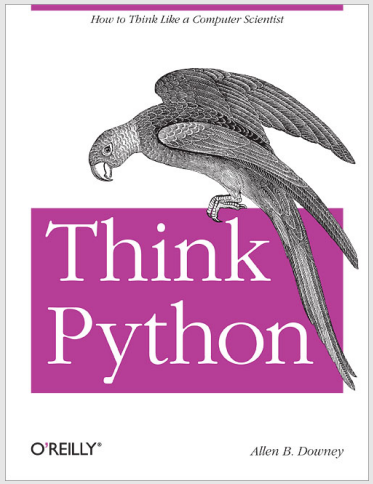
Contact

Dr. James Shackleford
shack@drexel.edu
Bossone 211

Office Hours: 3 – 4 pm (Tuesday)
Course Website: http://learn.dcollege.net

Textbook

Think Python
by Allen Downey
O'Reilly Press, 2015
ISBN-13: 978-1449330729
(Freely available in PDF format, check course website)



Grading

- 10% In-lab Programming Assignments
- 10% Take-Home Programming Assignments
- 35% Mid-term Exam
- 45% Final Exam

New Topic

Basic Exception Handling

Basic Exception Handling

Code:

```
1  my_list = range(5)
2
3  print 'for-loop:'
4  for i in my_list:
5      print i
6
7  print 'raw iter() w/ try-except:'
8  iterator = iter(my_list)
9  while True:
10     try:
11         i = iterator.next()
12     except:
13         break
14     else:
15         print i
16
17 del iterator
```

Output:

```
for-loop:
0
1
2
3
4
raw iter() w/ try-except:
0
1
2
3
4
```

Basic Exception Handling

Code:

```
1  my_list = range(5)
2
3  print 'for-loop:'
4  for i in my_list:
5      print i
6
7  print 'raw iter() w/ try-except:'
8  iterator = iter(my_list)
9  while True:
10     try:
11         i = iterator.next()
12     except:
13         break
14     else:
15         print i
16
17 del iterator
```

try this

Output:

```
for-loop:
0
1
2
3
4
raw iter() w/ try-except:
0
1
2
3
4
```

Basic Exception Handling

Code:

```
1  my_list = range(5)
2
3  print 'for-loop:'
4  for i in my_list:
5      print i
6
7  print 'raw iter() w/ try-except:'
8  iterator = iter(my_list)
9  while True:
10     try:
11         i = iterator.next()
12     except:
13         break
14     else:
15         print i
16
17 del iterator
```

if i = iterator.next( ) raises an exception, do this

Output:

```
for-loop:
0
1
2
3
4
raw iter() w/ try-except:
0
1
2
3
4
```

Basic Exception Handling

Code:

```
my_list = range(5)

print 'for-loop:'
for i in my_list:
    print i

print 'raw iter() w/ try-except:'
iterator = iter(my_list)
while True:
    try:
        i = iterator.next()
    except:
        break
    else:
        print i

del iterator
```

if i = iterator.next( ) DOES NOT raise an exception, do this

Output:

```
for-loop:
0
1
2
3
4
raw iter() w/ try-except:
0
1
2
3
4
```

New Topic

Basic File I/O

Basic File I/O

guests.txt

```
Date: August 1st, 2015
Guest Registry

Kiera Facio
Ozella Tallarico
Bella Hawke
Homer Ibanez
Regina Kunkel
Herb Schoonover
Terri Devilbiss
Lisbeth Schaner
Lori Lague
Angela Parkerson
```

Basic File I/O

read the ENTIRE file into memory as a list

```
>>> f = open('guests.txt')
>>> some_list = f.readlines()
>>> f.close()
>>> some_list
['Date: August 1st, 2015\n', 'Guest Registry\n', '\n', 'Kiera Facio\n', 'Ozella Tallarico\n',
 'Bella Hawke\n', 'Homer Ibanez\n', 'Regina Kunkel\n', 'Herb Schoonover\n', 'Terri Devilbiss\
n', 'Lisbeth Schaner\n', 'Lori Lague\n', 'Angela Parkerson\n']
```

guests.txt

```
Date: August 1st, 2015
Guest Registry

Kiera Facio
Ozella Tallarico
Bella Hawke
Homer Ibanez
Regina Kunkel
Herb Schoonover
Terri Devilbiss
Lisbeth Schaner
Lori Lague
Angela Parkerson
```

Basic File I/O

read the file into memory one line at a time

```
>>> f = open('guests.txt')
>>> f.readline()
'Date: August 1st, 2015\n'
>>> f.readline()
'Guest Registry\n'
>>> f.readline()
'\n'
>>> f.readline()
'Kiera Facio\n'
>>> f.readline()
'Ozella Tallarico\n'
>>> f.readline()
'Bella Hawke\n'
>>> f.readline()
'Homer Ibanez\n'
>>> f.readline()
'Regina Kunkel\n'
>>> f.readline()
'Herb Schoonover\n'
>>> f.readline()
'Terri Devilbiss\n'
>>>
```

```
>>> f.readline()
'Terri Devilbiss\n'
>>> f.readline()
'Lisbeth Schaner\n'
>>> f.readline()
'Lori Lague\n'
>>> f.readline()
'Angela Parkerson\n'
>>> f.readline()
''
>>> f.readline()
''
>>> f.readline()
''
```

Basic File I/O

Alternative, line by line access:

```
>>> f = open('guests.txt')
>>> for line in f:
...     print line,
...
Date: August 1st, 2015
Guest Registry

Kiera Facio
Ozella Tallarico
Bella Hawke
Homer Ibanez
Regina Kunkel
Herb Schoonover
Terri Devilbiss
Lisbeth Schaner
Lori Lague
Angela Parkerson
>>> f.close()
```

Note:

Our current position in the file is maintained by the OS.

When we read a byte, the file position is incremented by a byte.

When we write a byte, the file position is incremented by a byte.

We can also manually move the file position...

Basic File I/O

Reading one byte at a time:

```
>>> f = open('guests.txt')
>>> f.readline()
'Date: August 1st, 2015\n'
>>> f.read(5)
'Guest'
```

Read 5 bytes

(each character is a byte)

Basic File I/O

Using seek() to move around:

```
>>> f = open('guests.txt')
>>> f.readline()
'Date: August 1st, 2015\n'
>>> f.read(5)
'Guest'
>>> f.read(5)
' Regi'
>>> f.seek(0)
>>> f.read(5)
'Date:'
```

Move to byte zero

starting from the beginning of the file

Basic File I/O

Using seek() to move around (better):

```
>>> import os
>>> f.seek(0, os.SEEK_SET)
>>> f.seek(0, 0)
>>> f.seek(-10, os.SEEK_END)
>>> f.seek(-10, 2)
>>> f.seek(5, os.SEEK_CUR)
>>> f.seek(5, 1)
```

Move to byte zero

starting from the beginning of the file

Using seek() to move around (better):

```
>>> import os
>>> f.seek(0, os.SEEK_SET)
>>> f.seek(0, 0)
>>> f.seek(-10, os.SEEK_END)
>>> f.seek(-10, 2)
>>> f.seek(5, os.SEEK_CUR)
>>> f.seek(5, 1)
```

Move BACKWARD 10 bytes

starting from the END of the file

Using seek() to move around (better):

```
>>> import os
>>> f.seek(0, os.SEEK_SET)
>>> f.seek(0, 0)
>>> f.seek(-10, os.SEEK_END)
>>> f.seek(-10, 2)
>>> f.seek(5, os.SEEK_CUR)
>>> f.seek(5, 1)
```

Move FORWARD 5 bytes

starting from the CURRENT FILE POSITION

Basic File I/O

Using seek() to move around (better):

```
>>> import os
>>> f.seek(0, os.SEEK_SET)
>>> f.seek(0, 0)
>>> f.seek(-10, os.SEEK_END)
>>> f.seek(-10, 2)
>>> f.seek(5, os.SEEK_CUR)
>>> f.seek(5, 1)
```

SAME

SAME

SAME

Basic File I/O

Writing to a file:

```
>>> list = ['Bob\n', 'Mary\n', 'Sam', 'Bill']

>>> f = open('output.txt', 'w+')
>>> f.writelines(list)
>>> f.seek(0)
>>> for line in f:
...     print line,
...
Bob
Mary
SamBill
```

read/write mode

write data to file

rewind to start of file

Basic File I/O

Valid File Modes:

r Open text file for reading. The stream is positioned at the beginning of the file. **[Default]**

r+ Open for reading and writing. The stream is positioned at the beginning of the file.

w Truncate file to zero length or create text file for writing. The stream is positioned at the beginning of the file.

w+ Open for reading and writing. The file is created if it does not exist, otherwise it is truncated. The stream is positioned at the beginning of the file.

a Open for writing. The file is created if it does not exist. The stream is positioned at the end of the file. Subsequent writes to the file will always end up at the then current end of file, irrespective of any intervening seek() or similar.

a+ Open for reading and writing. The file is created if it does not exist. The stream is positioned at the end of the file. Subsequent writes to the file will always end up at the then current end of file, irrespective of any intervening seek() or similar.

Basic File I/O

Features	File Mode					
	r	r+	w	w+	a	a+
read	X	X		X		X
write		X	X	X	X	X
create			X	X	X	X
truncate			X	X		
position at start	X	X	X	X		
position at end					X	X

Basic File I/O

Writing to a file a few bytes at a time:

```
>>> f = open('foo.txt', 'w+')
>>> f.write('test')
>>> f.write('123')
>>> f.close()
>>> f = open('foo.txt', 'r')
>>> f.readline()
'test123'
```

Basic File I/O

Better file handling by using exceptions:

```
1  f = open('output.txt')
2  try:
3      f.writelines('test\n')
4      # Maybe do other stuff in here that could
5      # cause the program to throw an exception
6  finally:
7      # this will happen even if the code inside
8      # of the try block fails, so our file will
9      # safely close no matter what!
10     f.close()
```

Python has a built-in construct for this sorta thing that you will see far more often.

Basic File I/O

File I/O block with an unconditional built-in f.close()

```
1 with open('guests.txt') as f:
2     # do stuff with your file object f in here.
3     # If an exception is thrown, the file will
4     # automatically close.  When you exit this
5     # block, the file will close.
6     for line in f:
7         print line,
```

Pseudo-Code:

```
class file(object):
    def __enter__(self):
        # called when we enter the with block
        #   setup file descriptor stuff
        #   in here and return file object
        return file_object
    def __exit__(self):
        # called when we exit the with block
        #   close the file
        file_object.close()
```

This works because the file object returned by open() has the magic methods: __enter__() and __exit__()