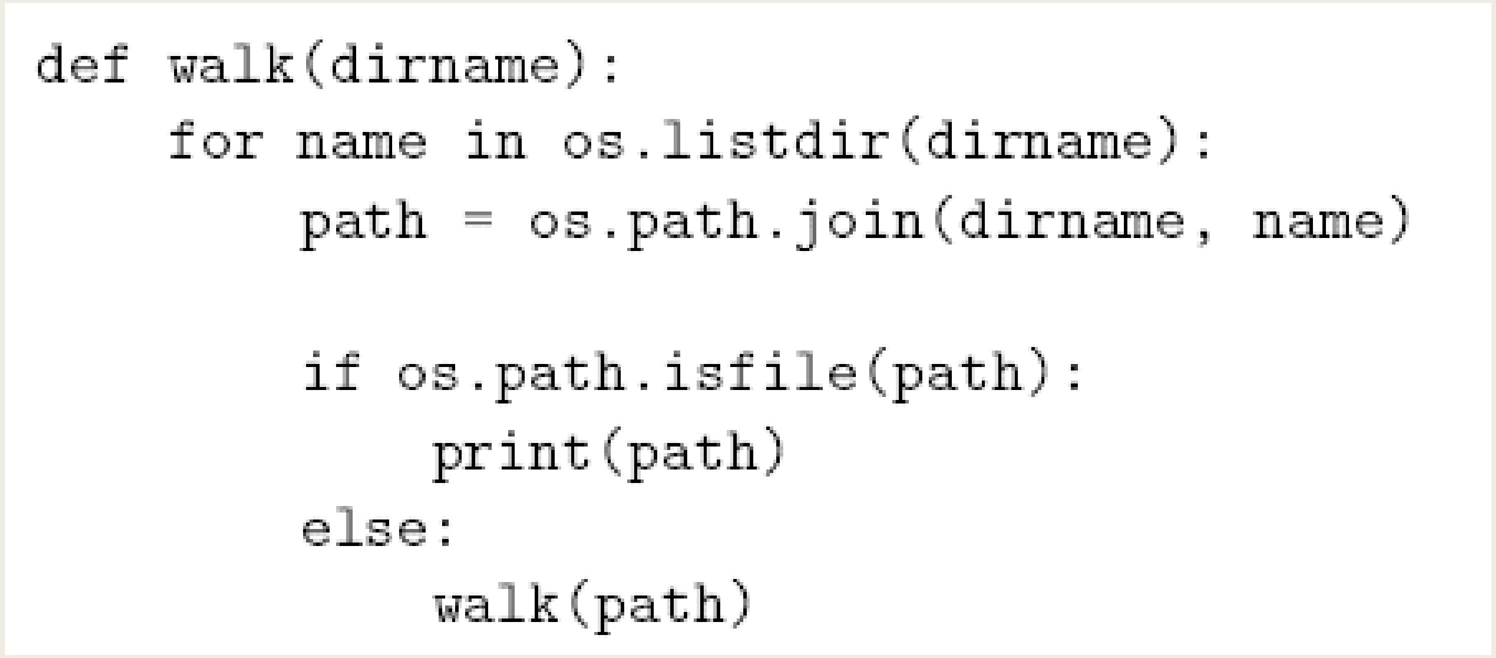
```
>>> os.path.exists('memo.txt')
True
```

```
>>> os.path.isdir('memo.txt')
False
>>> os.path.isdir('/home/dinsdale')
True
```

```
>>> os.listdir(cwd)
['music', 'photos', 'memo.txt']
```

Example: walk through a directory

- Walks through a directory, prints the names of all the files
  - *Recursive*



```
def walk(dirname):
    for name in os.listdir(dirname):
        path = os.path.join(dirname, name)

        if os.path.isfile(path):
            print(path)
        else:
            walk(path)
```

Error messages: open a file

```
>>> fin = open('bad_file')
IOError: [Errno 2] No such file or directory: 'bad_file'
```

```
>>> fout = open('/etc/passwd', 'w')
PermissionError: [Errno 13] Permission denied: '/etc/passwd'
```

```
>>> fin = open('/home')
IsADirectoryError: [Errno 21] Is a directory: '/home'
```

Catch Exception

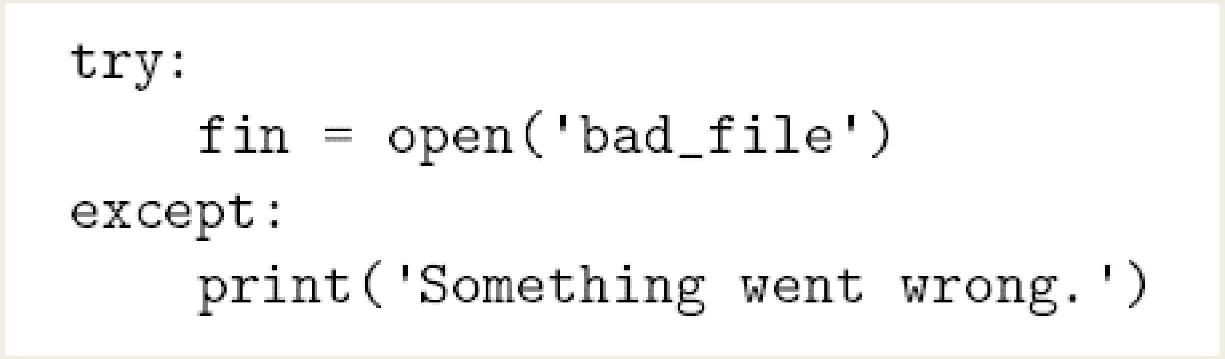
- Catching Exception
  - *fix the problem,*
  - *try again,*
  - *end the program gracefully*
- Syntax

  try:

  ...action to try...

  except:

  ...exception handling...



```
try:
    fin = open('bad_file')
except:
    print('Something went wrong.')
```

Database

```
>>> import dbm
>>> db = dbm.open('captions', 'c')
```

- import dbm
  - *keys and values have to be strings or bytes.*
- Open a database
  - *'c'*
  - *database should be created if it doesn't already exist.*

```
>>> db['cleese.png'] = 'Photo of John Cleese.'
```

- Create a new item
- Access an item
- Update an item
- Iteration with for
- Remember to close

```
>>> db['cleese.png']
b'Photo of John Cleese.'
```

```
>>> db['cleese.png'] = 'Photo of John Cleese doing a silly walk.'
>>> db['cleese.png']
b'Photo of John Cleese doing a silly walk.'
```

```
for key in db:
    print(key, db[key])
```

```
>>> db.close()
```

Pickling

- pickle module
  - *can store non-strings in a database*
- dump string
  - *pickle.dumps*
- load string
  - *pickle.loads*

```
>>> import pickle
>>> t = [1, 2, 3]
>>> pickle.dumps(t)
b'\x80\x03]q\x00(K\x01K\x02K\x03e.'
```

```
>>> t1 = [1, 2, 3]
>>> s = pickle.dumps(t1)
>>> t2 = pickle.loads(s)
>>> t2
[1, 2, 3]
```

```
>>> t1 == t2
True
>>> t1 is t2
False
```

Pipes: command line

```
>>> cmd = 'ls -l'
>>> fp = os.popen(cmd)
```

```
>>> res = fp.read()
```

```
>>> stat = fp.close()
>>> print(stat)
```

- Command line interface
  - *shell*
  - *Use pipe object in Python to execute command-line*
- Syntax

  os.popen()
- Getting the executing results

  read()

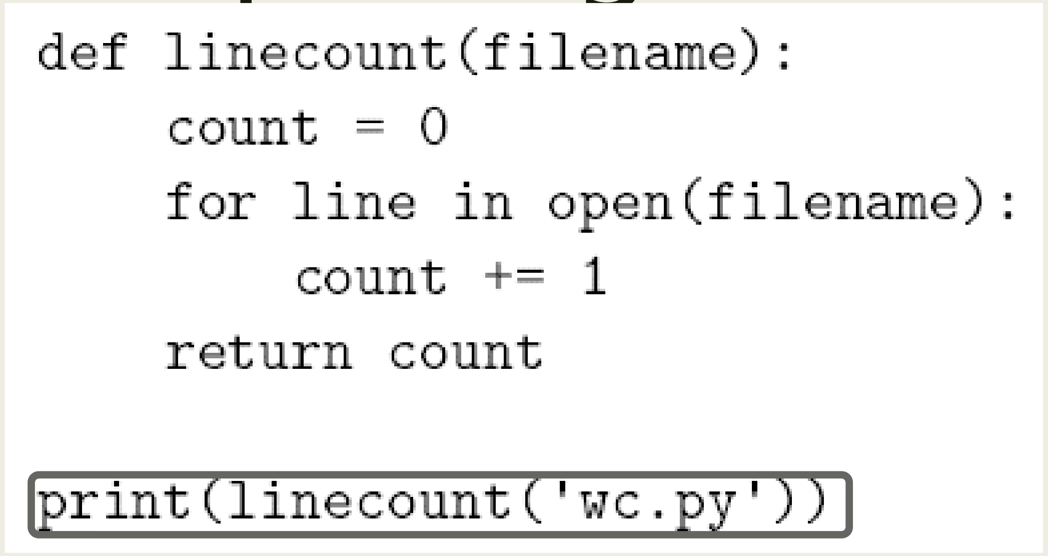
  readline()

  close()
  - *Close like a file*

```
>>> filename = 'book.tex'
>>> cmd = 'md5sum ' + filename
>>> fp = os.popen(cmd)
>>> res = fp.read()
>>> stat = fp.close()
>>> print(res)
1e0033f0ed0656636de0d75144ba32e0  book.tex
>>> print(stat)
None
```

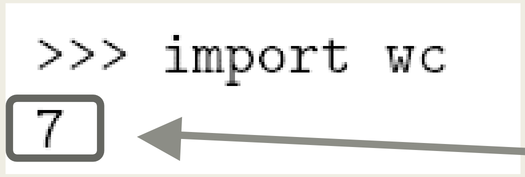
Writing module and importing

- Import functions from the other *.py file
  - *Example in textbook*
    - Define function linecount() in wc.py



```
def linecount(filename):
    count = 0
    for line in open(filename):
        count += 1
    return count

print(linecount('wc.py'))
```

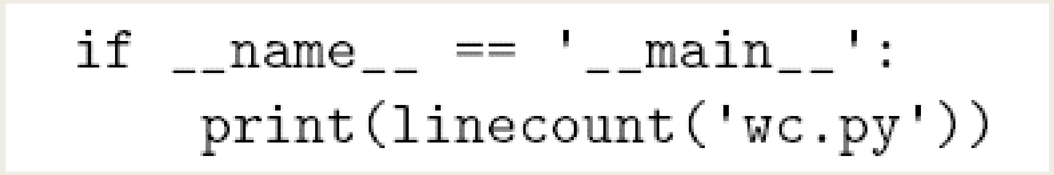


```
>>> import wc
7
```

- You want to import a module, but you don’t want to run some codes in the imported *.py file
- Syntax

  if __name__ == '__main__':

  things to do

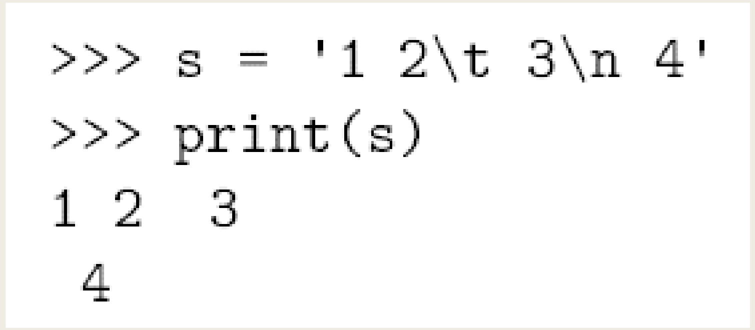


```
if __name__ == '__main__':
    print(linecount('wc.py'))
```

- __name__ is a built-in variable
  - *set when the program starts.*
  - *If the program is running, the test code runs.*
  - *if the module is being imported, the test code is skipped.*

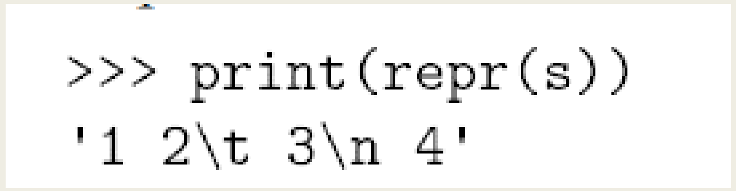
Debugging: seeing the spaces

- Syntax
  - *repr()*
  - *Show spaces/ tabs/new lines*



```
>>> s = '1 2\t 3\n 4'
>>> print(s)
1 2   3
 4
```

- Useful for debugging
- Be careful when you use different OS/file system



```
>>> print(repr(s))
'1 2\t 3\n 4'
```

Reading

- Chapter 14 in textbook “Think Python”