

Repeated Steps

Output:

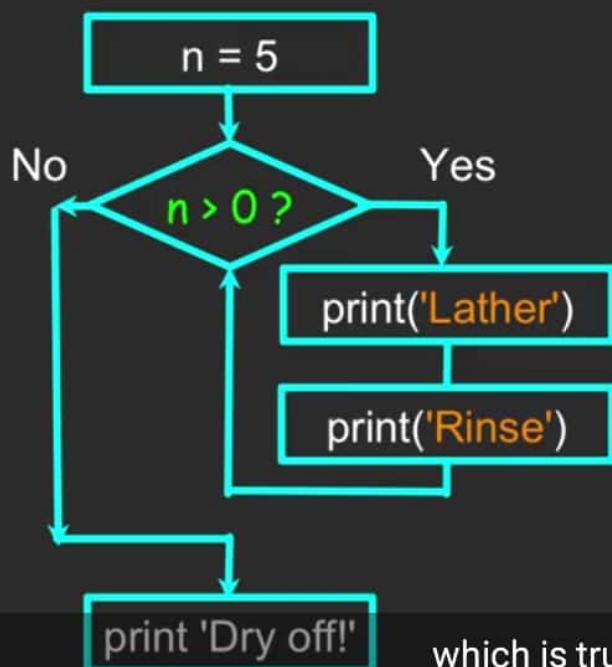
Program:

```

n = 5.
while n > 0 :
    print(n)
    n = n - 1
print('Blastoff!')
print(n)
  
```

5
4
3
2
1
Blastoff!
0

Loops (repeated steps) have **iteration variables** that change each time through a loop. Often these **iteration variables** go through a sequence of numbers. These are called indefinite loops.



An Infinite Loop

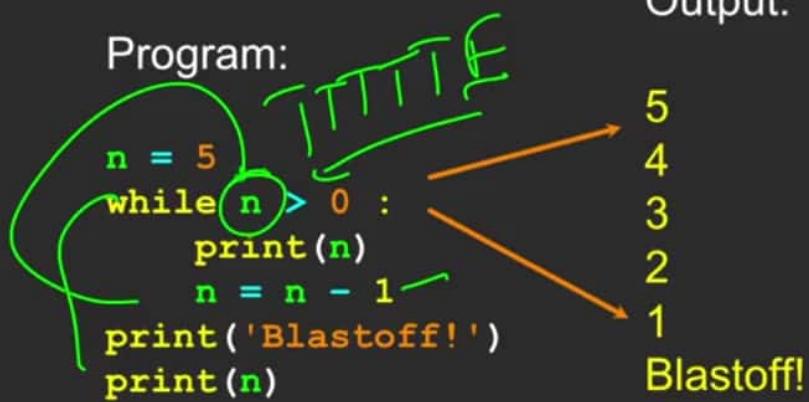
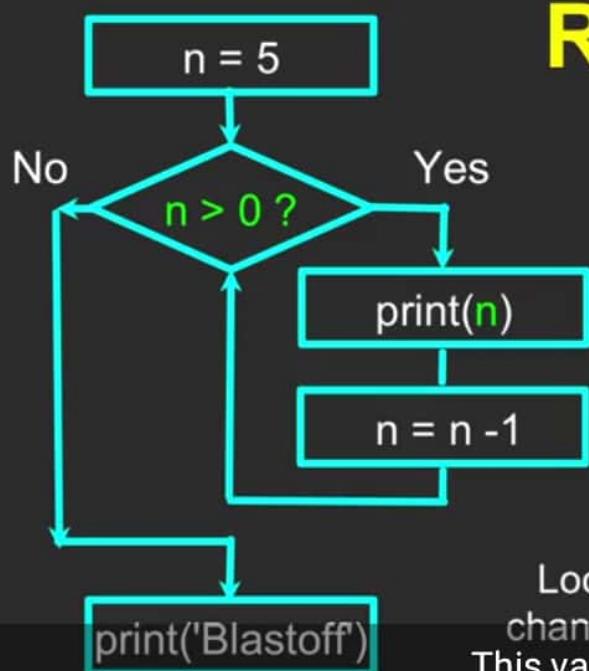
```
n = 5
while n > 0 :
    print('Lather')
    print('Rinse')
    print('Dry off!')
```

A handwritten note shows the pseudocode for the loop:

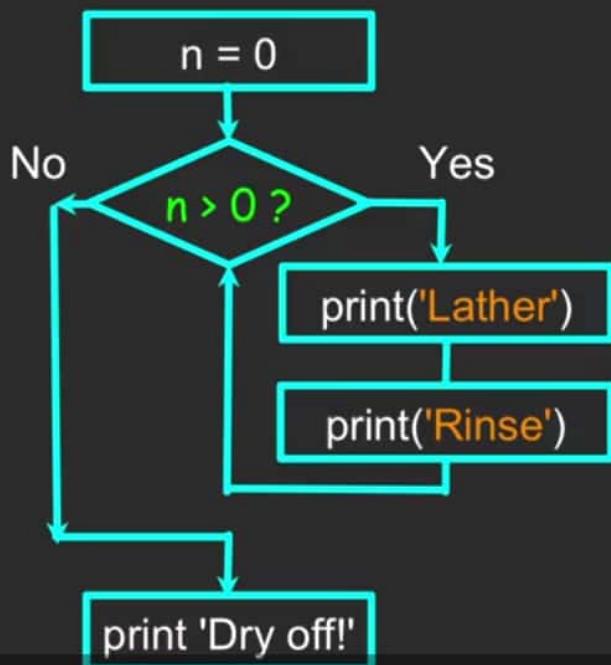
What is wrong with this loop?

which is true, print this, come back up,
check again. Is n greater than 0? Yeah.

Repeated Steps



Loops (repeated steps) have **iteration variables** that change each time through a loop. Often these **iteration variables** are **variables** that we use to control it. This **variables** goes through a sequence of numbers. is what we call an iteration variable.



Another Loop

```
n = 0
while n > 0 :
    print('Lather')
    print('Rinse')
print('Dry off!')
```

What is this loop doing?

are what are called zero-trip loop.

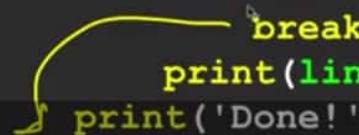
Breaking Out of a Loop

- The **break** statement ends the current loop and jumps to the statement immediately following the loop
- It is like a loop test that can happen anywhere in the body of the loop

```
while True:  
    line = input('> ')  
    if line == 'done' :  
        break  
    print(line)  
print('Done!')
```

moves to the line beyond
the end of the loop.

> hello there
hello there
> finished
finished
> done
Done!



Breaking Out of a Loop

- The **break** statement ends the current loop and jumps to the statement immediately following the loop
- It is like a loop test that can happen anywhere in the body of the loop

```
while True:  
    line = input('> ')  
    if line == 'done' :  
        break  
    print(line)  
print('Done!')
```

And it immediately leaves, so it doesn't print the word done here.

hello there
hello there
> finished
finished
><done>
Done!

Breaking Out of a Loop

- The **break** statement ends the current loop and jumps to the statement immediately following the loop
- It is like a loop test that can happen anywhere in the body of the loop

```
while True:  
    line = input('> ')  
    if line == 'done' :  
        break  
    print(line)  
print('Done!')
```

```
> hello there  
hello there  
> finished  
finished  
> done  
Done!
```

The break escapes the block, right?

Finishing an Iteration with Continue

The **continue** statement ends the current iteration and jumps to the top of the loop and starts the next iteration

```
while True:  
    line = input('> ')  
    if line[0] == '#':  
        continue  
    if line == 'done':  
        break  
    print(line)  
print('Done!')
```

> hello there
hello there
> # don't print this
> print this!
print this!
→> done
Done!

character, we skip back to the beginning of the loop.

18:09

... ⊞ 4G H 84



So break skips out of the loop and
continue skips to the top of the loop.

M

18:09

... ⊞ 4G H 84



Abandons the current iteration and
goes to the next iteration.

M

A Simple Definite Loop

```
~ /  
for i in [5, 4, 3, 2, 1] :  
    print(i)  
print('Blastoff!')  
5  
4  
3  
2  
1  
Blastoff!
```

in is another Python reserved word.

A Simple Definite Loop

```
for i in [5, 4, 3, 2, 1] :  
    print(i)  
print('Blastoff!')
```

5 ✓
4
3
2
1
Blastoff!

The second time, i is 4, print.

A Definite Loop with Strings

```
friends = ['Joseph', 'Glenn', 'Sally']
for friend in friends :
    print('Happy New Year:', friend)
print('Done!')
```

Happy New Year: Joseph
Happy New Year: Glenn
Happy New Year: Sally
Done!

So square brackets, list of strings:
Joseph, Glenn, and Sally.

Looking at In...

- The **iteration variable** “iterates” through the **sequence** (ordered set)
- The **block (body)** of code is executed once for each value **in** the **sequence**
- The **iteration variable** moves through all of the values **in** the **sequence**

Iteration variable
↓
`for i in [5, 4, 3, 2, 1] :
 print(i)`

Five-element sequence
↓

If you're a math person,

Making “smart” Loops

The trick is “knowing” something about the whole loop when you are stuck writing code that only sees one entry at a time

Set some variables to initial values

for `thing` in `data`:

Look for something or do something to each entry separately, updating a variable

But we're going to do something before the loop starts,

Look at the variables

Finding the Largest Value

```
largest_so_far = -1
print('Before', largest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num > largest_so_far :
        largest_so_far = the_num
    print(largest_so_far, the_num)

print('After', largest_so_far)
```

```
$ python largest.py
Before -1
9 9 .
41 41
41 12
41 3
74 74
74 15
After 74
```

We make a variable that contains the largest value we have seen so far. If the current number we are looking at is larger, it is the new largest value we have seen so far.

And so I'm going to make a variable.

String Data Type

- A string is a sequence of characters
- A string literal uses quotes
'Hello' or "Hello"
- For strings, + means "concatenate"
- When a string contains numbers, it is still a string
- We can convert numbers in a string into a number using `int()`

```
>>> str1 = "Hello" z
>>> str2 = 'there'
>>> bob = str1 + str2
>>> print(bob)
Hellothere
>>> str3 = '123'
>>> str3 = str3 + 1
Traceback (most recent call
last): File "<stdin>", line 1,
in <module>
TypeError: cannot concatenate
'str' and 'int' objects
>>> x = int(str3) + 1
>>> print(x)
124
>>>
```

And so, you know, we take two strings,

Reading and Converting

- We prefer to read data in using **strings** and then parse and convert the data as we need
- This gives us more control over error situations and/or bad user input
- Input numbers must be converted from strings

```
>>> name = input('Enter: ')
Enter:Chuck
>>> print(name)
Chuck
>>> apple = input('Enter: ')
Enter:100
>>> x = apple - 10
Traceback (most recent call
last): File "<stdin>", line 1,
in <module>
TypeError: unsupported operand
type(s) for -: 'str' and 'int'
>>> x = int(apple) - 10
>>> print(x)
```

The input function prints out a prompt.
That's a prompt.

Reading and Converting

- We prefer to read data in using **strings** and then parse and convert the data as we need
- This gives us more control over error situations and/or bad user input
- Input numbers must be **converted** from strings

```
>>> name = input('Enter: ')
Enter: Chuck
>>> print(name)
Chuck
>>> apple = input('Enter: ')
Enter: 100
>>> x = apple - 10
Traceback (most recent call
last): File "<stdin>", line 1,
in <module>
TypeError: unsupported operand
type(s) for -: 'str' and 'int'
>>> x = int(apple) - 10
>>> print(x)
90
```

Input gives us back a string,



Looking Inside Strings

- We can get at any single character in a string using an index specified in **square brackets**
- The index value must be an integer and starts at zero
- The index value can be an expression that is computed

b	a	n	a	n	a
0	1	2	3	4	5

```
>>> fruit = 'bahana'  
>>> letter = fruit[1]  
>>> print(letter)  
a  
>>> x = 3  
>>> w = fruit[x - 1]  
>>> print(w)
```

We have this string banana,

n

A Character Too Far

- You will get a **python error** if you attempt to index beyond the end of a string.
- So be careful when constructing index values and slices

```
>>> zot = 'abc'  
>>> print(zot[5])  
Traceback (most recent call  
last): File "<stdin>", line  
1, in <module>  
IndexError:  
string index out of range  
>>>
```

.

and in this one, I'm making a mistake.

Strings Have Length

b	a	n	a	n	a
0	1	2	3	4	5

The built-in function `len` gives us the length of a string

```
>>> fruit = 'banana'  
>>> print(len(fruit))  
6
```

We can pass a string into the `len` function,

Looping Through Strings

Using a `while` statement and an `iteration variable`, and the `len` function, we can construct a loop to look at each of the letters in a string individually

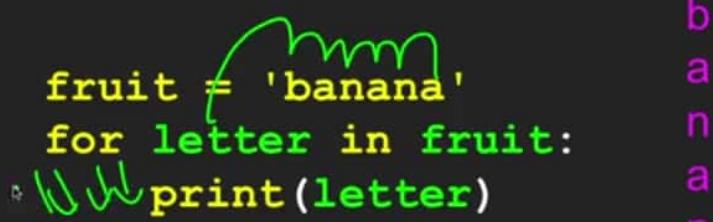
```
fruit = 'banana'      - 0 b
index = 0             - 1 a
while index < len(fruit): : 2 n
    letter = fruit[index] 3 a
    print(index, letter) 4 n
    index = index + 1     5 a
```

the index and the letter that happens to be in that string at the index.

Looping Through Strings

- A definite loop using a `for` statement is much more elegant
- The `iteration variable` is completely taken care of by the `for` loop

```
fruit = 'banana'  
for letter in fruit:  
    print(letter)
```

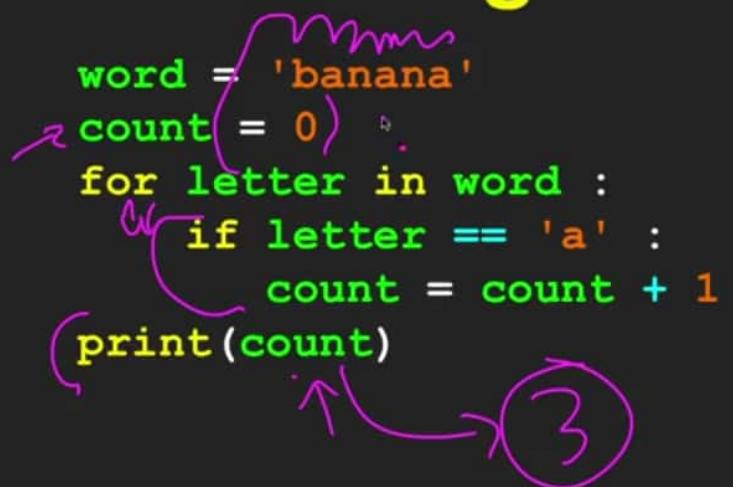


And that means it's going to run
this loop six times,

Looping and Counting

This is a simple loop that loops through each letter in a string and counts the number of times the loop encounters the 'a' character

```
word = 'banana'  
count = 0  
for letter in word :  
    if letter == 'a' :  
        count = count + 1  
print(count)
```



and out comes 3 because there are 3 a's.

Looking Deeper into `in`

- The **iteration variable** “iterates” through the **sequence** (ordered set)
- The **block (body)** of code is executed once for each value **in** the **sequence**
- The **iteration variable** moves through all of the values **in** the **sequence**

how we're supposed to just run
this loop six times,

```
for letter in 'banana' :  
    print(letter)
```

Slicing Strings

- We can also look at any continuous section of a string using a **colon operator**
- The second number is one beyond the end of the slice - “up to but not including”
- If the second number is beyond the end of the string, it stops at the end

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4		5	6	7	8	9	10 11

```
>>> s = 'Monty Python'  
>>> print(s[0:4])  
Mont  
>>> print(s[6:7])  
P  
>>> print(s[6:20])  
Python
```

And we're going to use the same square bracket to do slicing,

Slicing Strings

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4		5	6	7	8	9	10 11

If we leave off the first number or the last number of the slice, it is assumed to be the beginning or end of the string respectively

```
>>> s = 'Monty Python'  
>>> print(s[:2])  
Mo  
>>> print(s[8:])  
thon  
>>> print(s[:])  
Monty Python
```

it's really common to either eliminate the first character,



String Concatenation

When the `+` operator is applied to strings, it means “concatenation”

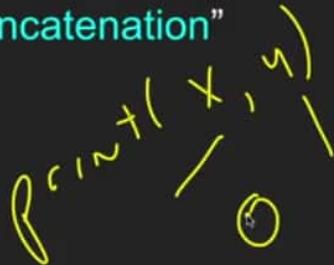
```
>>> a = 'Hello'  
>>> b = a + 'There'  
>>> print(b)  
HelloThere  
>>> c = a + ' ' + 'There'  
>>> print(c)  
Hello There  
>>>
```

There and so we've explicitly put the space in.



String Concatenation

When the + operator is applied to strings, it means "concatenation"



```
>>> a = 'Hello'  
>>> b = a + 'There'  
>>> print(b)  
HelloThere  
>>> c = a + ' ' + 'There'  
>>> print(c)  
Hello There  
>>>
```

the two things come out and there's a space in between them.

Using **in** as a Logical Operator

- The **in** keyword can also be used to check to see if one string is “in” another string
- The **in** expression is a logical expression that returns **True** or **False** and can be used in an **if** statement

```
>>> fruit = 'banana'  
>>> 'n' in fruit == True  
  
>>> 'm' in fruit == False  
>>> 'nan' in fruit == True  
>>> if 'a' in fruit:  
...     print('Found it!')  
...  
Found it!  
>>>
```

and so we get back a **True**.



Using **in** as a Logical Operator

- The **in** keyword can also be used to check to see if one string is “in” another string
- The **in** expression is a logical expression that returns **True** or **False** and can be used in an **if** statement

```
>>> fruit = 'banana'  
>>> 'n' in fruit  
True  
>>> 'm' in fruit  
False  
>>> 'nan' in fruit  
True  
>>> if ('a' in fruit):  
...     print('Found it!')  
...  
Found it!
```

You have to give this blank line
to convince it.



Using **in** as a Logical Operator

- The **in** keyword can also be used to check to see if one string is “in” another string
- The **in** expression is a logical expression that returns **True** or **False** and can be used in an **if** statement

```
>>> fruit = 'banana'  
>>> 'n' in fruit  
True  
>>> 'm' in fruit  
False  
>>> 'nan' in fruit  
True  
>>> if ('a' in fruit):  
...     print('Found it!')  
...  
Found it!  
>>>
```

Just a little note, if you're using the interactive interpreter,



Using **in** as a Logical Operator

- The **in** keyword can also be used to check to see if one string is “in” another string
- The **in** expression is a logical expression that returns **True** or **False** and can be used in an **if** statement

```
>>> fruit = 'banana'  
>>> 'n' in fruit  
True  
>>> 'm' in fruit  
False  
>>> 'nan' in fruit  
True  
>>> if ('a' in fruit):  
...     print('Found it!')  
...  
Found it!  
>>>
```

and you're using – you actually have to throw a blank line here.



Using **in** as a Logical Operator

- The **in** keyword can also be used to check to see if one string is “in” another string
- The **in** expression is a logical expression that returns **True** or **False** and can be used in an **if** statement

```
>>> fruit = 'banana'  
>>> 'n' in fruit  
True  
>>> 'm' in fruit  
False  
>>> 'nan' in fruit  
True  
>>> if ('a' in fruit):  
...     print('Found it!')  
...  
Found it!  
>>>
```

You don't need a blank line in real Python,
like if you're writing in a file.



Using **in** as a Logical Operator

- The **in** keyword can also be used to check to see if one string is “in” another string
- The **in** expression is a logical expression that returns **True** or **False** and can be used in an **if** statement

```
>>> fruit = 'banana'  
>>> 'n' in fruit  
True  
>>> 'm' in fruit  
False  
>>> 'nan' in fruit  
True  
>>> if ('a' in fruit):  
...     print('Found it!')  
...  
Found it!  
>>>
```

But, you know, if you type this and then you indent that, it works,



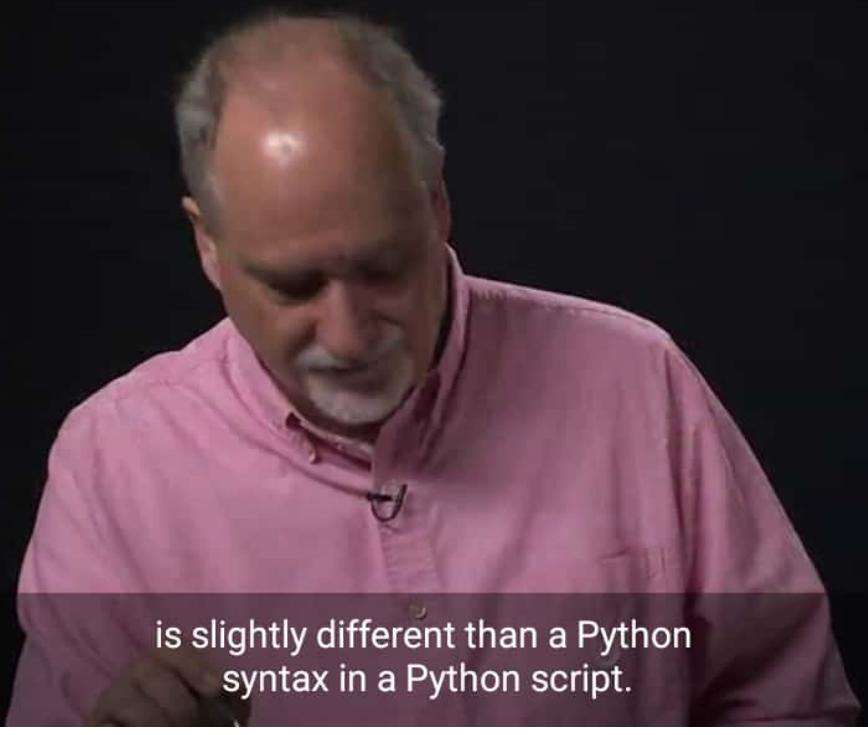
Using **in** as a Logical Operator

- The **in** keyword can also be used to check to see if one string is “in” another string
- The **in** expression is a logical expression that returns **True** or **False** and can be used in an **if** statement

```
>>> fruit = 'banana'  
>>> 'n' in fruit  
True  
>>> 'm' in fruit  
False  
>>> 'nan' in fruit  
True  
>>> if ('a' in fruit):  
...     print('Found it!')  
...  
Found it!
```

It's a situation where the interpreter,
the chevron prompt,





is slightly different than a Python
syntax in a Python script.



String Comparison

```
if word == 'banana':  
    print('All right, bananas.')  
  
if word < 'banana':  
    print('Your word, ' + word + ', comes before banana.')  
elif word > 'banana':  
    print('Your word, ' + word + ', comes after banana.')  
else:  
    print('All right, bananas.')  
  
Chuck  
Gоварь  
Пчек  
Говарь
```

But, it's ok. It makes some sense,
it's all consistent.

- Python has a number of string **functions** which are in the **string library**
- These **functions** are already **built into** every string - we invoke them by appending the function to the string variable
- These **functions** do not modify the original string, instead they return a new string that has been altered

like calling a function called `lower()`
and passing `greet` into it.

String Library

```
>>> greet = 'Hello Bob'  
>>> zap = greet.lower()  
>>> print(zap)  
hello bob  
>>> print(greet)  
Hello Bob  
>>> print('Hi There'.lower())  
hi there  
>>>
```

(down)

- Python has a number of string **functions** which are in the **string library**
- These **functions** are already **built into** every string - we invoke them by appending the function to the string variable
- These **functions** do not modify the original string, instead they return a new string that has been altered

String Library

```
>>> greet = 'Hello Bob'  
>>> zap = greet.lower()  
>>> print(zap)  
hello bob  
>>> print(greet)  
Hello Bob  
>>> print('Hi There'.lower())  
hi there  
>>>
```

Annotations:

- A purple bracket groups 'greet' and 'lower()' in the first line.
- A purple bracket groups 'print(zap)' and 'hello bob' in the second line.
- A purple bracket groups 'greet' in the third line.
- A purple bracket groups 'print('Hi There'.lower())' in the fourth line.
- A purple bracket groups 'lower()' in the fifth line.
- A purple bracket groups 'hello bob' and 'Cop M L. C.' in the sixth line.

OK? And so even constants have this sort of built-in capability.



```
>>> stuff = 'Hello world'  
>>> type(stuff)  
<class 'str'>   
>>> dir(stuff)   
['capitalize', 'casefold', 'center', 'count', 'encode',  
'endswith', 'expandtabs', 'find', 'format', 'format_map',  
'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit',  
'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace',  
'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip',  
'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust',  
'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines',  
'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper',  
'zfill']
```

-- METHODS in str

<https://docs.python.org/3/library/stdtypes.html#string-methods>

But these are a bunch of methods
in the class str.



String Library

```
str.capitalize()  
str.center(width[, fillchar])  
str.endswith(suffix[, start[, end]])  
str.find(sub[, start[, end]])  
str.lstrip([chars])
```

```
str.replace(old, new[, count])  
str.lower()  
str.rstrip([chars])  
str.strip([chars])  
str.upper()
```

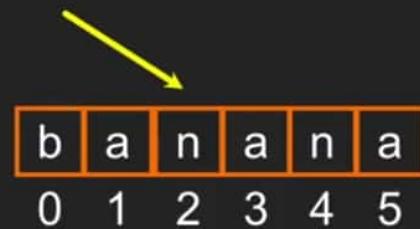
.

which takes a string like, you know, abc,

Searching a String

- We use the `find()` function to search for a substring within another string
- `find()` finds the first occurrence of the substring
- If the substring is not found, `find()` returns `-1`
- Remember that string position starts at zero

Or, find says, where inside the banana is 'na'?



```
>>> fruit = 'banana'  
>>> pos = fruit.find('na')  
>>> print(pos)  
2  
>>> aa = fruit.find('z')  
>>> print(aa)  
-1
```

Search and Replace

- The `replace()` function is like a “search and replace” operation in a word processor
- It replaces **all** occurrences of the **search string** with the **replacement string**

```
>>> greet = 'Hello Bob'  
>>> nstr = greet.replace('Bob', 'Jane')  
>>> print(nstr)  
Hello Jane  
>>> nstr = greet.replace('o', 'X')  
>>> print(nstr)  
HellX BXb  
>>>
```

So that says go find all the Bobs and replace them with



Stripping Whitespace

- Sometimes we want to take a string and remove whitespace at the beginning and/or end
- `lstrip()` and `rstrip()` remove whitespace at the left or right
- `strip()` removes both beginning and ending whitespace And we can strip from the right side if we want and then we can strip from the left side.

```
>>> greet = ' ~Hello Bob ~'
>>> greet.lstrip()
'Hello Bob '
>>> greet.rstrip()
' ~Hello Bob'
>>> greet.strip()
'Hello Bob'
```

Prefixes

```
>>> line = 'Please have a nice day'  
>>> line.startswith('Please')  
True ✓  
>>> line.startswith('p')  
False ↗
```

it doesn't start with a lowercase p. So,



6:03

... ⊞ H 65

Strings - Part 2

PYTHON FOR
Click with the mouse or tablet to draw with pen 2

Parsing and Extracting

21 31
↓ ↓

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
>>> data = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'  
>>> atpos = data.find('@')  
>>> print(atpos)  
21  
>>> spos = data.find(' ',atpos)  
>>> print(spos)  
31  
>>> host = data[atpos+1 : spos]  
>>> print(host)  
uct.ac.za
```



So slicing is the word for using
that : operator. So let's take a look.

Parsing and Extracting

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

21 31

```
>>> data = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'  
>>> atpos = data.find('@')  
>>> print(atpos)  
21  
>>> spos = data.find(' ',atpos)  
>>> print(spos)  
31  
>>> host = data[atpos+1 : spos]  
>>> print(host)  
uct.ac.za
```



but not including the space position,

Strings and Character Sets

Python 2.7.10

```
>>> x = '이광춘'  
>>> type(x)  
<type 'str'>  
>>> x = u'이광춘'  
>>> type(x)  
<type 'unicode'>  
>>>
```

Python 3.5.1

```
>>> x = '이광춘'  
>>> type(x)  
<class 'str'>  
>>> x = u'이광춘'  
>>> type(x)  
<class 'str'>  
>>>
```

In Python 3, all strings are Unicode

And so, you would indicate a Unicode
string by adding this u prefix.



Strings and Character Sets

```
Python 2.7.10
>>> x = '0이광춘'
>>> type(x)
<type 'str'>
>>> x = u'0이광춘'
>>> type(x)
<type 'unicode'>
>>>
```

```
Python 3.5.1
>>> x = '0이광춘'
>>> type(x)
<class 'str'>
>>> x = u'0이광춘'
>>> type(x)
<class 'str'>
>>>
```

In Python 3, all strings are Unicode
you'd have to kind of go through some
conversion and it was a little bit weird.

6:10 ... ⌂ H 65

Chrome File Edit View History Bookmarks People Window Help

PY4E - Python for Everybody Chapter 6 | Python For Everyone

https://books.trinket.io/pfe/06-strings.html

Python for Everyone Chapters Strings

Exercises

Exercise 5: Take the following Python code that stores a string:

```
str = 'X-DSPAM-Confidence: 0.8475 '
```

Use `find` and string slicing to extract the portion of the string after the colon character and then use the `float` function to convert the extracted string into a floating point number.

Exercise 6:

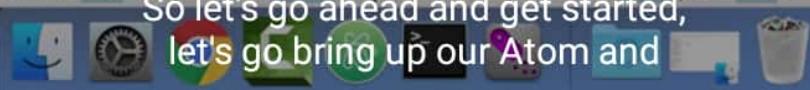
Read the documentation of the string methods at

<https://docs.python.org/3.5/library/stdtypes.html#string-methods>

You might want to experiment with some of them to make sure you understand how they work. `strip` and `replace` are particularly useful.

The documentation uses a syntax that might be confusing. For example in `find(sub[, start[, end]])`, the brackets indicate optional arguments. So `sub` is required, but `start` is optional, and if you include `start`, then `end` is optional.

So let's go ahead and get started,
let's go bring up our Atom and



6:15 ... ⚡ Wed 7:22 AM

Atom File Edit View Selection Find Packages Window Help

ex_06_05.py — /Users/py4e/Desktop/py4e

```
py4e
> ex_02_02
> ex_02_03
> ex_03_01
> ex_03_02
> ex_04_06
> ex_05_01
> ex_06_05
first.py
```

ex_06_05.py *

```
1 str = 'X-DSPAM-Confidence: 0.8475'
2
3 ipos = str.find(':')
4 print(ipos)
5 piece = str[ipos:]
6 print(piece)
```

-bash — 65x24

```
pwd
_01
cd ..

ex_04_06      ex_06_05
ex_05_01      first.py
ex_06_05
ls
python3 ex_06_05.py
python3 ex_06_05.py
python3 ex_06_05.py
```

And when I'm doing string parsing,
tearing strings apart,

6:16 ... ⌂ 64

Atom File Edit View Selection Find Packages Window Help

ex_06_05.py — /Users/py4e/Desktop/py4e

```
py4e
├── ex_02_02
├── ex_02_03
├── ex_03_01
├── ex_03_02
├── ex_04_06
├── ex_05_01
└── ex_06_05
    └── first.py
```

ex_06_05: *

```
1 str = 'X-DSPAM-Confidence: 0.8475'
2
3 ipos = str.find(':')
4 print(ipos)
5 piece = str[ipos+1:]
6 print(piece)
```

-bash — 85x24

```
pwd
.01
cd ..
```

```
ex_04_06      ex_06_05
ex_05_01      first.py
ex_06_05
ls
```

```
python3 ex_06_05.py
python3 ex_06_05.py
python3 ex_06_05.py
python3 ex_06_05.py
```

```
python3 ex_06_05.py
```

now I can just see if value equals
float(piece) because piece is a string.

6:17

... H 64

The screenshot shows a Mac OS X desktop environment. In the top menu bar, the following items are visible: Terminal, Shell, Edit, View, Window, Help. The system status bar in the top right corner shows the date as "Wed 7:23 AM".

The main window contains two panes:

- Left Pane (File Browser):** A Finder-style interface showing a directory structure under "py4e". The current folder is "ex_06_05". Inside, there are several subfolders (ex_02_02, ex_02_03, ex_03_01, ex_03_02, ex_04_06, ex_05_01, ex_06_05) and a file named "first.py".
- Right Pane (Terminal):** A terminal window titled "ex_06_05 — bash — 65x24". It displays the output of running the "ex_06_05.py" script multiple times. The output includes:
 - "Exercise 6.5"
 - "X-DSPAM-Confidence: 0.8475"
 - "18"
 - "0.8475"

At the bottom of the screen, there is a dock with icons for various applications, including Mail, Finder, Safari, and others.

6:17 ... ⌂ 4G H 64

A screenshot of a Mac OS X desktop environment. In the center is a Terminal window titled "ex_06_05.py — /Users/py4e/Desktop/py4e". The terminal shows the execution of a Python script named "ex_06_05.py" which prints the value of the "X-DSPAM-Confidence" header from an email message. The output shows the confidence score being printed multiple times. To the left of the terminal is a file browser window showing a directory structure under "py4e" containing various Python files like "ex_02_02", "ex_02_03", etc., and a file named "first.py". The status bar at the bottom of the screen indicates "ex_06_05/ex_06_05.py 9:17" and "then end is optional." A text overlay "and that would actually work, right?" is visible at the bottom of the screen.

```
str = 'X-DSPAM-Confidence: 0.8475'
ipos = str.find(':')
print(ipos)
piece = str[ipos+2:]
print(piece)
value = float(piece)
print(value)
print(value+42.0)
```

```
m-c02m92uxfd57:ex_06_05 py4e$ python3 ex_06_05.py
X-DSPAM-Confidence:0.8475
m-c02m92uxfd57:ex_06_05 py4e$ python3 ex_06_05.py
18
m-c02m92uxfd57:ex_06_05 py4e$ python3 ex_06_05.py
18
: 0.8475
m-c02m92uxfd57:ex_06_05 py4e$ python3 ex_06_05.py
18
0.8475
0.8475
0.8475
m-c02m92uxfd57:ex_06_05 py4e$ python3 ex_06_05.py
18
0.8475
0.8475
0.8475
0.8475
42.8475
m-c02m92uxfd57:ex_06_05 py4e$
```

File Processing

A text file can be thought of as a sequence of lines

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
Return-Path: <postmaster@collab.sakaiproject.org>
Date: Sat, 5 Jan 2008 09:12:18 -0500
To: source@collab.sakaiproject.org
From: stephen.marquard@uct.ac.za
Subject: [sakai] svn commit: r39772 - content/branches/
Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772
```

<http://www.py4e.com/code/mbox-short.txt>

It's some characters,
and then a newline.

Opening a File

- Before we can read the contents of the file, we must tell Python which file we are going to work with and what we will be doing with the file
- This is done with the `open()` function
- `open()` returns a “**file handle**” - a variable used to perform operations on the file
- Similar to “File -> Open” in a Word Processor

And so before we can work
with a file inside Python,

Using open()

- handle = `open(filename, mode)` `fhand = open('mbox.txt', 'r')`
- returns a handle use to manipulate the file
- filename is a string
- mode is optional and should be 'r' if we are planning to read the file and 'w' if we are going to write to the file

It's just making the file available to the code that we're going to write.

Using open()

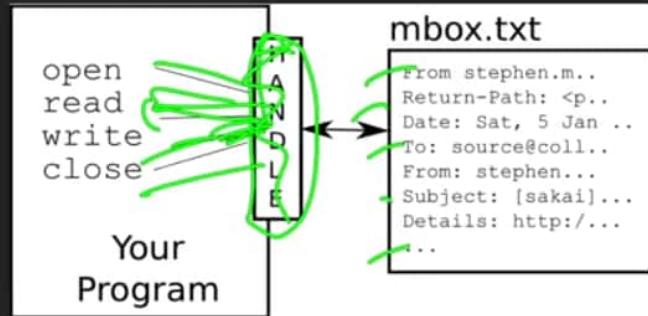
- handle = `open(filename, mode)` fhand = `open('mbox.txt', 'r')`
- returns a handle use to manipulate the file
- filename is a string
- mode is optional and should be 'r' if we are planning to read the file and 'w' if we are going to write to the file

If we leave this out,
it's going to be read.



What is a Handle?

```
>>> fhand = open('mbox.txt')
>>> print(fhand)
<_io.TextIOWrapper name='mbox.txt' mode='r' encoding='UTF-8'>
```



It says this is a file, and it's a thing.



When Files are Missing

```
>>> fhand = open('stuff.txt')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or
directory: 'stuff.txt'
```

that's not there, not surprisingly,
we get a traceback, okay?



The newline Character

- We use a special character called the “newline” to indicate when a line ends
- We represent it as `\n` in strings
- **Newline** is still one character - not two

```
>>> stuff = 'Hello\nWorld!'
>>> stuff
'Hello\nWorld!'
>>> print(stuff)
Hello
World!
>>> stuff = 'X\nY'
>>> print(stuff)
X
Y
>>> len(stuff)
3
```

is what's called the newline character.

File Processing

A text file has **newlines** at the end of each line

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008\nReturn-Path: <postmaster@collab.sakaiproject.org>\nDate: Sat, 5 Jan 2008 09:12:18 -0500\nTo: source@collab.sakaiproject.org\nFrom: stephen.marquard@uct.ac.za\nSubject: [sakai] svn commit: r39772 - content/branches/\n\nDetails: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772\n
```

It's like hitting the Enter button.



File Handle as a Sequence

- A **file handle** open for read can be treated as a **sequence** of strings where each line in the file is a string in the sequence
- We can use the **for** statement to iterate through a **sequence**
- Remember - a **sequence** is an ordered set

```
xfile = open('mbox.txt')
for cheese in xfile:
    print(cheese)
```

Now this is not the same as putting a string there, the for loop is smart.

File Handle as a Sequence

- A **file handle** open for read can be treated as a **sequence** of strings where each line in the file is a string in the sequence
- We can use the **for** statement to iterate through a **sequence**
- Remember - a **sequence** is an ordered set

```
xfile = open('mbox.txt')
for cheese in xfile:
    print(cheese)
```

This is some kind of a sequence of things.

File Handle as a Sequence

- A **file handle** open for read can be treated as a **sequence** of strings where each line in the file is a string in the sequence
- We can use the **for** statement to iterate through a **sequence**
- Remember - a **sequence** is an ordered set

```
xfile = open('mbox.txt')
for cheese in xfile:
    print(cheese)
```

So a file handle to the for loop looks like a sequence of lines.

File Handle as a Sequence

- A **file handle** open for read can be treated as a sequence of strings where each line in the file is a string in the sequence
- We can use the **for** statement to iterate through a **sequence**
- Remember - a **sequence** is an ordered set

So again, the for loop is going to run this code multiple times where the iteration

```
xfile = open('mbox.txt')
for cheese in xfile:
    print(cheese)
```

File Handle as a Sequence

- A **file handle** open for read can be treated as a **sequence** of strings where each line in the file is a string in the sequence
- We can use the **for** statement to iterate through a **sequence**
- Remember - a **sequence** is an ordered set

```
xfile = open('mbox.txt')
for cheese in xfile:
    print(cheese)
```

variable, cheese in this case,
is going to take on the successive lines.

6:38

... ⊞ 4G H 63

A middle-aged man with a beard and mustache, wearing a pink button-down shirt, is gesturing with his right hand while speaking. He appears to be in a dark room or studio setting.

If this file has 10 lines,
this loop is going to run 10 times.



6:38

... ⊞ H 63



Cheese is going to be the first line,
the second line, third line.

M

Counting Lines in a File

- Open a `file` read-only
- Use a `for` loop to read each line
- Count the lines and print out the number of lines

```
fhand = open('mbox.txt')
count = 0
for line in fhand:
    count = count + 1
print('Line Count:', count)

$ python open.py
Line Count: 132045
```

So, we set the counter to 0, and then we loop through each line, or

Reading the *Whole* File

We can **read** the whole file (newlines and all) into a **single string**

```
>>> fhand = open('mbox-short.txt')
>>> inp = fhand.read()
>>> print(len(inp))
94626
>>> print(inp[:20])
From stephen.marquar
```

And so this time, we will read the whole thing in with .read.

Reading the *Whole* File

We can **read** the whole file (newlines and all) into a **single string**

```
>>> fhand = open('mbox-short.txt')
>>> inp = fhand.read()
>>> print(len(inp))
94626
>>> print(inp[:20])
From stephen.marquar
```

Now the thing about this is,
it doesn't split it into lines.

Reading the *Whole* File

We can **read** the whole file (newlines and all) into a **single string**

```
>>> fhand = open('mbox-short.txt')
>>> inp = fhand.read()
>>> print(len(inp))
94626
>>> print(inp[:20])
From stephen.marquar
```

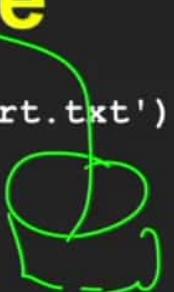


It actually just reads all the stuff with a newline, all the stuff with a newline,

Reading the *Whole* File

We can **read** the whole file (newlines and all) into a **single string**

```
>>> fhand = open('mbox-short.txt')
>>> inp = fhand.read()
>>> print(len(inp))
94626
>>> print(inp[:20])
From stephen.marquar
```

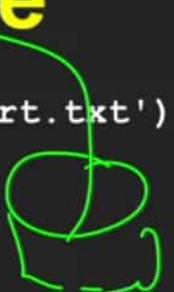


Well, we got 94626 characters in this case.

Reading the *Whole* File

We can **read** the whole file (newlines and all) into a **single string**

```
>>> fhand = open('mbox-short.txt')
>>> inp = fhand.read()
>>> print(len(inp))
94626
>>> print(inp[:20])
From stephen.marquar
```

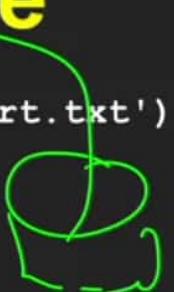


up to but
not including the 20th character.

Reading the *Whole* File

We can **read** the whole file (newlines and all) into a **single string**

```
>>> fhand = open('mbox-short.txt')
>>> inp = fhand.read()
>>> print(len(inp))
94626
>>> print(inp[:20])
From stephen.marquar
```



So that's really the first 20 characters that we would see

Searching Through a File

We can put an `if` statement in our `for` loop to only print lines that meet some criteria

```
fhand = open('mbox-short.txt')
for line in fhand:
    if line.startswith('From:'):
        print(line)
```

And then we're going to ask if the line starts with From: print.

OOPS!



What are all these blank lines doing here?

- Each line from the file has a **newline** at the end
- The print statement adds a **newline** to each line

```
From: stephen.marquard@uct.ac.za\n\nFrom: louis@media.berkeley.edu\n\nFrom: zqian@umich.edu\n\nFrom: rjlowe@iupui.edu\n\n...
```

this is the newline that was added by the print statement.

Searching Through a File (fixed)

- We can strip the whitespace from the right-hand side of the string using `rstrip()` from the string library
- The newline is considered “white space” and is stripped

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if line.startswith('From:'):
        print(line)
```

From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
....

called `rstrip`,
which strips off whitespace,

Skipping with Continue

We can conveniently skip a line by using the **continue** statement

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if not line.startswith('From:'):
        continue
    print(line)
```

Goes back to the top of the loop, so

Using `in` to Select lines

We can look for a string anywhere `in` a `line` as our selection criteria

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if not '@uct.ac.za' in line :
        continue
    print(line)
```

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
X-Authentication-Warning: set sender to stephen.marquard@uct.ac.za using -f
From: stephen.marquard@uct.ac.za
Author: stephen.marquard@uct.ac.za
From david.horwitz@uct.ac.za Fri Jan  4 07:02:32 2008
X-Authentication-Warning: set sender to david.horwitz@uct.ac.za using -f...
```

have `uct`, and print those lines out.

Prompt for File Name

```
fname = input('Enter the file name: ')
fhand = open(fname)
count = 0
for line in fhand:
    if line.startswith('Subject:'):
        count = count + 1
print('There were', count, 'subject lines in', fname)
```

→ Enter the file name: mbox.txt
→ There were 1797 subject lines in mbox.txt

→ Enter the file name: mbox-short.txt
→ There were 27 subject lines in mbox-short.txt

we prompt for the name of the file on
input, and then we open that file instead.



Bad File Names

```
fname = input('Enter the file name: ')
try:
    fhand = open(fname)
except:
    print('File cannot be opened:', fname)
    quit()

count = 0
for line in fhand:
    if line.startswith('Subject:'):
        count = count + 1
print('There were', count, 'subject lines in', fname)
```

Enter the file name: mbox.txt

There were 1797 subject lines in mbox.txt

Enter the file name: na na boo boo

File cannot be opened: na na boo boo

So we don't know if this
is going to work or not.



Bad File Names

```
fname = input('Enter the file name: ')
try:
    fhand = open(fname)
except:
    print('File cannot be opened:', fname)
    quit()
count = 0
for line in fhand:
    if line.startswith('Subject:'):
        count = count + 1
print('There were', count, 'subject lines in', fname)
```

Enter the file name: mbox.txt

There were 1797 subject lines in mbox.txt

Enter the file name: na na boo boo

File cannot be opened: na na boo boo

If it fails, if something goes bad here,
it jumps to the except block.



12:18

85

Chrome File Edit View History Bookmarks People Window Help

PY4E - Python for Everybody Chapter 7 | Python For Everyone

https://books.trinket.io/pfe/07-files.html

Python for Everyone Chapters Files

A sequence of characters stored in permanent storage like a hard drive.

Exercises

Exercise 1: Write a program to read through a file and print the contents of the file (line by line) all in upper case.

Executing the program will look as follows:

```
python shout.py
Enter a file name: mbox-short.txt
FROM STEPHEN.MARQUARD@UCT.AC.ZA SAT JAN 5 09:14:16 2008
RETURN-PATH: <POSTMASTER@COLLAB.SAKAIPROJECT.ORG>
RECEIVED: FROM MURDER (MAIL.UMICH.EDU [141.211.14.90])
    BY FRANKENSTEIN.MAIL.UMICH.EDU (CYRUS V2.3.8) WITH LMTPA;
    SAT, 05 JAN 2008 09:14:16 -0500
```

You can download the file from

www.pythontutorial.net/code3/mbox-short.txt

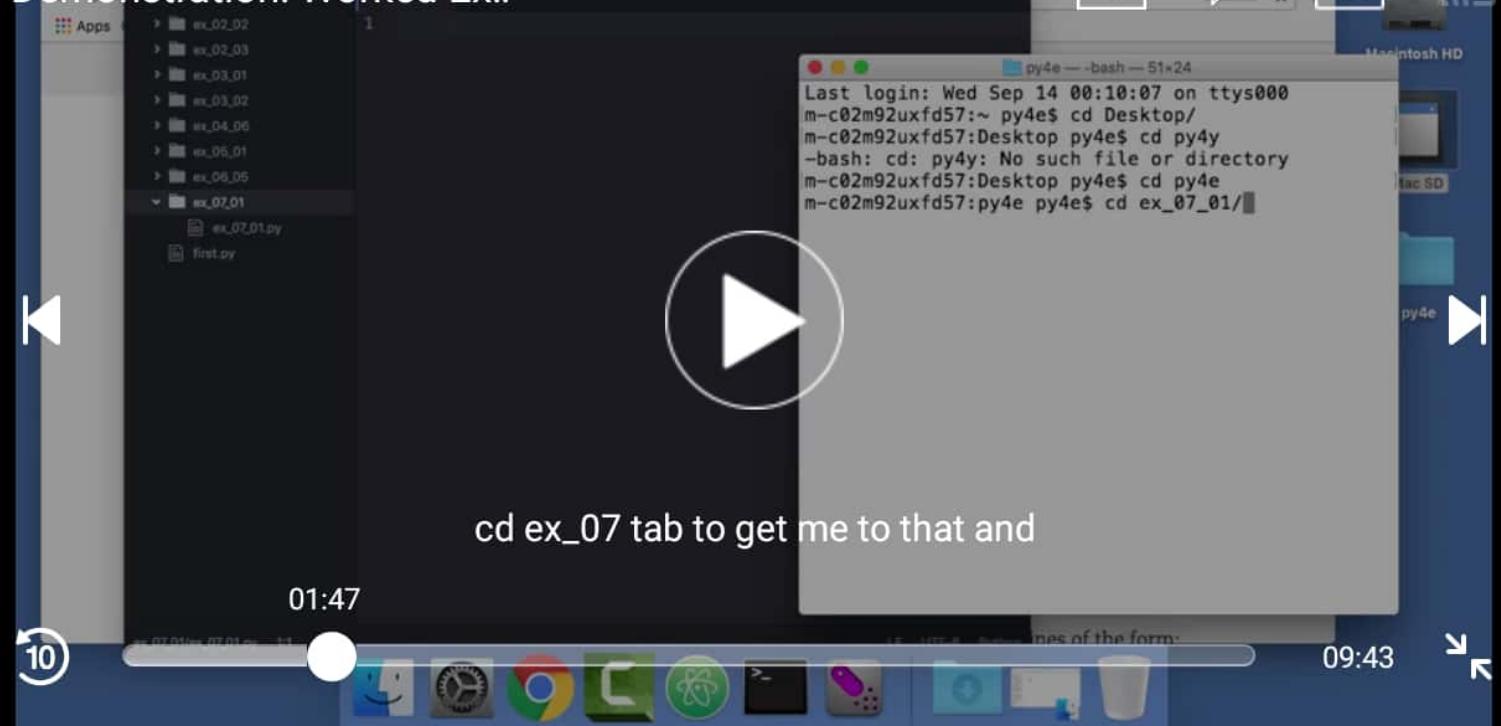
Exercise 2: Write a program to prompt for a file name, and then read through the file and look for lines of the form:



12:20

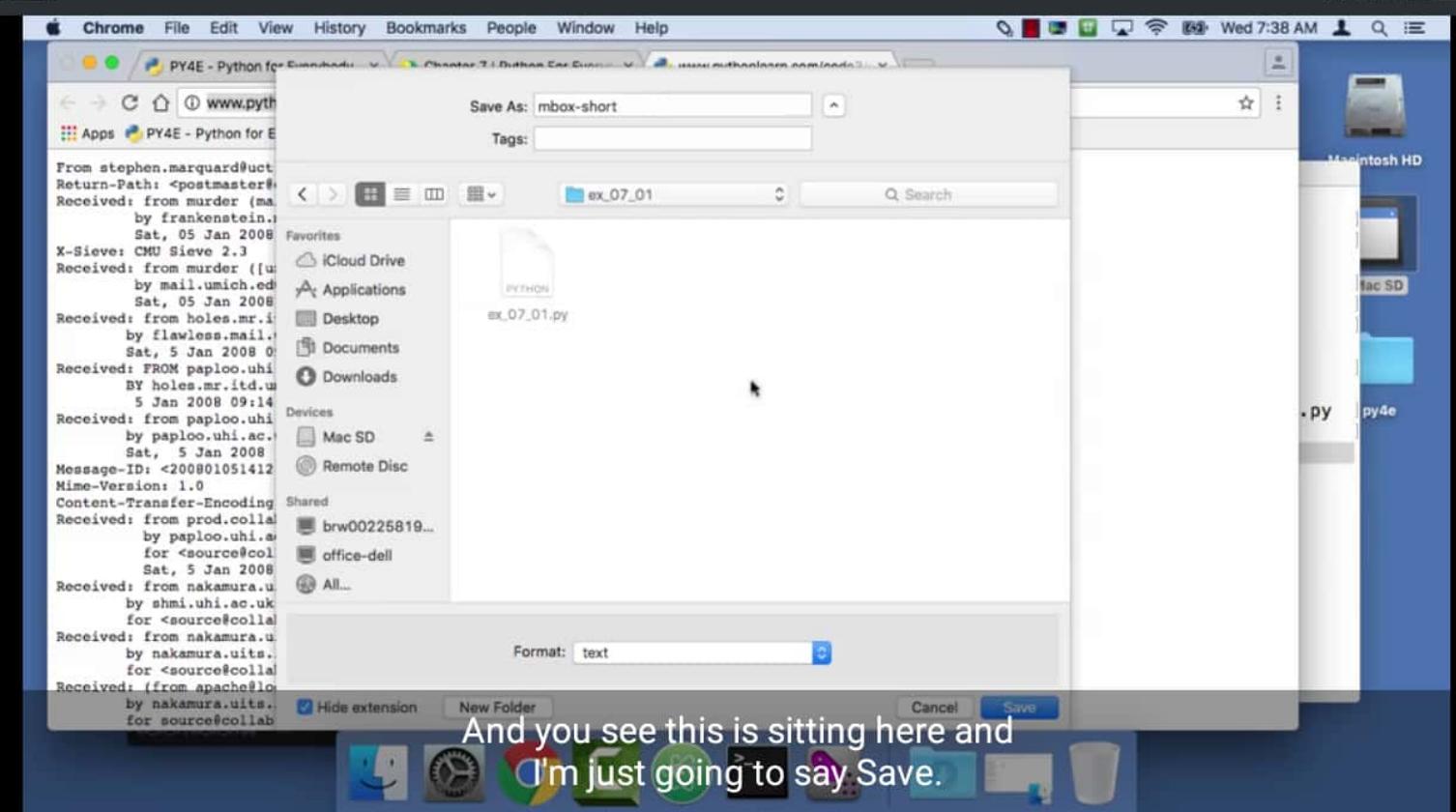
85

Demonstration: Worked Ex..



cd ex_07 tab to get me to that and

12:22



12:23

84

Demonstration: Worked Ex...

Apps PY4E - Python for Everybody

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
Return-Path: <postmaster@collab.sakaiproject.org>
Received: from murder (mail.umich.edu [141.211.14.90])
    by frankenstein.mail.umich.edu (Cyrus v2.3.8) with LMTPA;
    Sat, 05 Jan 2008 09:14:16 -0500
X-Sieve: CMU Sieve 2.3
Received: from murder ({unix socket})
    by mail.umich.edu (Cyrus v2.2.12) with LMTPA;
    Sat, 05 Jan 2008 09:14:16 -0500
Received: from holes.mr.itd.umich.edu (holes.mr.itd.umich.edu [141.211.14.79])
    by flawless.mail.umich.edu () with ESMTP id m05EEFR1013674;
    Sat, 5 Jan 2008 09:14:15 -0500
Received: FROM paploo.uhi.ac.uk (appl.prod.collab.uhi.ac.uk [194.35.219.184])
    BY holes.mr.itd.umich.edu ID 477F90B0.2DB2F12494 ;
    5 Jan 2008 09:14:10 -0500
Received: from paploo.uhi.ac.uk (localhost [127.0.0.1])
    by paploo.uhi.ac.uk (Postfix) with ESMTP id 5F919BC2F2;
    Sat, 5 Jan 2008 14:10:05 +0000 (GMT)
Message-ID: <200801051412.m05ECIaH010327@nakamura.uits.iupui.edu>
Mime-Version: 1.0
Content-Transfer-Encoding: 7bit
Received: from prod.collab.uhi.ac.uk ([194.35.219.182])
    by paploo.uhi.ac.uk (JAMES SMTP Server 2.1.3) with SMTP ID 899
    for <source@collab.sakaiproject.org>;
    Sat, 5 Jan 2008 14:09:50 +0000 (GMT)
Received: from nakamura.uits.iupui.edu (nakamura.uits.iupui.edu [127.0.0.1])
    by shmi.uhi.ac.uk (Postfix) with ESMTP id A275243007
    for <source@collab.sakaiproject.org>;
Received: from nakamura.uits.iupui.edu (localhost [127.0.0.1])
    by nakamura.uits.iupui.edu (8.12.11.20060308/8.12.11) with ESMTP id m05ECJV
    for <source@collab.sakaiproject.org>; Sat, 5 Jan 2008 09:12:19 -0500
```

03:50

```
Last login: Wed Sep 14 00:10:07 on ttys000
m-c02m92uxfd57:~ py4e$ cd Desktop/
m-c02m92uxfd57:Desktop py4e$ cd py4y
-bash: cd: py4y: No such file or directory
m-c02m92uxfd57:Desktop py4e$ cd py4e
m-c02m92uxfd57:py4e py4e$ cd ex_07_01/
m-c02m92uxfd57:ex_07_01 py4e$ pwd
/Users/py4e/Desktop/py4e/ex_07_01
m-c02m92uxfd57:ex_07_01 py4e$ ls
ex_01.py
m-c02m92uxfd57:ex_07_01 py4e$ python3 ex_07_01.py
m-c02m92uxfd57:ex_07_01 py4e$ pwd
/Users/py4e/Desktop/py4e/ex_07_01
m-c02m92uxfd57:ex_07_01 py4e$ ls
ex_01.py      mbox-short.txt
m-c02m92uxfd57:ex_07_01 py4e$
```



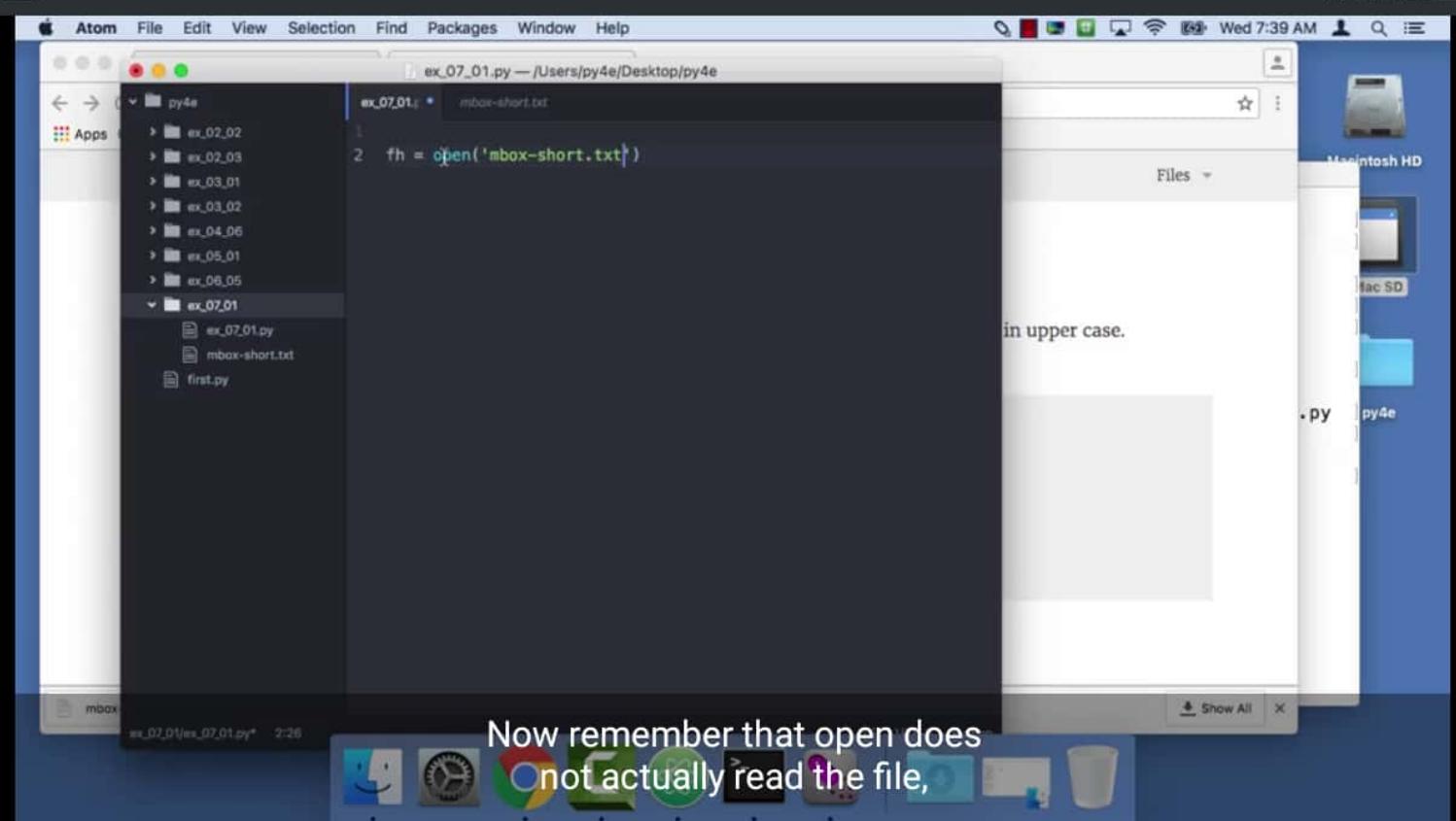
I have saved the file into the exact same folder that I've got the code for 7_1.

10

09:43

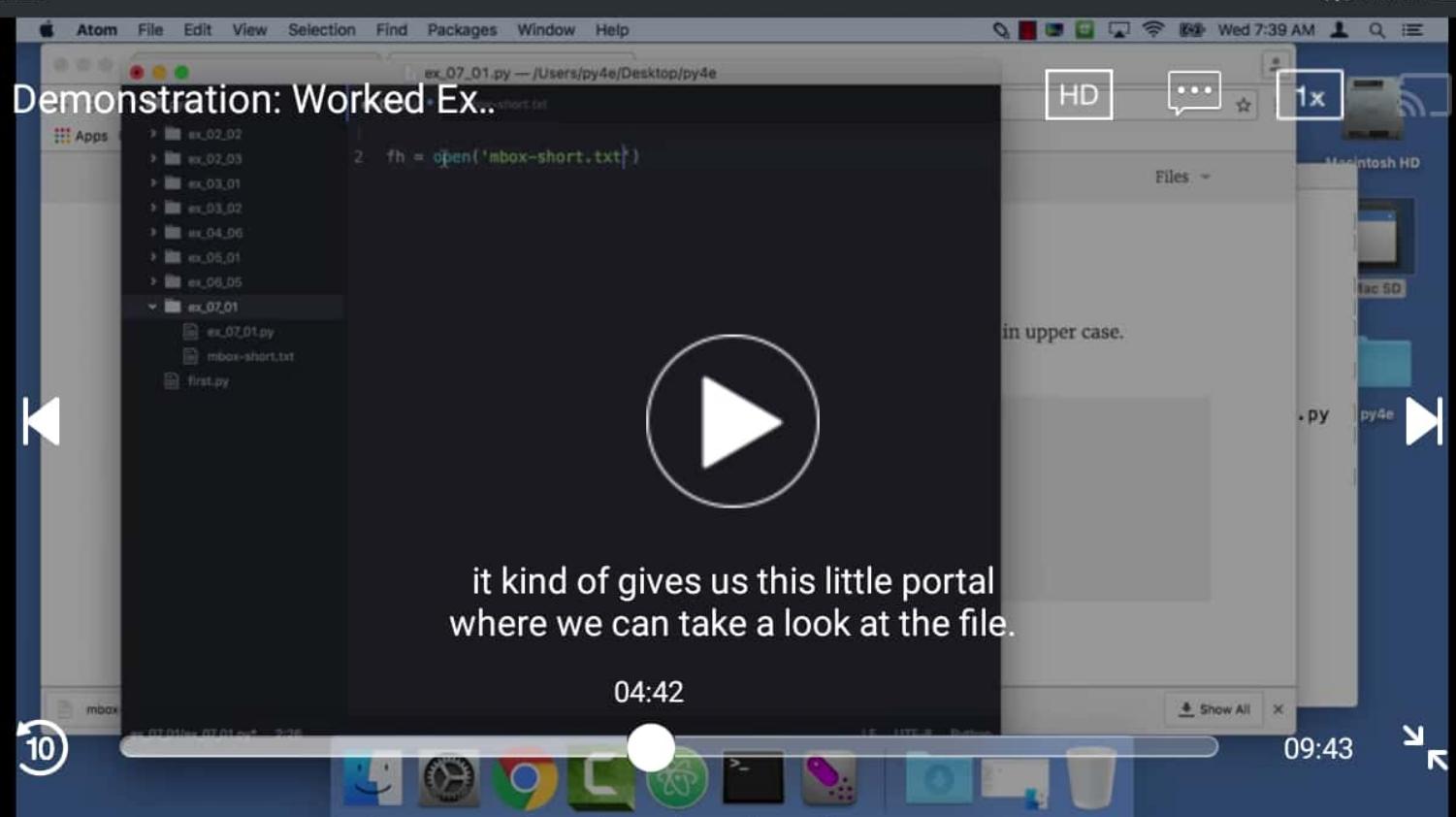
12:25

84



12:25

84



12:26

Ω 4G H 84%

The screenshot shows a Mac OS X desktop environment. In the foreground, a Terminal window is open with the title "ex_07_01 — -bash — 51x24". The terminal is displaying a command-line session where a user is navigating through their home directory and executing a Python script named "ex_07_01.py". The session includes the user's last login information and the execution of the script, which prints the contents of a file named "mbox-short.txt".

```
Last login: Wed Sep 14 00:10:07 on ttys000
m-c02m92uxfd57:~ py4e$ cd Desktop/
m-c02m92uxfd57:Desktop py4e$ cd py4y
-bash: cd: py4y: No such file or directory
m-c02m92uxfd57:Desktop py4e$ cd py4e
m-c02m92uxfd57:py4e py4e$ cd ex_07_01/
m-c02m92uxfd57:ex_07_01 py4e$ pwd
/Users/py4e/Desktop/py4e/ex_07_01
m-c02m92uxfd57:ex_07_01 py4e$ ls
ex_07_01.py
m-c02m92uxfd57:ex_07_01 py4e$ python3 ex_07_01.py
m-c02m92uxfd57:ex_07_01 py4e$ pwd
/Users/py4e/Desktop/py4e/ex_07_01
m-c02m92uxfd57:ex_07_01 py4e$ ls
ex_07_01.py    mbox-short.txt
m-c02m92uxfd57:ex_07_01 py4e$ python3 ex_07_01.py
<_io.TextIOWrapper name='mbox-short.txt' mode='r' encoding='UTF-8'>
m-c02m92uxfd57:ex_07_01 py4e$
```

In the background, a code editor window titled "ex_07_01.py — /Users/py4e/Desktop/py4e" is visible. It contains the following Python code:

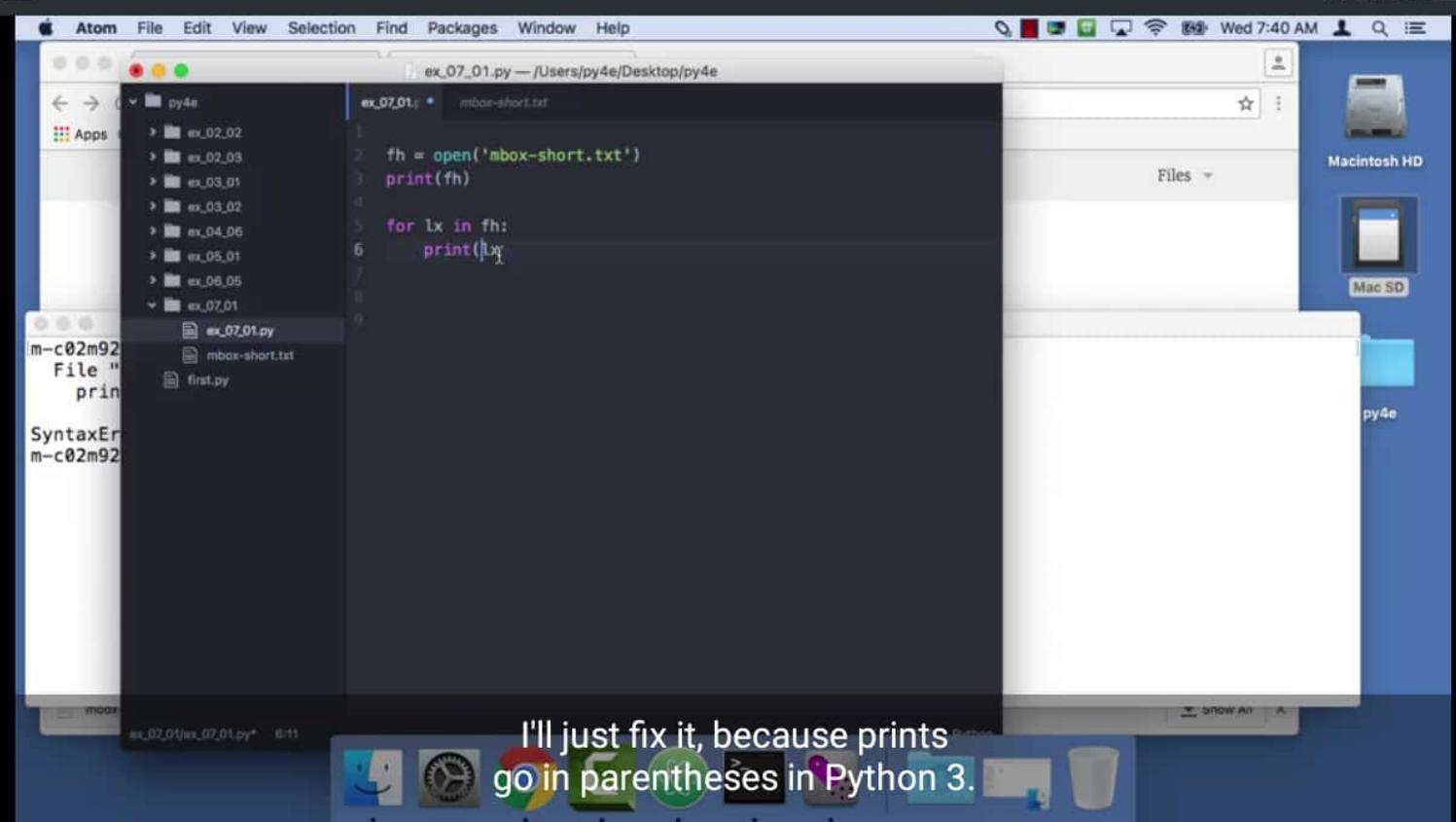
```
fh = open('mbox-short.txt')
print(fh)
for lx in fh:
    print lx
```

The status bar at the bottom of the screen shows the file path "ex_07_01/ex_07_01.py" and the line count "8:5".

Okay? So this is going to loop through every line in the file and print it out.

12:27

84



12:28

Wed 7:41 AM

Atom File Edit View Selection Find Packages Window Help

File: mbox-short.txt — /Users/py4e/Desktop/py4e

ex_07_01.py mbox-short.txt

```
1 From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
2 Return-Path: <postmaster@collab.sakaiproject.org>
3 Received: from murder (mail.umich.edu [141.211.14.90])
4 by frankenstein.mail.umich.edu (Cyrus v2.3.8) with LMTPA;
5 Sat, 05 Jan 2008 09:14:16 -0500
6 X-Sieve: CMU Sieve 2.3
7 Received: from murder ([unix socket])
8 by mail.umich.edu (Cyrus v2.2.12) with LMTPA;
9 Sat, 05 Jan 2008 09:14:16 -0500
10 Received: from holes.mr.itd.umich.edu (holes.mr.itd.umich.edu [141.211.14.79])
11 by flawless.mail.umich.edu () with ESMTP id m05EEFR1013674;
12 Sat, 5 Jan 2008 09:14:15 -0500
13 Received: FROM paploo.uhi.ac.uk (app1.prod.collab.uhi.ac.uk [194.35.219.184])
14 BY holes.mr.itd.umich.edu ID 477F90B0.2DB2F.12494 ;
15 5 Jan 2008 09:14:10 -0500
16 Received: from paploo.uhi.ac.uk (localhost [127.0.0.1])
17 by paploo.uhi.ac.uk (Postfix) with ESMTP id 5F919BC2F2;
18 Sat, 5 Jan 2008 14:10:05 +0000 (GMT)
19 Message-ID: <200801051412.m05ECIaH010327@nakamura.uits.iupui.edu>
20 Mime-Version: 1.0
21 Content-Transfer-Encoding: 7bit
22 Received: from prod.collab.uhi.ac.uk ([194.35.219.182])
23 by paploo.uhi.ac.uk (JAMES SMTP Server 2.1.3) with SMTP ID 899
24 for <source@collab.sakaiproject.org>;
25 Sat, 5 Jan 2008 14:09:50 +0000 (GMT)
```

There is a non-printing character at the end of every line called a newline which

LF UTF-8 Plain Text

m-c02m92 <_io.Text
From ste
Return-P
Received
X-Sieve:
Received

py4e ex_02_02 ex_02_03 ex_03_01 ex_03_02 ex_04_06 ex_05_01 ex_06_05 ex_07_01 ex_07_01.py mbox-short.txt first.py

12:28

Wed 7:41 AM

Atom File Edit View Selection Find Packages Window Help

mbox-short.txt — /Users/py4e/Desktop/py4e

ex_07_01.py mbox-short.txt

```
1 From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
2 Return-Path: <postmaster@collab.sakaiproject.org>
3 Received: from murder (mail.umich.edu [141.211.14.90])
4 by frankenstein.mail.umich.edu (Cyrus v2.3.8) with LMTPA;
5 Sat, 05 Jan 2008 09:14:16 -0500
6 X-Sieve: CMU Sieve 2.3
7 Received: from murder ([unix socket])
8 by mail.umich.edu (Cyrus v2.2.12) with LMTPA;
9 Sat, 05 Jan 2008 09:14:16 -0500
10 Received: from holes.mr.itd.umich.edu (holes.mr.itd.umich.edu [141.211.14.79])
11 by flawless.mail.umich.edu () with ESMTP id m05EEFR1013674;
12 Sat, 5 Jan 2008 09:14:15 -0500
13 Received: FROM paploo.uhi.ac.uk (app1.prod.collab.uhi.ac.uk [194.35.219.184])
14 BY holes.mr.itd.umich.edu ID 477F90B0.2DB2F.12494 ;
15 5 Jan 2008 09:14:10 -0500
16 Received: from paploo.uhi.ac.uk (localhost [127.0.0.1])
17 by paploo.uhi.ac.uk (Postfix) with ESMTP id 5F919BC2F2;
18 Sat, 5 Jan 2008 14:10:05 +0000 (GMT)
19 Message-ID: <200801051412.m05ECIaH010327@nakamura.uits.iupui.edu>
20 Mime-Version: 1.0
21 Content-Transfer-Encoding: 7bit
22 Received: from prod.collab.uhi.ac.uk ([194.35.219.182])
23 by paploo.uhi.ac.uk (JAMES SMTP Server 2.1.3) with SMTP ID 899
24 for <source@collab.sakaiproject.org>;
25 Sat, 5 Jan 2008 14:09:50 +0000 (GMT)
```

is the way in files we store the fact
that it goes back to the beginning.

12:30

84

Atom File Edit View Selection Find Packages Window Help

ex_07_01.py — /Users/py4e/Desktop/py4e

```
1 fh = open('mbox-short.txt')
2
3 for lx in fh:
4     ly = lx.rstrip()
5     print(ly.upper())
6
```

Content-
X-DSPAM-
X-DSPAM-
X-DSPAM-
X-DSPAM-

Details:

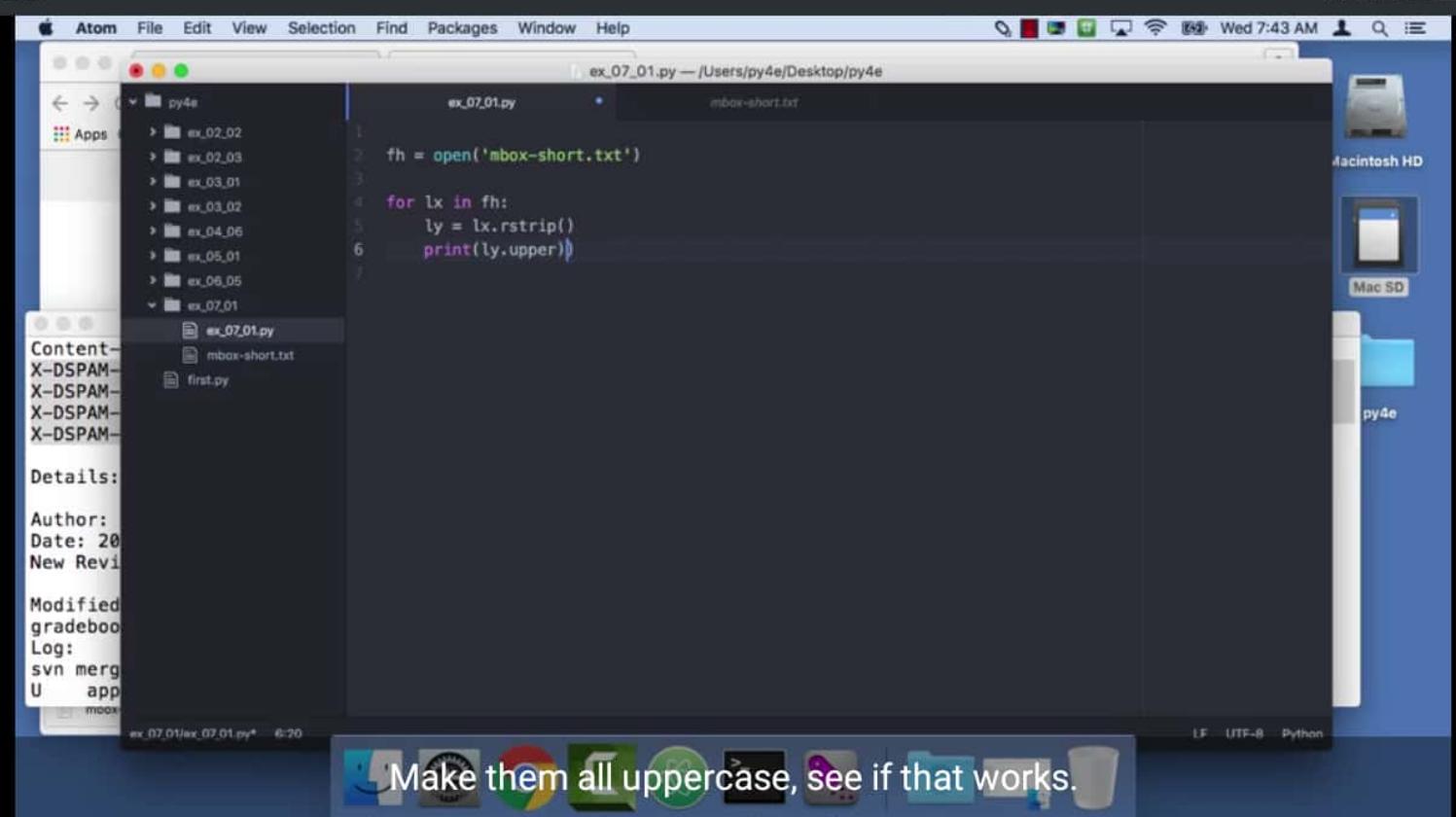
Author:
Date: 20
New Revi

Modified
gradeboo
Log:
svn merg
U app
mbox

ex_07_01/ex_07_01.py* 6:20

LF UTF-8 Python

Make them all uppercase, see if that works.



12:31

Wed 7:43 AM

This is a string variable and upper is
a method within a string variable that

The screenshot shows the Atom code editor on a Mac OS X desktop. The main window displays a Python script named 'ex_07_01.py' with the following code:

```
1 fh = open('mbox-short.txt')
2
3 for lx in fh:
4     ly = lx.rstrip()
5     print(ly.upper())
6
```

To the right of the code editor is a preview pane showing the contents of the 'mbox-short.txt' file, which contains several lines of text starting with 'RECEIVED'. Below the code editor, the status bar shows the file path 'ex_07_01/ex_07_01.py', the line count '6:21', and the character count '(1, 7)'. The bottom of the screen features the macOS Dock with various application icons.

12:31

84

Atom File Edit View Selection Find Packages Window Help

ex_07_01.py — /Users/py4e/Desktop/py4e

py4e

ex_02_02
ex_02_03
ex_03_01
ex_03_02
ex_04_06
ex_05_01
ex_06_05
ex_07_01

ex_07_01.py mbox-short.txt

```
1 fh = open('mbox-short.txt')
2
3 for lx in fh:
4     ly = lx.rstrip()
5     print(ly.upper())
6
```

m-c02m92
FROM STE
RETURN-P
RECEIVED

X-SIEVE:
RECEIVED

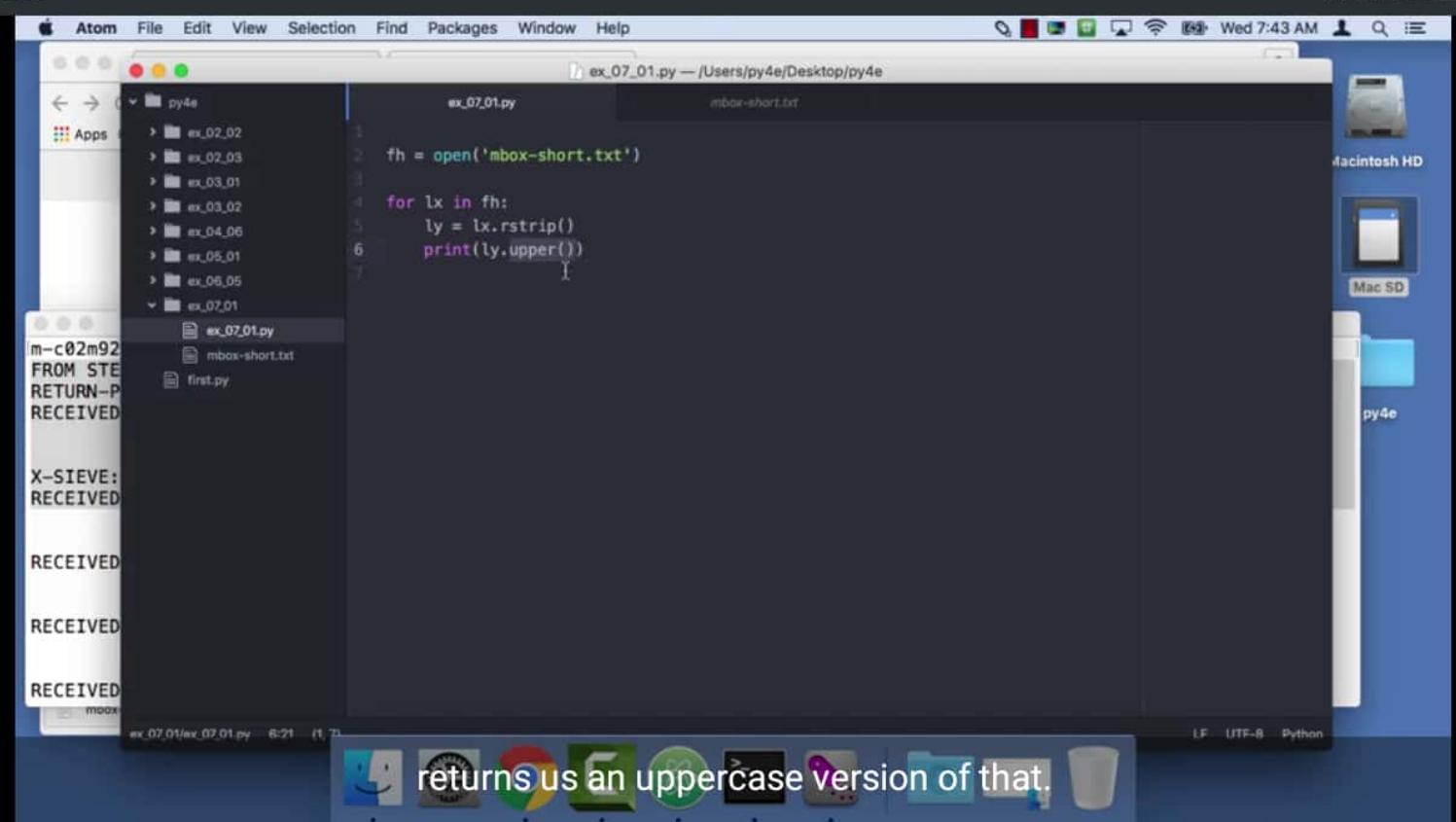
RECEIVED

RECEIVED

RECEIVED

ex_07_01/ex_07_01.py 6:21 (1, 72) LF UTF-8 Python

returns us an uppercase version of that,



Programming

Algorithms

- A set of rules or steps used to solve a problem

Data Structures

- A particular way of organizing data in a computer

<https://en.wikipedia.org/wiki/Algorithm>

https://en.wikipedia.org/wiki/Data_structure
talking about ways to use variables
differently.

What is Not A “Collection”?

Most of our **variables** have one value in them - when we put a new value in the **variable**, the old value is overwritten

```
$ python
>>> x = 2
>>> x = 4
>>> print(x)
4
```

Then later it says $x = 4$. It finds
that same piece of memory,

A List Is a Kind of Collection



- A **collection** allows us to put many values in a single “**variable**”
- A **collection** is nice because we can carry many values around in one convenient package.

```
friends = [ 'Joseph', 'Glenn', 'Sally' ]
```

```
carryon = [ 'socks', 'shirt', 'perfume' ]
```

And so in this we've actually,

A List Is a Kind of Collection



- A **collection** allows us to put many values in a single “**variable**”
- A **collection** is nice because we can carry many values around in one convenient package.

```
friends = [ 'Joseph', 'Glenn', 'Sally' ]  
carryon = [ 'socks', 'shirt', 'perfume' ]
```

This is a list. Square brackets is a list constant.

A List Is a Kind of Collection



- A **collection** allows us to put many values in a single “**variable**”
- A **collection** is nice because we can carry many values around in one convenient package.

```
friends = [ 'Joseph', 'Glenn', 'Sally' ]
```

```
carryon = [ 'socks', 'shirt', 'perfume' ]
```

It's a list of strings, which is different than a string.

List Constants

- List constants are surrounded by square brackets and the elements in the list are separated by commas
- A list element can be any Python object - even another list
- A list can be empty

```
>>> print([1, 24, 76])  
[1, 24, 76]  
>>> print(['red', 'yellow',  
'blue'])  
['red', 'yellow', 'blue']  
>>> print(['red', 24, 98.6])  
['red', 24, 98.6]  
>>> print([1, [5, 6], 7])  
[1, [5, 6], 7]  
>>> print([])  
[]
```

So like I said, list constants are anything in square brackets.

List Constants

- List constants are surrounded by square brackets and the elements in the list are separated by commas
- A list element can be any Python object - even another list
- A list can be empty

```
>>> print([1, 24, 76])
[1, 24, 76]
>>> print(['red', 'yellow',
'blue'])
['red', 'yellow', 'blue']
>>> print(['red', 24, 98.6])
['red', 24, 98.6]
>>> print([1, [5, 6], 7])
[1, [5, 6], 7]
>>> print([])
[]
```

The first element in the list is an integer.

List Constants

- List constants are surrounded by square brackets and the elements in the list are separated by commas
- A list element can be any Python object - even another list
- A list can be empty

```
>>> print([1, 24, 76])
[1, 24, 76]
>>> print(['red', 'yellow',
'blue'])
['red', 'yellow', 'blue']
>>> print(['red', 24, 98.6])
['red', 24, 98.6]
>>> print([1, [5, 6], 7])
[1, [5, 6], 7]
>>> print([])
[]
```

The second element in the list is another list.

List Constants

- List constants are surrounded by square brackets and the elements in the list are separated by commas
- A list element can be any Python object - even another list
- A list can be empty

```
>>> print([1, 24, 76])
[1, 24, 76]
>>> print(['red', 'yellow',
'blue'])
['red', 'yellow', 'blue']
>>> print(['red', 24, 98.6])
['red', 24, 98.6]
>>> print([1, [5, 6], 7])
[1, [5, 6], 7]
>>> print([])
[]
```

And so there are actually three elements in,



Looking Inside Lists

Just like strings, we can get at any single element in a list using an index specified in square brackets

Joseph	Glenn	Sally
0	1	2

```
>>> friends = [ 'Joseph', 'Glenn', 'Sally' ]  
>>> print(friends[1])  
Glenn  
>>>
```

So this is friends sub 1.

Lists Are Mutable

- Strings are “**immutable**” - we cannot change the contents of a string - we must make a **new string** to make any change
- Lists are “**mutable**” - we can **change** an element of a list using the **index operator**

```
>>> fruit = 'Banana'  
>>> fruit[0] = 'b'   
Traceback  
TypeError: 'str' object does not support item assignment  
>>> x = fruit.lower()  
>>> print(x)  
banana  
>>> lotto = [2, 14, 26, 41, 63]  
>>> print(lotto)  
[2, 14, 26, 41, 63]  
>>> lotto[2] = 28  
>>> print(lotto)  
[2, 14, 28, 41, 63]
```

This seems like it would change that first letter to lowercase b.

Lists Are Mutable

- Strings are “**immutable**” - we cannot change the contents of a string - we must make a **new string** to make any change
- Lists are “**mutable**” - we can **change** an element of a list using the **index operator**

```
>>> fruit = 'Banana'  
>>> fruit[0] = 'b' —  
Traceback  
TypeError: 'str' object does not  
support item assignment  
>>> x = fruit.lower()  
>>> print(x)  
banana  
>>> lotto = [2, 14, 26, 41, 63]  
>>> print(lotto)  
[2, 14, 26, 41, 63]  
>>> lotto[2] = 28  
>>> print(lotto)  
[2, 14, 28, 41, 63]
```

Item assignment is another word that Python uses to talk about this.

How Long is a List?

- The `len()` function takes a `list` as a parameter and returns the number of `elements` in the `list`
- Actually `len()` tells us the number of elements of any set or sequence (such as a string...)

```
>>> greet = 'Hello Bob'  
>>> print(len(greet))  
9  
>>> x = [1, 2, 'joe', 99]  
>>> print(len(x))  
4  
>>>
```

How many things are in there?
Tell me about that.

Using the Range Function

- The `range` function returns a list of numbers that range from zero to one less than the parameter
- We can construct an index loop using `for` and an integer iterator

```
>>> print(range(4))
[0, 1, 2, 3]
>>> friends = ['Joseph', 'Glenn', 'Sally']
>>> print(len(friends))
3
>>> print(range(len(friends)))
[0, 1, 2]
>>>
```

loops but the range function returns
a list, returns a list.

Using the Range Function

- The `range` function returns a list of numbers that range from zero to one less than the parameter
- We can construct an index loop using `for` and an integer iterator

```
>>> print(range(4))  
[0, 1, 2, 3]  
>>> friends = ['Joseph', 'Glenn', 'Sally']  
>>> print(len(friends))  
3  
>>> print(range(len(friends)))  
[0, 1, 2]  
>>>
```

0, 1, 2, up to but not including 3.



A Tale of Two Loops...

```

friends = ['Joseph', 'Glenn', 'Sally']
for friend in friends :
    print('Happy New Year:', friend)
for i in range(len(friends)) :
    friend = friends[i]
    print('Happy New Year:', friend)
    
```

```

>>> friends = ['Joseph', 'Glenn', 'Sally']
>>> print(len(friends))
3
>>> print(range(len(friends)))
[0, 1, 2]
    
```

Happy New Year: Joseph
 Happy New Year: Glenn
 Happy New Year: Sally



And so then *i* is going to go through 0, 1, and 2.

8.2 - Manipulating Lists

HD

...

1.25



Concatenating Lists Using +

We can create a new list by adding two existing lists together



```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print(c)
[1, 2, 3, 4, 5, 6]
>>> print(a)
[1, 2, 3]
```

So the first thing that we see here is that we can add them together and

00:17

10

09:13



Lists Can Be Sliced Using :

```
>>> t = [9, 41, 12, 3, 74, 15]
>>> t[1:3]
[41, 12]
>>> t[:4]
[9, 41, 12, 3]
>>> t[3:]
[3, 74, 15]
>>> t[:]
[9, 41, 12, 3, 74, 15]
```

Remember: Just like in strings, the second number is “up to but not including”

.

Slicing operator, again think of strings.

Lists Can Be Sliced Using :

```
>>> t = [9, 41, 12, 3, 74, 15]
>>> t[1:3]
[41, 12]
>>> t[:4]
[9, 41, 12, 3]
>>> t[3:]
[3, 74, 15]
>>> t[:]
[9, 41, 12, 3, 74, 15]
```

Remember: Just like in strings, the second number is “up to but not including”

3 to the end so 0, 1, 2, 3 that's 3 to the end so that's 3, 74, and 15.



Lists Can Be Sliced Using :

```
>>> t = [9, 41, 12, 3, 74, 15]
>>> t[1:3]
[41, 12]
>>> t[:4]
[9, 41, 12, 3]
>>> t[3:]
[3, 74, 15]
>>> t[:]
[9, 41, 12, 3, 74, 15]
```

Remember: Just like in strings, the second number is “up to but not including”

3 to the end so 0, 1, 2, 3 that's 3 to the end so that's 3, 74, and 15.



List Methods

```
>>> x = list()
>>> type(x)
<type 'list'>
>>> dir(x)
['append', 'count', 'extend', 'index', 'insert',
 'pop', 'remove', 'reverse', 'sort']
>>>
```

<http://docs.python.org/tutorial/datastructures.html>

they're all very well documented
in the Python documentation.

Building a List from Scratch

- We can create an empty `list` and then add elements using the `append` method
- The `list` stays in order and new elements are `added` at the end of the `list`

```
>>> stuff = list() ✓  
>>> stuff.append('book')  
>>> stuff.append(99)  
>>> print(stuff)  
['book', 99]  
>>> stuff.append('cookie')  
>>> print(stuff)  
['book', 99, 'cookie']
```

We're calling this, this is constructor
and so list is a predefined type.



Building a List from Scratch

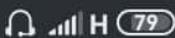
- We can create an empty `list` and then add elements using the `append` method
- The `list` stays in order and new elements are `added` at the end of the `list`

```
>>> stuff = list()      []
>>> stuff.append('book')
>>> stuff.append(99)
>>> print(stuff)
['book', 99]
>>> stuff.append('cookie')
>>> print(stuff)
['book', 99, 'cookie']
```

and then assign it into `stuff`. So that is a list with nothing in it, so



14:00



Lists - Part 2

PYTHON FOR
EVERYBODY

8.2 - Manipulating Lists

HD



1x



Building a List from Scratch

- We can create an empty `list` and then add elements using the `append` method
- The `list` stays in order and new elements are `added` at the end of the `list`

that's because lists are mutable,
but not strings.

```
>>> stuff = list() [ ]
>>> stuff.append('book') [ ]
>>> stuff.append(99) [ ]
>>> print(stuff) [ ]
['book', 99]
>>> stuff.append('cookie')
>>> print(stuff)
['book', 99, 'cookie']
```

02:48

09:13

10



Is Something in a List?

- Python provides two operators that let you check if an item is in a list
- These are logical operators that return **True** or **False**
- They do not modify the list

```
>>> some = [1, 9, 21, 10, 16]
>>> 9 in some
True
>>> 15 in some
False
>>> 20 not in some
True
>>>
```

you're saying is 9 somewhere in the list?



Lists are in Order

- A **list** can hold many items and keeps those items in the order until we do something to change the order
- A **list** can be **sorted** (i.e., change its order)
- The **sort** method (unlike in strings) means “**sort yours**”

```
>>> friends = [ 'Joseph', 'Glenn', 'Sally' ]  
>>> friends.sort()  
>>> print(friends)  
['Glenn', 'Joseph', 'Sally']  
>>> print(friends[1])  
Joseph  
>>>
```

And so it's now in alphabetical order,
Glenn, Joseph, and Sally.



Built-in Functions and Lists

- There are a number of **functions** built into Python that take **lists** as parameters
- Remember the loops we built? These are much simpler.

```
>>> nums = [3, 41, 12, 9, 74, 15]
>>> print(len(nums))  ←
6
>>> print(max(nums))  ←
74
>>> print(min(nums))  ←
3
>>> print(sum(nums))  ←
154
>>> print(sum(nums)/len(nums))
25.6
```

What's the sum of this list?



14:07

Ω 4G H 78

Lists - Part 2

PYTHON FOR
EVERYBODY

```
total = 0
count = 0
while True :
    inp = input('Enter a number: ')
    if inp == 'done' : break
    value = float(inp)
    total = total + value
    count = count + 1

average = total / count
print('Average:', average)
```

Enter a number: 3 }
Enter a number: 9 }
Enter a number: 5 }
Enter a number: done
Average: 5.666666666667

```
numlist = list()
while True :
    inp = input('Enter a number: ')
    if inp == 'done' : break
    value = float(inp)
    numlist.append(value)

average = sum(numlist) / len(numlist)
print('Average:', average)
```



Best Friends: Strings and Lists

```
< />
>>> abc = 'With three words'
>>> stuff = abc.split()
>>> print(stuff)
['With', 'three', 'words']
>>> print(len(stuff))
3
>>> print(stuff[0])
With
>>> print(stuff)
['With', 'three', 'words']
>>> for w in stuff :
...     print(w)
...
With
Three
Words
>>>
```

Split breaks a string into parts and produces a list of strings. We think of these as words. We can access a particular word or loop through all the words.

```
14:25  
LISTS - Part 3  
>>> line = 'A lot           .      of spaces'  
>>> etc = line.split()  
>>> print(etc)  
['A', 'lot', 'of', 'spaces']  
>>>  
>>> line = 'first;second;third'  
>>> thing = line.split()  
>>> print(thing)  
['first;second;third']  
>>> print(len(thing))  
1  
>>> thing = line.split(';')  
>>> print(thing)  
['first', 'second', 'third']  
>>> print(len(thing))  
3  
>>>
```

- When you do not specify a **delimiter**, multiple spaces are treated like one delimiter
- You can specify what **delimiter** character to use in the **splitting**

by default on whitespace and it treats more than one space as a single space.

```
>>> line = 'A lot of spaces'  
>>> etc = line.split()  
>>> print(etc)  
['A', 'lot', 'of', 'spaces']  
>>> line = 'first;second;third'  
>>> thing = line.split()  
>>> print(thing)  
['first;second;third']  
>>> print(len(thing))  
1  
>>> thing = line.split(';')  
>>> print(thing)  
['first', 'second', 'third']  
>>> print(len(thing))  
3  
>>>
```

- When you do not specify a **delimiter**, multiple spaces are treated like one delimiter
- You can specify what **delimiter** character to use in the **splitting**

And it doesn't realize these semicolons are what we want it to split

14:27

... ⊞ H 78

EVERYBODY

```
>>> line = 'A lot of spaces'
>>> etc = line.split()
>>> print(etc)
['A', 'lot', 'of', 'spaces']
>>>
>>> line = 'first;second;third'
>>> thing = line.split()
>>> print(thing)
['first;second;third']
>>> print(len(thing))
1
>>> thing = line.split(';')
>>> print(thing)
['first', 'second', 'third']
>>> print(len(thing))
3
>>>
```

- When you do not specify a **delimiter**, multiple spaces are treated like one delimiter
- You can specify what **delimiter** character to use in the **splitting**

So when you tell it to split
on a different character,

14:30

... ⊞ H 78

LISTS - Part 3

EVERYBODY

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if not line.startswith('From ') : continue
    words = line.split()
    print(words[2])
```

Sat
Fri
Fri
Fri
...

```
>>> line = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
>>> words = line.split()
>>> print(words)
['From', 'stephen.marquard@uct.ac.za', 'Sat', 'Jan', '5', '09:14:16', '2008']
>>>
```

From space, in this case.
If it's not, continue.

14:30

... H 78

LISTS - Part 3

EVERYBODY

From stephen.marquard@uct.ac.za [Sat Jan 5 09:14:16 2008]

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if not line.startswith('From ') : continue
    words = line.split()
    print(words[2])
```

Sat
Fri
Fri
Fri
...

```
>>> line = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
>>> words = line.split()
>>> print(words)
['From', 'stephen.marquard@uct.ac.za', 'Sat', 'Jan', '5', '09:14:16', '2008']
>>>
```

And so word sub 2 is always
going to be the day of the week.

The Double Split Pattern

Sometimes we split a line one way, and then grab one of the pieces of the line and split that piece again

From `stephen.marquard@uct.ac.za` Sat Jan 5 09:14:16 2008

```
words = line.split()  
email = words[1]
```

And that's you split something and
then you split it again.

The Double Split Pattern

Sometimes we split a line one way, and then grab one of the pieces of the line and split that piece again

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
words = line.split()
email = words[1]
```

So that's not the host name,
that's the email address, okay?

14:33

LISTS - Part 3

... ⊞ H 77

EVERYBODY

The Double Split Pattern

/

From `stephen.marquard@uct.ac.za` Sat Jan 5 09:14:16 2008

```
words = line.split()
email = words[1]
pieces = email.split('@')
                           stephen.marquard@uct.ac.za
                           ['stephen.marquard', 'uct.ac.za']
```

into the `email` variable and
now we are going to split this,

14:34

... ⊞ H 77

LISTS - Mail 3

Open with the original object to draw with pen 2

The Double Split Pattern

From **stephen.marquard@uct.ac.za** Sat Jan 5 09:14:16 2008

```
words = line.split()
email = words[1]
pieces = email.split('@')
print(pieces[1])
```

stephen.marquard@uct.ac.za
['**stephen.marquard**', '**uct.ac.za**']
'uct.ac.za'.

It's not words sub 1, but it's pieces sub 1.



List Summary

- Concept of a collection
- Lists and definite loops
- Indexing and lookup
- List mutability
- Functions: len, min, max, sum
- Slicing lists
- List methods: append, remove
- Sorting lists
- Splitting strings into lists of words
- Using split to parse strings

Lists are organized sequentially with the bracket operator as the lookup operator.

14:36

Wed 4:45 PM

Chrome File Edit View History Bookmarks People Window Help

www.py4e.com/code3/mbox-short.txt

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
Return-Path: <postmaster@collab.sakaiproject.org>
Received: from murder (mail.umich.edu [141.211.14.90])
by frankenstein.mail.umich.edu (Cyrus v2.3.8) with LMTPA;
Sat, 05 Jan 2008 09:14:16 -0500
X-Sieve: CMU Sieve 2.3
Received: from murder ([unix socket])
by mail.umich.edu (Cyrus v2.2.12) with LMTPA;
Sat, 05 Jan 2008 09:14:16 -0500
Received: from holes.mr.itd.umich.edu (holes.mr.itd.umich.edu [141.211.14.79])
by flawless.mail.umich.edu () with ESMTP id m05EEFR1013674;
Sat, 5 Jan 2008 09:14:15 -0500
Received: FROM paploo.uhi.ac.uk (appl.prod.collab.uhi.ac.uk [194.35.219.184])
BY holes.mr.itd.umich.edu ID 477F90B0.2DB2F.12494 ;
5 Jan 2008 09:14:10 -0500
Received: from paploo.uhi.ac.uk (localhost [127.0.0.1])
by paploo.uhi.ac.uk (Postfix) with ESMTP id 5F919BC2F2;
Sat, 5 Jan 2008 14:10:05 +0000 (GMT)
Message-ID: <200801051412.m05ECIaH010327@nakamura.uits.iupui.edu>
Mime-Version: 1.0
Content-Transfer-Encoding: 7bit
Received: from prod.collab.uhi.ac.uk ([194.35.219.182])
by paploo.uhi.ac.uk (JAMES SMTP Server 2.1.3) with SMTP ID 899
for <source@collab.sakaiproject.org>;
Sat, 5 Jan 2008 14:09:50 +0000 (GMT)
Received: from nakamura.uits.iupui.edu (nakamura.uits.iupui.edu [134.68.220.122])
by shmi.uhi.ac.uk (Postfix) with ESMTP id A215243002
for <source@collab.sakaiproject.org>; Sat, 5 Jan 2008 14:13:33 +0000 (GMT)
Received: from nakamura.uits.iupui.edu (localhost [127.0.0.1])
by nakamura.uits.iupui.edu (8.12.11.20060308/8.12.11) with ESMTP id m05ECJVp010329
for <source@collab.sakaiproject.org>; Sat, 5 Jan 2008 09:12:19 -0500
Received: (from apache@localhost)
by nakamura.uits.iupui.edu (8.12.11.20060308/8.12.11) with ESMTP id m05ECJVp010329

lines that begin with From space,
and extract the third word.

14:37

Wed 4:46 PM

```
dow.py
1 han = open('mbox-short.txt')
2
3 for line in han:
4     line = line.rstrip()
5     wds = line.split()
6     if wds[0] != 'From':
7         continue
8     print(wds[2])
```

File: ex_08/dow.py 6:1 (2,23) LF UTF-8 Python

Received: by nakamura.uits.iupui.edu (8.12.11.20060308/8.12.11) with ESMTP id m05ECJVp010329
for <source@collab.sakaiproject.org>; Sat, 5 Jan 2008 09:12:19 -0500
by nakamura.uits.iupui.edu (8.12.11.20060308/8.12.11) with ESMTP id m05ECJVp010329
for <source@collab.sakaiproject.org>; Sat, 5 Jan 2008 09:12:19 -0500

zeroth word, the first word, is From and if
it's not we skip and read the next line.

14:38

Wed 4:47 PM

A screenshot of a Mac OS X desktop environment. In the center is a Terminal window titled 'dow.py — /Users/py4e/Desktop/py4e'. The window shows a file tree on the left and the contents of 'dow.py' on the right. The code in 'dow.py' reads:

```
1 han = open('mbox-short.txt')
2
3 for line in han:
4     line = line.rstrip()
5     wds = line.split()
6     if wds[0] == 'From':
7         print(wds[1])
8     else:
9         continue
```

The terminal output shows the script running and then displaying a traceback:

```
m-c02m92uxfd57:ex_08 py4e$ python3 dow.py
Sat
Traceback (most recent call last):
  File "dow.py", line 6, in <module>
    if wds[0] != 'From' :
IndexError: list index out of range
m-c02m92uxfd57:ex_08 py4e$
```

Below the terminal window, there is a snippet of email message content:

```
ex_08/dow.py 9:1 (2,18)
by nakamura.uits.iupui.edu (8.12.11)
for <source@collab.sakaiproject.org>
Received: (from apache@localhost)
by nakamura.uits.iupui.edu (8.12.11)
```

At the bottom of the screen, there is a decorative footer bar with various icons.

And, by now you've seen a few
tracebacks and there you go.

14:39

Wed 4:48 PM

76

Worked Exercise: Lists

```
dow.py  mbox-short.txt
han = open('mbox-short.txt')

for line in han:
    line = line.rstrip()
    wds = line.split()
    print('Words:',wds)
    if wds[0] != 'From':
        continue
    print(wds[2])
```



So that I know really when this
finally does blow up what was going on

```
ms_08/dow.py* 8:24
by nakamura.uits.iupui.edu (8.12.11.
for <source><cc>siproject.org>
Received: (from apache@localhost)
by nakamura.uits.iupui.edu (8.12.11.
```

LF UTF-8 Python

11:33

14:40

Ω 4G H 76

The screenshot shows a Mac OS X desktop environment. In the foreground, a terminal window titled "dow.py" is open, displaying Python code and its execution output. The code reads lines from a file named "mbox-short.txt" and prints them. The output shows several lines of text, including "Sat" and "Words: ['Received:', 'from', 'murder', '(mail.umich.edu', '[141.211.14.90])']". Below the terminal, a file browser window is visible, showing a directory structure under "py4e" containing files like "ex_02_02", "ex_02_03", etc., and sub-directories "DS_Store" and "mbox-short.txt". A message bar at the bottom of the screen says "prints out Saturday, which is exactly what we expect."

```
1 han = open('mbox-short.txt')
2
3 for line in han:
4     line = line.rstrip()
5     wds = line.split()
6     print(m-c02m92uxfd57:ex_08 py4e$ python3 dow.py
7 if W
8 Words: ['From', 'stephen.marquard@uct.ac.za', 'Sat', 'Jan', '5', '09:14:16', '20
9 08']
10 Sat
11 printWords: ['Return-Path:', '<postmaster@collab.sakaiproject.org>']
12 Words: ['Received:', 'from', 'murder', '(mail.umich.edu', '[141.211.14.90])']
13 Words: ['by', 'frankenstein.mail.umich.edu', '(Cyrus', 'v2.3.8)', 'with', 'LMTPA
14 ;']
15 Words: ['Sat,', '05', 'Jan', '2008', '09:14:16', '-0500']
16 Words: ['X-Sieve:', 'CMU', 'Sieve', '2.3']
17 Words: ['Received:', 'from', 'murder', '([unix', 'socket])']
18 Words: ['by', 'mail.umich.edu', '(Cyrus', 'v2.2.12)', 'with', 'LMTPA;']
19 Words: ['Sat,', '05', 'Jan', '2008', '09:14:16', '-0500']
20 Words: ['Received:', 'from', 'holes.mr.itd.umich.edu', '(holes.mr.itd.umich.edu'
21 ', [141.211.14.79])']
22 Words: ['by', 'flawless.mail.umich.edu', '()', 'with', 'ESMTP', 'id', 'm05EEFR10
23 13674;']
24 Words: ['Sat,', '5', 'Jan', '2008', '09:14:15', '-0500']
25 Words: ['Received:', 'FROM', 'paploo.uhi.ac.uk', '(app1.prod.collab.uhi.ac.uk',
26 '[194.35.219.184])']
27 Words: ['BY', 'holes.mr.itd.umich.edu', 'ID', '477F90B0.2DB2F.12494', ';']
28 Words: ['5', 'Jan', '2008', '09:14:10', '-0500']
29 Words: ['Received:', 'from', 'paploo.uhi.ac.uk', '(localhost', '[127.0.0.1])']
30 Words: ['by', 'paploo.uhi.ac.uk', '(Postfix)', 'with', 'ESMTP', 'id', '5F919BC2F
```

prints out Saturday,
which is exactly what we expect.

14:41

Ω 4G H 76

Worked Exercise: Lists

The screenshot shows a Mac OS X desktop environment. In the center is a Terminal window titled "dow.py — /Users/joedie/Desktop/py4e". The code in the terminal is:

```
dow.py mbox-short.txt
han = open('mbox-short.txt')

for line in han:
    line = line.rstrip()
    wds = line.split()
    print(wds)

    if len(wds) == 0:
        continue
    if wds[0] != 'From':
        continue
    print(wds)
    print('Ignore')
    words = []
    for word in wds[1:]:
        if word[0] == '#':
            continue
        words.append(word)
    ignore.append(words)
    print(ignore)
```

Below the terminal, a file browser window is open, showing a directory structure with files like "ex_02", "ex_02_02", "ex_02_03", etc., and sub-directories "DS_Store" and "mbox-short.txt".

A message box in the center of the screen says:

So we really now know exactly what happened before the traceback.

The status bar at the bottom right shows the date and time: "Wed 4:49 PM" and "11:33".

14:41

Wed 4:50 PM

The screenshot shows a Mac OS X desktop environment. In the foreground, a terminal window is open with the command `python mbox-short.txt`. The terminal output shows a Python script named `dow.py` reading from a file named `mbox-short.txt`. The script processes each line, strips whitespace, splits it into words, and prints them. It handles various email headers like `Subject`, `X-Content-Type-Outer-Envelope`, and `X-DSPAM-Result`. The terminal also displays a stack trace for an `IndexError` occurring at line 7 of `dow.py`.

```
python mbox-short.txt
han = open('mbox-short.txt')
for line in han:
    line = line.rstrip()
    wds = line.split()
    print(wds)
if wds[0] != 'From' :
    raise IndexError: list index out of range
Received: (from apache@localhost)
    by nakamura.uits.iupui.edu (8.12.11.1)
        for <source@collab.sakaiproject.org>
IndexError: list index out of range
```

And the interesting thing is that there
is an empty string, I mean, empty array.

14:42

Wed 4:50 PM

A screenshot of a Mac OS X desktop environment. In the center is a terminal window titled "dow.py — /Users/py4e/Desktop/py4e". The terminal displays Python code and its execution output. The code reads lines from a file named "mbox-short.txt". The output shows various email headers and body content being processed. A specific error occurs at line 8, where a list index is out of range, indicating a blank line was encountered. The file browser on the left shows a directory structure under "py4e" containing files like "ex_02_02", "ex_02_03", etc., and sub-directories "ex_08" and ".DS_Store". The desktop background shows icons for "Macintosh HD" and "Time Machine".

```
dow.py
han = open('mbox-short.txt')
for line in han:
    line = line.rstrip()
    print('Line:', line)
    wds = line.split()
    if len(wds) > 0 and wds[0] == 'From':
        print(wds)
        ignore
        line = line[1:]
        words = line.split()
        if len(words) > 2 and words[0] == 'X-DSPAM-Result':
            print(words)
            ignore
            line = line[1:]
            words = line.split()
            if len(words) > 1 and words[0] == 'X-DSPAM-Confidence':
                print(words)
                ignore
                line = line[1:]
                words = line.split()
                if len(words) > 1 and words[0] == 'X-DSPAM-Probability':
                    print(words)
                    ignore
                    line = line[1:]
                    words = line.split()
                    if len(words) > 1 and words[0] == 'Subject':
                        print(words)
                        ignore
                        line = line[1:]
                        words = line.split()
                        if len(words) > 1 and words[0] == 'To':
                            print(words)
                            ignore
                            line = line[1:]
                            words = line.split()
                            if len(words) > 1 and words[0] == 'Cc':
                                print(words)
                                ignore
                                line = line[1:]
                                words = line.split()
                                if len(words) > 1 and words[0] == 'Bcc':
                                    print(words)
                                    ignore
                                    line = line[1:]
                                    words = line.split()
                                    if len(words) > 1 and words[0] == 'X-Content-Type-Message-Body':
                                        print(words)
                                        ignore
                                        line = line[1:]
                                        words = line.split()
                                        if len(words) > 1 and words[0] == 'text/plain; charset=UTF-8':
                                            print(words)
                                            ignore
                                            line = line[1:]
                                            words = line.split()
                                            if len(words) > 1 and words[0] == 'Content-Type':
                                                print(words)
                                                ignore
                                                line = line[1:]
                                                words = line.split()
                                                if len(words) > 1 and words[0] == 'text/plain; charset=UTF-8':
                                                    print(words)
                                                    ignore
                                                    line = line[1:]
                                                    words = line.split()
                                                    if len(words) > 1 and words[0] == 'X-DSPAM-Result: Innocent':
                                                        print(words)
                                                        ignore
                                                        line = line[1:]
                                                        words = line.split()
                                                        if len(words) > 1 and words[0] == 'X-DSPAM-Processed: Sat Jan 5 09:14:16 2008':
                                                            print(words)
                                                            ignore
                                                            line = line[1:]
                                                            words = line.split()
                                                            if len(words) > 1 and words[0] == 'X-DSPAM-Processed: Sat, Jan 5, 09:14:16, 2008':
                                                                print(words)
                                                                ignore
                                                                line = line[1:]
                                                                words = line.split()
                                                                if len(words) > 1 and words[0] == 'X-DSPAM-Confidence: 0.8475':
                                                                    print(words)
                                                                    ignore
                                                                    line = line[1:]
                                                                    words = line.split()
                                                                    if len(words) > 1 and words[0] == 'X-DSPAM-Probability: 0.0000':
                                                                        print(words)
                                                                        ignore
                                                                        line = line[1:]
                                                                        words = line.split()
                                                                        if len(words) > 1 and words[0] == 'X-DSPAM-Probability: 0.0000':
                                                                            print(words)
                                                                            ignore
                                                                            line = line[1:]
                                                                            words = line.split()
                                                                            if len(words) > 1 and words[0] == 'Subject:':
                                                                                print(words)
                                                                                ignore
                                                                                line = line[1:]
                                                                                words = line.split()
                                                                                if len(words) > 1 and words[0] == 'To:':
                                                                                    print(words)
                                                                                    ignore
                                                                                    line = line[1:]
                                                                                    words = line.split()
                                                                                    if len(words) > 1 and words[0] == 'Cc:':
                                                                                        print(words)
                                                                                        ignore
                                                                                        line = line[1:]
                                                                                        words = line.split()
                                                                                        if len(words) > 1 and words[0] == 'Bcc:':
                                                                                            print(words)
                                                                                            ignore
                                                                                            line = line[1:]
                                                                                            words = line.split()
                                                                                            if len(words) > 1 and words[0] == 'X-Content-Type-Message-Body:':
                                                                                                print(words)
                                                                                                ignore
                                                                                                line = line[1:]
                                                                                                words = line.split()
                                                                                                if len(words) > 1 and words[0] == 'text/plain; charset=UTF-8':
                                                                                                    print(words)
                                                                                                    ignore
                                                                                                    line = line[1:]
                                                                                                    words = line.split()
                                                                                                    if len(words) > 1 and words[0] == 'Content-Type':
                                                                                                        print(words)
                                                                                                        ignore
                                                                                                        line = line[1:]
                                                                                                        words = line.split()
                                                                                                        if len(words) > 1 and words[0] == 'text/plain; charset=UTF-8':
                                                                                                            print(words)
                                                                                                            ignore
                                                                                                            line = line[1:]
................................................................
ex_08/dow.py 5:23
by nakamura.uits.iupui.edu (8.12.11.1)
for <source@collab.sakaiproject.org>
Received: (from apache@localhost)
by nakamura.uits.iupui.edu (8.12.11.1)
IndexError: list index out of range
m-c02m92ubfd57:ex_08_py4e$
```

And because it's a blank line,
the split returns no words, and

14:43

Wed 4:50 PM

The screenshot shows a Mac OS X desktop environment. In the center is a terminal window titled "dow.py — /Users/py4e/Desktop/py4e". The terminal displays Python code and its execution output. The code reads lines from a file named "mbox-short.txt" and prints them. It then processes each line to extract specific headers and print them. The output shows several header fields like "Content-Type", "X-DSPAM-Result", and "X-DSPAM-Confidence". A stack trace at the end indicates an IndexError: list index out of range, which occurs when trying to access the first element of an empty list. The file browser on the left shows a directory structure under "py4e" containing various files and sub-directories labeled ex_02 through ex_08.

```
han = open('mbox-short.txt')
for line in han:
    line = line.rstrip()
    print('Line:', line)
    wds = []
    if wds[0] != 'From':
        print('Ignore')
        Line: Content-Type: text/plain; charset=UTF-8
        Words: ['Content-Type:', 'text/plain;', 'charset=UTF-8']
        Ignore
        Line: X-DSPAM-Result: Innocent
        Words: ['X-DSPAM-Result:', 'Innocent']
        print('Ignore')
        Line: X-DSPAM-Processed: Sat Jan 5 09:14:16 2008
        Words: ['X-DSPAM-Processed:', 'Sat', 'Jan', '5', '09:14:16', '2008']
        Ignore
        Line: X-DSPAM-Confidence: 0.8475
        Words: ['X-DSPAM-Confidence:', '0.8475']
        Ignore
        Line: X-DSPAM-Probability: 0.0000
        Words: ['X-DSPAM-Probability:', '0.0000']
        Ignore
        Line: I
        Words: []
Traceback (most recent call last):
  File "dow.py", line 8, in <module>
    if wds[0] != 'From' :
IndexError: list index out of range
m-c92m92uxfd57:ex_08 py4e$
```

So wds sub 0 is not valid, which is
the first word, when there are no words.

14:44

Wed 4:51 PM

The screenshot shows a Mac desktop with a window titled "dow.py — /Users/py4e/Desktop/py4e". The code in "dow.py" reads a file "mbox-short.txt" and prints lines starting with "From". The terminal below shows the output and an IndexError at line 8.

```
Atom  File  Edit  View  Selection  Find  Packages  Window  Help
dow.py — /Users/py4e/Desktop/py4e
dow.py      mbox-short.txt
1 han = open('mbox-short.txt')
2
3 for line in han:
4     line = line.rstrip()
5     print('Line:',line)
6     wds = line.split()
7     print('Words:',wds)
8     if len(wds) < 1 :
9         continue
10    if wds[0] != 'From' :
11        print('Ignore')
12        continue
13    print(wds[2])
14

ex_08/dow.py  8:7  (2, 32)
by nakamura.uits.iupui.edu (8.12.11)      if wds[0] != 'From' :
for <source@collab.sakaiproject.org> IndexError: list index out of range
Received: (from apache@localhost) m-c02m92uxfd57:ex_08 py4e$
```

if we don't have any words, meaning it's
a blank line, then we're going to skip it.

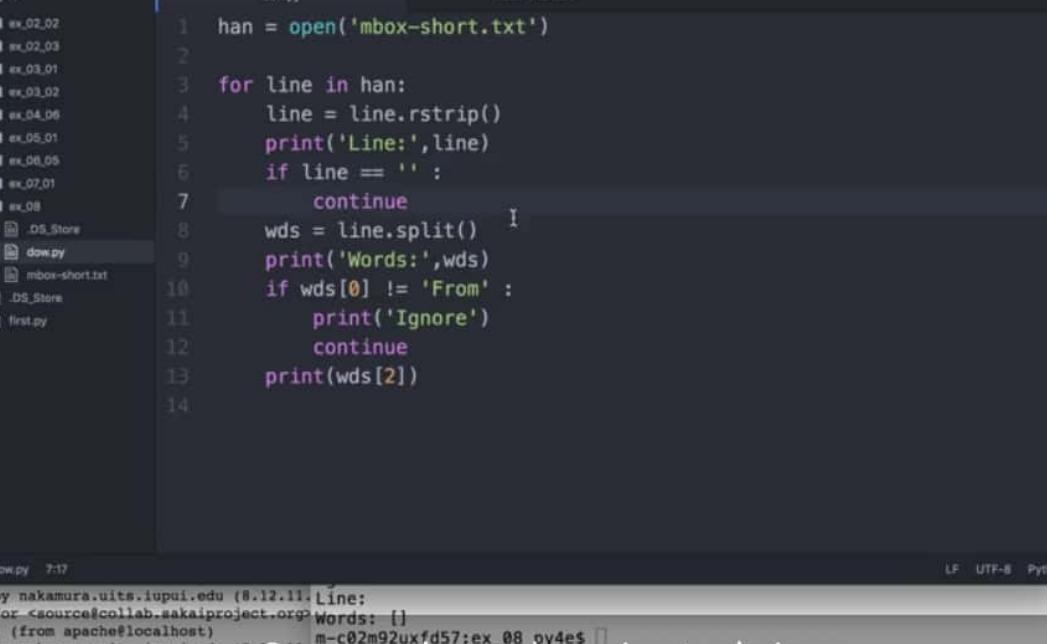
14:44

Wed 4:51 PM

```
dow.py — /Users/py4e/Desktop/py4e
dow.py mbox-short.txt
1 han = open('mbox-short.txt')
2
3 for line in han:
4     line = line.rstrip()
5     print('Line:',line)
6     wds = line.split()
7     print('Words:',wds)
8     # Guardian
9     if len(wds) < 1 :
10         continue
11     if wds[0] != 'From':
12         print('Ignore')
13         continue
14     print(wds[2])
15
Rec ex_08/dow.py* 8:16
by nakamura.uits.iupui.edu (8.12.11) Line:
for <source@collab.sakaiproject.org> Words: []
Received: (from apache@localhost) m-c02m92uxfd57:ex_08 py4e$
```

Right? Guardian pattern, because this,
this is dangerous.

14:45



```
dow.py — /Users/py4e/Desktop/py4e
dow.py      mbox-short.txt
1 han = open('mbox-short.txt')
2
3 for line in han:
4     line = line.rstrip()
5     print('Line:',line)
6     if line == '':
7         continue
8     wds = line.split()  I
9     print('Words:',wds)
10    if wds[0] != 'From' :
11        print('Ignore')
12        continue
13    print(wds[2])
14

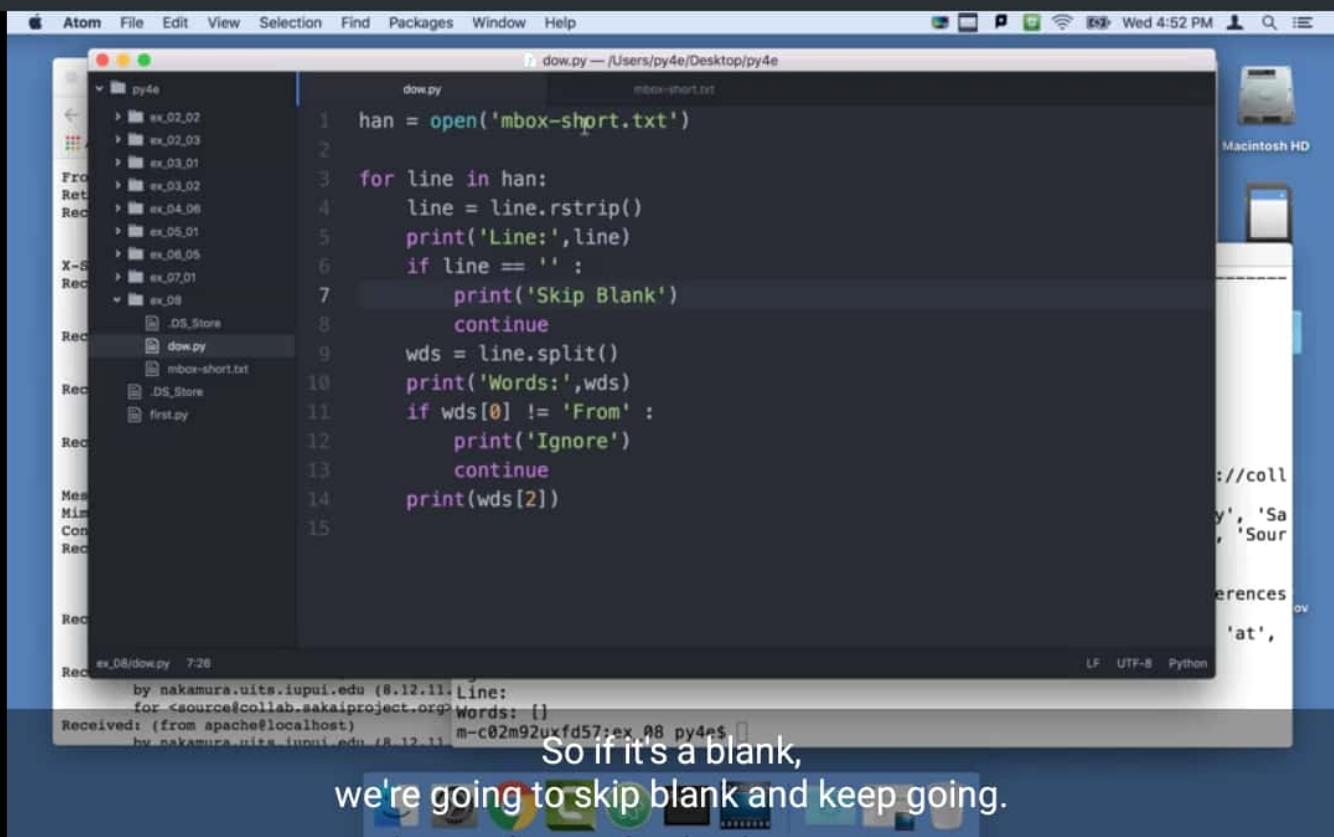
by nakamura.uits.iupui.edu (8.12.11) Line:
for <source@collab.sakaiproject.org> Words: []
Received: (from apache@localhost) m-c02m92uxfd57:ex_08 py4e$
```

So now what we're going to do is
we're going to skip blank lines.

So now what we're going to do is we're going to skip blank lines.

14:45

Wed 4:52 PM



The screenshot shows the Atom code editor with a file named `dow.py` open. The code reads an mbox file and prints specific lines based on conditions. A terminal window below the editor shows the execution of the script and its output.

```
han = open('mbox-short.txt')
for line in han:
    line = line.rstrip()
    print('Line:',line)
    if line == '':
        print('Skip Blank')
        continue
    wds = line.split()
    print('Words:',wds)
    if wds[0] != 'From' :
        print('Ignore')
        continue
    print(wds[2])
```

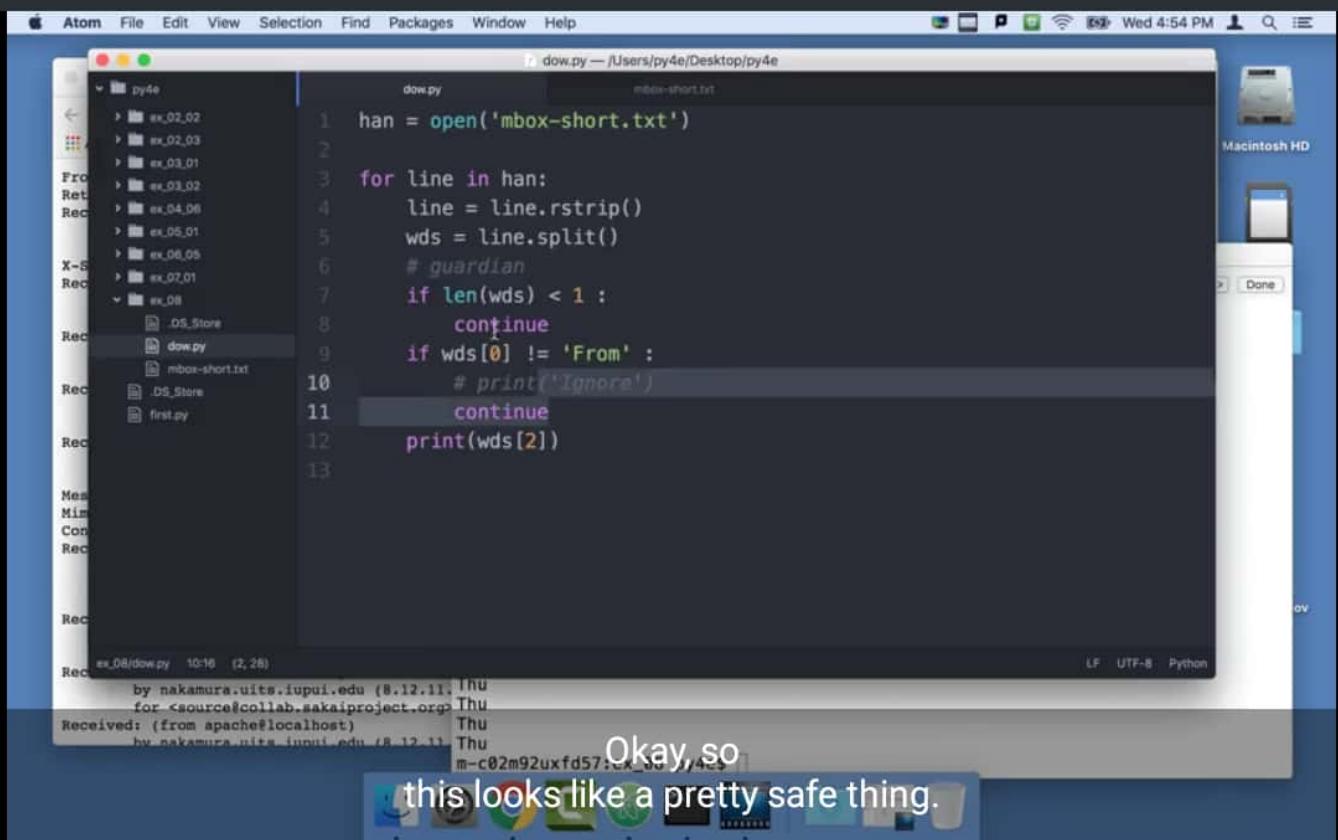
Terminal output:

```
ex_08/dow.py 7:26
by nakamura.uits.iupui.edu (8.12.11) Line:
for <source@collab.sakaiproject.org> Words: []
Received: (from apache@localhost) m-c02m92uxfd57:ex_08_py4e$
```

So if it's a blank,
we're going to skip blank and keep going.

14:48

Wed 4:54 PM



The screenshot shows the Atom code editor on a Mac OS X desktop. The window title is "dow.py — /Users/py4e/Desktop/py4e". The code in the editor is:

```
han = open('mbox-short.txt')
for line in han:
    line = line.rstrip()
    wds = line.split()
    # guardian
    if len(wds) < 1:
        continue
    if wds[0] != 'From':
        # print('Ignore')
        continue
    print(wds[2])
```

The status bar at the bottom shows the file path "ex_08/dow.py" and line numbers "10:16 (2,28)". The bottom right corner of the screen has a small "Done" button.

Okay, so

this looks like a pretty safe thing.

14:48

Wed 4:54 PM

The screenshot shows a Mac desktop with an Atom code editor window open. The file `dow.py` is being edited, reading from `mbox-short.txt`. The code uses a guardian pattern to skip lines that don't start with 'From'. A status bar at the bottom shows the file is 10:17 (2, 38) and includes tabs for LF, UTF-8, and Python. Below the editor, the terminal window displays the processed email headers.

```
han = open('mbox-short.txt')
for line in han:
    line = line.rstrip()
    wds = line.split()
    # guardian
    if len(wds) < 1:
        continue
    if wds[0] != 'From':
        continue
    print(wds[2])
```

Rec ex_08/dow.py* 10:17 (2, 38) LF UTF-8 Python

```
by nakamura.uits.iupui.edu (8.12.11) Thu
for <source@collab.sakaiproject.org> Thu
Received: (from apache@localhost) Thu
by nakamura.uits.iupui.edu (8.12.11) Thu
2023-04-07 14:54:57 UTC Python/3.10.4
```

changed from when we started except
we've added this little guardian.

14:49

75

The screenshot shows a Mac OS X desktop environment. A terminal window is open, displaying a Python script named `dow.py`. The script reads from a file named `mbox-short.txt` and prints lines starting with 'From'. The terminal window also shows the output of the script, which includes several lines of email headers.

```
dow.py — /Users/py4e/Desktop/py4e
dow.py      mbox-short.txt
1 han = open('mbox-short.txt')
2
3 for line in han:
4     line = line.rstrip()
5     wds = line.split()
6     # guardian
7     if len(wds) < 1 :
8         continue
9     if wds[0] != 'From' :
10        continue
11    print(wds[2])
12

ex_08/dow.py* 10:17 (2, 38)
by nakamura.uits.iupui.edu (8.12.11. Thu
for <source@collab.sakaiproject.org> Thu
Received: (from apache@localhost)
hu_nakamura.uits_iupui_edu (8.12.11. Thu
```

Now, the interesting thing is if it comes through here, and prints wds sub 2,

14:49

75

The screenshot shows a Mac OS X desktop environment. In the foreground, a terminal window is open with the command `ls` and its output:

```
ex_08/ dow.py* 10:17 (2, 38)
by nakamura.uits.iupui.edu (8.12.11. Thu
for <source@collab.sakaiproject.org> Thu
Received: (from apache@localhost [127.0.0.1])
        by nakamura.uits.iupui.edu (8.12.11. Thu
        for <source@collab.sakaiproject.org> Thu
```

Below the terminal, a code editor window titled "dow.py — /Users/py4e/Desktop/py4e" displays the following Python script:

```
1 han = open('mbox-short.txt')
2
3 for line in han:
4     line = line.rstrip()
5     wds = line.split()
6     # guardian
7     if len(wds) < 1 :
8         continue
9     if wds[0] != 'From' :
10        continue
11    print(wds[2])
```

14:50

HD 1x Macintosh HD

Worked Exercise: Lists

```
dow.py — /Users/py4e/Desktop/py4e
dow.py      mbox-short.txt

1 han = open('mbox-short.txt')
2
3     for line in han:
4         line = line.rstrip()
5         wds = line.split()
6         # guardian
7         if len(wds) < 1:
8             continue
9         if wds[0] != 'From':
10            continue
11         print(wds[2])
12
```



there's only one word on,
this is going to blow up.

```
ex_08/dow.py* 0:23 (1,4)
by nakamura.uits.iupui.edu (8.12.11.Thu
for <source@collab.sakaiproject.org> Thu
Received: (from apache@localhost)
    by nakamura.uits.iupui.edu (8.12.11.Thu
m-c02m92uxfd57:ex_08 py4e$
```

08:42

11:33

10

14:50

HD 1x Macintosh HD

Worked Exercise: Lists

```
dow.py — /Users/py4e/Desktop/py4e
dow.py      mbox-short.txt

1 han = open('mbox-short.txt')
2
3 for line in han:
4     line = line.rstrip()
5     wds = line.split()
6     # guardian a bit stronger
7     if len(wds) < 3 :
8         continue
9     if wds[0] != 'From':
10        continue
11    print(wds[2])
12
```



we're going to skip this line if
it doesn't have three words in it.

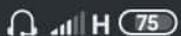
```
Rec ex_08/dow.py* 7:30
by nakamura.uits.iupui.edu (8.12.11. Thu
for <source@collab.sakaiproject.org> Thu
Received: (from apache@localhost) Thu
by nakamura.uits.iupui.edu (8.12.11. Thu
```

08:52

LF UTF-8 Python

11:33

14:51



Worked Exercise: Lists



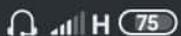
```
1  han = open('mbox-short.txt')
2
3  for line in han:
4      line = line.rstrip()
5      wds = line.split()
6      # guardian in a compound statement
7      if len(wds) < 3 or wds[0] != 'From':
8          continue
9      print(wds[2])
```

is true if either that's true or
this is true.

09:55 11:33

10

14:52



Worked Exercise: Lists

The screenshot shows a Mac desktop environment. In the center is a terminal window titled "dow.py — /Users/py4e/Desktop/py4e". The terminal contains the following Python code:

```
han = open('mbox-short.txt')
for line in han:
    line = line.rstrip()
    wds = line.split()
    # guardian in a compound statement
    if len(wds) < 3 or wds[0] != 'From':
        continue
    print(wds[2])
```

Below the terminal is a large white play button icon. At the bottom of the screen, there is a status bar with the following information:

10:07 11:33

So if we flip this order, it would fail.
If we do did in this order, it will work.

14:52

Macintosh HD

The screenshot shows a Mac OS X desktop environment. In the center is a terminal window titled "dow.py — /Users/py4e/Desktop/py4e". The terminal contains the following Python code:

```
1 han = open('mbox-short.txt')
2
3 for line in han:
4     line = line.rstrip()
5     wds = line.split()
6     # guardian in a compound statement
7     if len(wds) < 3 or wds[0] != 'From':
8         continue
9     print(wds[2])
```

Below the terminal, a message box displays the output of the script:

```
ex_08/dow.py* 7:40 (1,18)
by nakamura.uits.iupui.edu (6.12.11,
for <source@collab.sakaiproject.org>
Received: (from apache@localhost)
hu_nakamura.uits_iupui.edu (P)
```

The message box also contains the text: "This is called short circuit evaluation where it knows that as long as this".

14:52

Macintosh HD

The screenshot shows a Mac OS X desktop environment. A terminal window is open, displaying Python code and its execution output. The terminal window title is "dow.py — /Users/py4e/Desktop/py4e". The code in the terminal is:

```
dow.py
1 han = open('mbox-short.txt')
2
3 for line in han:
4     line = line.rstrip()
5     wds = line.split()
6     # guardian in a compound statement
7     if len(wds) < 3 or wds[0] != 'From':
8         continue
9     print(wds[2])
10
```

The output of the command "ex_08/dow.py" is shown in the terminal window:

```
ex_08/dow.py* 7:40 (1,18)
by nakamura.uits.iupui.edu (8.12.11
for <source@collab.sakaiproject.org>
Received: (from apache@localhost)
    by nakamura.uits.iupui.edu (8.12.11)
```

A tooltip message is overlaid on the screen, reading "part's true, it doesn't evaluate this second part." This indicates that the code is correctly identifying lines starting with "From" and skipping them.



What is a Collection?



- A collection is nice because we can put more than one value in it and carry them all around in one convenient package
- We have a bunch of values in a single “variable”
- We do this by having more than one place “in” the variable
- We have ways of finding the different places in the variable
 - more than one thing in a single variable,
and a way of indexing and looking up and



A Story of Two Collections..

- List

- A linear collection of values that stay in order



- Dictionary

- A “bag” of values, each with its own label
Everything is very precise,
zero, one, two, three, four.





A Story of Two Collections..

- List

- A linear collection of values that stay in order



- Dictionary

- A “bag” of values, each with its own label

You put things in, and you give them a label, and you get things back out.



17:26

... 🔍 4G H 79



And so they don't always stay in order,
the order shifts around.

M

Dictionaries



- Dictionaries are Python's most powerful data collection
- Dictionaries allow us to do fast database-like operations in Python
- Dictionaries have different names in different languages
 - Associative Arrays - Perl / PHP
 - Properties or Map or HashMap - Java
 - Property Bag - C# / .Net

So dictionaries are Python's
most powerful collection.

Dictionaries

- Lists **index** their entries based on the position in the list
- **Dictionaries** are like bags - no order
- So we **index** the things we put in the **dictionary** with a "lookup tag"

And this is a constructor that basically says, make me an empty dictionary object.

```
>>> purse = dict()
>>> purse['money'] = 12
>>> purse['candy'] = 3
>>> purse['tissues'] = 75
>>> print(purse)
{'money': 12, 'tissues': 75, 'candy': 3}
>>> print(purse['candy'])
3
>>> purse['candy'] = purse['candy'] + 2
>>> print(purse)
{'money': 12, 'tissues': 75, 'candy': 5}
```

Dictionaries



- Lists **index** their entries based on the position in the list
- **Dictionaries** are like bags - no order
- So we **index** the things we put in the **dictionary** with a "lookup tag"

```
>>> purse = dict()  
>>> purse['money'] = 12  
>>> purse['candy'] = 3  
>>> purse['tissues'] = 75  
>>> print(purse)  
{'money': 12, 'tissues': 75, 'candy': 3}  
>>> print(purse['candy'])  
3  
>>> purse['candy'] = purse['candy'] + 2  
>>> print(purse)  
{'money': 12, 'tissues': 75, 'candy': 5}
```

Curly braces are for dictionaries.



Dictionaries

- Lists **index** their entries based on the position in the list
- **Dictionaries** are like bags - no order
- So we **index** the things we put in the **dictionary** with a "lookup tag"

And so 12 is going in, 12 is the value and we're putting it into purse, but

```
>>> purse = dict() ✓
>>> purse['money'] = 12
>>> purse['candy'] = 3
>>> purse['tissues'] = 75
>>> print(purse)
{'money': 12, 'tissues': 75, 'candy': 3}
>>> print(purse['candy'])
3
>>> purse['candy'] = purse['candy'] + 2
>>> print(purse)
{'money': 12, 'tissues': 75, 'candy': 5}
```

9.1 - Dictionaries

Dictionaries

HD

...

1x

...

- Lists **index** their entries based on the position in the list
- **Dictionaries** are like bags - no order
- So we **index** the things we put in the **dictionary** with a "lookup tag"

03:40

```
>>> purse = dict()
>>> purse['money'] = 12
>>> purse['candy'] = 3
>>> purse['tissues'] = 75
>>> print(purse)
{'money': 12, 'tissues': 75, 'candy': 3}
>>> print(purse['candy'])
3
>>> purse['candy'] = purse['candy'] + 2
>>> print(purse)
{'money': 12, 'tissues': 75, 'candy': 5}
```

10

09:32

...



Dictionaries

- Lists **index** their entries based on the position in the list
- **Dictionaries** are like bags - no order
- So we **index** the things we put in the **dictionary** with a "lookup tag"

```
>>> purse = dict()
>>> purse['money'] = 12 ✓
>>> purse['candy'] = 3
>>> purse['tissues'] = 75
>>> print(purse)
{'money': 12, 'tissues': 75, 'candy': 3}
>>> print(purse['candy'])
3
>>> purse['candy'] = purse['candy'] + 2
>>> print(purse)
{'money': 12, 'tissues': 75, 'candy': 5}
```

we're storing it under money.

Dictionaries

- Lists **index** their entries based on the position in the list
- **Dictionaries** are like bags - no order
- So we **index** the things we put in the **dictionary** with a "lookup tag"

Money is the label, the index, the tag, whatever you want to call it.

```
>>> purse = dict()  
>>> purse['money'] = 12  
>>> purse['candy'] = 3  
>>> purse['tissues'] = 75  
>>> print(purse)  
{'money': 12, 'tissues': 75, 'candy': 3}  
>>> print(purse['candy'])  
3  
>>> purse['candy'] = purse['candy'] + 2  
>>> print(purse)  
{'money': 12, 'tissues': 75, 'candy': 5}
```

Comparing Lists and Dictionaries

Dictionaries are like lists except that they use keys instead of numbers to look up values

```
>>> lst = list()  
>>> lst.append(21)  
>>> lst.append(183)  
>>> print(lst)  
[21, 183]  
>>> lst[0] = 23  
>>> print(lst)  
[23, 183]
```



```
>>> ddd = dict()  
>>> ddd['age'] = 21  
>>> ddd['course'] = 182  
>>> print(ddd)  
{'course': 182, 'age': 21}  
>>> ddd['age'] = 23  
>>> print(ddd)  
{'course': 182, 'age': 23}
```

Here we put a into this dictionary
under the key age 21 and



Comparing Lists and Dictionaries

Dictionaries are like lists except that they use keys instead of numbers to look up values

```
>>> lst = list()  
>>> lst.append(21)  
>>> lst.append(183)  
>>> print(lst)  
[21, 183]  
>>> lst[0] = 23  
>>> print(lst)  
[23, 183]
```

```
>>> ddd = dict()  
>>> ddd['age'] = 21  
>>> ddd['course'] = 182  
>>> print(ddd)  
{'course': 182, 'age': 21}  
>>> ddd['age'] = 23  
>>> print(ddd)  
{'course': 182, 'age': 23}
```

Just don't expect the order of dictionaries to be a predictable thing.



Comparing Lists and Dictionaries

Dictionaries are like lists except that they use keys instead of numbers to look up values

```
>>> lst = list()  
>>> lst.append(21)  
>>> lst.append(183)  
>>> print(lst)  
[21, 183]  
>>> lst[0] = 23  
>>> print(lst)  
[23, 183]
```

```
>>> ddd = dict()  
>>> ddd['age'] = 21  
>>> ddd['course'] = 182  
>>> print(ddd)  
{'course': 182, 'age': 21}  
>>> ddd['age'] = 23  
>>> print(ddd)  
{'course': 182, 'age': 23}
```

We put a 23 into position 0 and so when we see that, we see a 23 coming out.

Comparing Lists and Dictionaries

Dictionaries are like lists except that they use keys instead of numbers to look up values

```
>>> lst = list()
>>> lst.append(21)
>>> lst.append(183)
>>> print(lst)
[21, 183]
>>> lst[0] = 23
>>> print(lst)
[23, 183]
```

```
>>> ddd = dict()
>>> ddd['age'] = 21
>>> ddd['course'] = 182
>>> print(ddd)
{'course': 182, 'age': 21}
>>> ddd['age'] = 23
>>> print(ddd)
{'course': 182, 'age': 23}
```

go find that thing marked with
the tag age and replace that with 23.



Most Common Name?

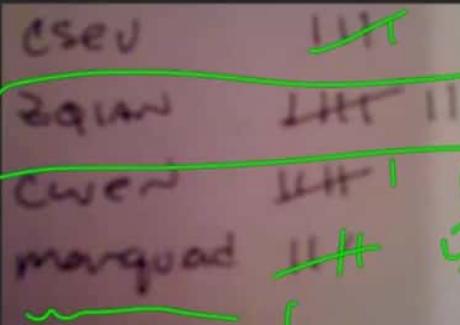
marquard

zhen

csev

zhen

cwen



cwen

zhen

csev

marquard

zhen

grow this one, and then you're all done
you got sort of the tallest histogram.

Many Counters with a Dictionary

One common use of dictionaries is counting how often we “see” something

```
>>> ccc = dict()
>>> ccc['csev'] = 1
>>> ccc['cwen'] = 1
>>> print(ccc)
{'csev': 1, 'cwen': 1}
>>> ccc['cwen'] = ccc['cwen'] + 1
>>> print(ccc)
{'csev': 1, 'cwen': 2}
```

So cwen has 2, chuck has 1, and away we go.

Key	Value
csev	11
brian	LHT 11
cwen	LHT
marquard	11'

Dictionary Tracebacks

- It is an **error** to reference a key which is not in the dictionary
- We can use the **in** operator to see if a key is in the dictionary

```
>>> ccc = dict()
>>> print(ccc['csev'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'csev'
>>> 'csev' in ccc
False
```

So Python is unhappy when you go and look for a key that doesn't exist.

Dictionary Tracebacks

- It is an **error** to reference a key which is not in the dictionary
- We can use the **in** operator to see if a key is in the dictionary

```
>>> ccc = dict()
>>> print(ccc['csev'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'csev'
>>> 'csev' in ccc
False
```

And it asks not the value, it's saying
is this key csev in this dictionary ccc?

When We See a New Name

When we encounter a new name, we need to add a new entry in the **dictionary** and if this is the second or later time we have seen the **name**, we simply add one to the count in the **dictionary** under that **name**

```
counts = dict()
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for name in names :
    if name not in counts:
        counts[name] = 1
    else :
        counts[name] = counts[name] + 1
print(counts)
```

{'csev': 2, 'zqian': 1, 'cwen': 2}

csev	111
zqian	1111
cwen	111
mangad	111

if the name we're looking at is not in our dictionary, then set counts sub name = 1.

When We See a New Name

When we encounter a new name, we need to add a new entry in the **dictionary** and if this is the second or later time we have seen the **name**, we simply add one to the count in the **dictionary** under that **name**

```
counts = dict()
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for name in names :
    if name not in counts:
        counts[name] = 1
    else:
        counts[name] = counts[name] + 1
print(counts)
```

{'csev': 2, 'zqian': 1, 'cwen': 2}

csev	111
zqian	1111
cwen	111
newgoud	111

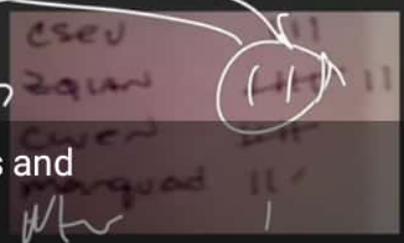
that's getting things started. That's adding a new entry and setting it to 1.

When We See a New Name

When we encounter a new name, we need to add a new entry in the **dictionary** and if this the second or later time we have seen the **name**, we simply add one to the count in the **dictionary** under that **name**

```
counts = dict()
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for name in names :
    if name not in counts:
        counts[name] = 1
    else:
        counts[name] = counts[name] + 1
print(counts)
```

So as this runs it both makes new ones and then updates the existing ones.



The get Method for Dictionaries

The pattern of checking to see if a **key** is already in a dictionary and assuming a default value if the **key** is not there is so common that there is a **method** called **get()** that does this for us

Default value if key does not exist
(and no Traceback).

Meaning this is the value I get back if the key doesn't exist.

```
if name in counts:  
    x = counts[name]  
else :  
    x = 0  
  
x = counts.get(name, 0)
```

By DEFAULT

{'csev': 2, 'zqian': 1, 'cwen': 2}

Simplified Counting with `get()`

We can use `get()` and provide a **default value of zero** when the **key** is not yet in the dictionary - and then just add one

```
counts = dict()
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for name in names :
    counts[name] = counts.get(name, 0) + 1
print(counts)
```

If there's nothing in there we get
0 plus 1 and then we get a 1 and

Default ('csev': 2, 'zqian': 1, 'cwen': 2}

Simplified Counting with `get()`

```
counts = dict()
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for name in names :
    counts[name] = counts.get(name, 0) + 1
print(counts)
```



<http://www.youtube.com/watch?v=EHJ9uYx5L58>

we're just going to use this idiom
over and over and over again.

Counting Pattern

```
counts = dict()
print('Enter a line of text:')
line = input('')
words = line.split()
print('Words:', words)

print('Counting...')
for word in words:
    counts[word] = counts.get(word, 0) + 1
print('Counts', counts)
```

Line of text goes into this variable
line, then we split it.

The general pattern to count the words in a line of text is to **split** the line into words, then loop through the words and use a **dictionary** to track the count of each word independently.

Counting Pattern

```
counts = dict()
print('Enter a line of text:')
line = input('')
words = line.split()
print('Words:', words)

print('Counting...')
for word in words:
    counts[word] = counts.get(word, 0) + 1
print('Counts', counts)
```

The general pattern to count the words in a line of text is to **split** the line into words, then loop through the words and use a **dictionary** to track the count of each word independently.

So this is both going to do the new and the existing.

```
python wordcount.py
```

Enter a line of text:

the clown ran after the car and the car ran into the tent
and the tent fell down on the clown and the car



Words: ['the', 'clown', 'ran', 'after', 'the', 'car',
'and', 'the', 'car', 'ran', 'into', 'the', 'tent', 'and',
'the', 'tent', 'fell', 'down', 'on', 'the', 'clown',
'and', 'the', 'car']

Counting...

Counts {'and': 3, 'on': 1, 'ran': 2, 'car': 3, 'into': 1,
'after': 1, 'clown': 2, 'down': 1, 'fell': 1, 'the': 7,
'tent': 2}



And then at the very end, we're going to print all these counts out.

23:05

... 4G H 48

EVERYBODY

```
counts = dict()
line = input('Enter a line of text:')
words = line.split()
print('Words:', words)
print('Counting...')

for word in words:
    counts[word] = counts.get(word, 0) + 1
print('Counts', counts)
```



```
python wordcount.py
Enter a line of text:
the clown ran after the car and the car ran
into the tent and the tent fell down on the
clown and the car
```

```
Words: ['the', 'clown', 'ran', 'after', 'the', 'car',
'and', 'the', 'car', 'ran', 'into', 'the', 'tent', 'and',
'the', 'tent', 'fell', 'down', 'on', 'the', 'clown',
'and', 'the', 'car']
Counting...
```

```
Counts {'and': 3, 'on': 1, 'ran': 2, 'car': 3,
'into': 1, 'after': 1, 'clown': 2, 'down': 1, 'fell':
1, 'the': 7, 'tent': 2}
```

And then we have this word that goes iteration across this, the words in a line.

Definite Loops and Dictionaries

Even though **dictionaries** are not stored in order, we can write a **for** loop that goes through all the **entries** in a **dictionary** - actually it goes through all of the **keys** in the **dictionary** and **looks up** the values

```
>>> counts = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> for key in counts:
...     print(key, counts[key])
...
jan 100
chuck 1
fred 42
>>>
```

it goes through the values, but in a dictionary, it goes through the keys.

Retrieving lists of Keys and Values

You can get a list of **keys**, **values**, or **items** (both) from a dictionary

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print(list(jjj)) 
['jan', 'chuck', 'fred']
>>> print(jjj.keys())
['jan', 'chuck', 'fred']
>>> print(jjj.values())
[100, 1, 42]
>>> print(jjj.items())
[('jan', 100), ('chuck', 1), ('fred', 42)]
>>>
```

What is a “tuple”? - coming soon...

telling Python to convert this dictionary variable to a list.

Retrieving lists of Keys and Values

You can get a list of **keys**, **values**, or **items** (both) from a dictionary

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print(list(jjj)) _____
['jan', 'chuck', 'fred']
>>> print(jjj.keys())
['jan', 'chuck', 'fred']
>>> print(jjj.values())
[100, 1, 42]
>>> print(jjj.items())
[('jan', 100), ('chuck', 1), ('fred', 42)]
>>>
```

What is a “tuple”? - coming soon...

So lists have less information and that ends up giving the keys, right?

Retrieving lists of Keys and Values

You can get a list of **keys**, **values**, or **items** (both) from a dictionary

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print(list(jjj))
['jan', 'chuck', 'fred'] ← (LIST OF ITEMS)
>>> print(jjj.keys())
['jan', 'chuck', 'fred'] ← (KEYS)
>>> print(jjj.values())
[100, 1, 42] ←
>>> print(jjj.items())
[('jan', 100), ('chuck', 1), ('fred', 42)]
>>>
```

What is a "tuple"? - coming soon...

a moment later, they come out in corresponding order.

Retrieving lists of Keys and Values

You can get a list of **keys**, **values**, or **items** (both) from a dictionary

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print(list(jjj))
['jan', 'chuck', 'fred']
>>> print(jjj.keys())
['jan', 'chuck', 'fred']
>>> print(jjj.values())
[100, 1, 42]
>>> print(jjj.items())
[('jan', 100), ('chuck', 1), ('fred', 42)]
```

What is a "tuple"? - coming soon...

these little guys with parentheses
are called tuples,

Bonus: Two Iteration Variables!

- We loop through the **key-value** pairs in a dictionary using *two* iteration variables
- Each iteration, the first variable is the **key** and the second variable is the corresponding **value** for the key

```
jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
for aaa,bbb in jjj.items() :
    print(aaa, bbb)
```

aaa jan	bbb 100
chuck	1
fred	42

No other language that I know of has the ability to do more than one iteration variable.

Bonus: Two Iteration Variables!

- We loop through the **key-value** pairs in a dictionary using *two* iteration variables
- Each iteration, the first variable is the **key** and the second variable is the corresponding **value** for the key

```
jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
for aaa,bbb in jjj.items() :
    print(aaa, bbb)
```

aaa	bbb
[jan]	100
[chuck]	1
[fred]	42

And aaa goes through the keys and bbb goes through the, simultaneously, the values.

Bonus: Two Iteration Variables!

- We loop through the **key-value** pairs in a dictionary using *two* iteration variables
- Each iteration, the first variable is the **key** and the second variable is the corresponding **value** for the key

```
jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
for aaa,bbb in jjj.items() :
    print(aaa, bbb)
```

jan 100
chuck 1
fred 42

Key VALUE
aaa bbb

→ [jan] 100
→ [chuck] 1
→ [fred] 42

So this is a very succinct and convenient
way to go through

23:29 ... 4G H 46

Dictionaries - Part 3

```
name = input('Enter file: ')
handle = open(name)

counts = dict()
for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1

bigcount = None
bigword = None
for word, count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

print(bigword, bigcount)
```

```
python words.py
Enter file: words.txt
to 16
```

```
python words.py
Enter file: clown.txt
the 7
```

So what we're going to do is we got Using two nested loops this histogram in counts, you know.

9.3 - Dictionaries and Files

```

handle = open(name)

counts = dict()
for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word,0) + 1

bigcount = None
bigword = None
for word, count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

print(bigword, bigcount)

```



python words.py
Enter file: words.txt
to 16

python words.py
Enter file: clown.txt
the 7

11:36

Using two nested loops

13:37

23:30

... 4G H 46

EVERYBODY

DICTIONARIES - PART 3

```
name = input('Enter file: ')
handle = open(name)

counts = dict()
for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1

bigcount = None
bigword = None
for word, count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

print(bigword, bigcount)
```



python words.py
Enter file: words.txt
to 16

python words.py
Enter file: clown.txt
the 7

Using two nested loops
which means we're on the first word,

23:30

... 4G H 46

EVERYBODY

DICTIONARIES - PART 3

```
name = input('Enter file: ')
handle = open(name)

counts = dict()
for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1

bigcount = None
bigword = None
for word, count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

print(bigword, bigcount)
```



python words.py
Enter file: words.txt
to 16

python words.py
Enter file: clown.txt
the 7

one or if count is greater than
the biggest count, then remember it.

7:52

... ⚡ H 100%

The screenshot shows a Mac desktop environment. In the center is a terminal window titled "ex_09.py" with the path "/Users/py4e/Desktop/py4e". The terminal displays the following Python code:

```
1 fname = input('Enter File: ')
2 if len(fname) < 1 : fname = 'clown.txt'
3 hand = open(fname)
4
5 for lin in hand:
6     lin = lin.rstrip()
7     print(lin)
```

Below the terminal, a file browser window is open, showing a directory structure under "the". The "ex_09" folder contains files: ".DS_Store", "dow.py", "mbox-short.txt", "ex_09.py", "clown.txt", "intro.txt", ".DS_Store", and "first.py".

The status bar at the bottom of the screen shows "ex_09/ex_09.py 7:14". A tooltip message is displayed in the foreground: "I want to hit Enter now, and it's going to assume hopefully clown.txt".

7:56

... 🔍 ⌂ H 99

The screenshot shows a Mac desktop environment. In the center is a terminal window titled 'ex_09.py' with the command 'python3 ex_09.py'. The terminal displays a script named 'ex_09.py' which reads a file name from input or defaults to 'clown.txt', then opens it and prints its contents line by line. Below the terminal, a message box contains the text: 'That's the words that have been split from the line.' To the left of the terminal is a file browser showing a directory structure under 'the' folder, including subfolders 'py4e' and 'ex_09' containing various Python scripts and text files like 'clown.txt' and 'intro.txt'. The desktop background shows icons for 'Macintosh HD' and a small image of a person.

```
the
  py4e
    ex_02_02
    ex_02_03
    ex_03_01
    ex_03_02
    ex_04_06
    ex_05_01
    ex_06_05
    ex_07_01
  ex_08
    .DS_Store
    dow.py
    mbox-short.txt
  ex_09
    clown.txt
    ex_09.py
    intro.txt
    .DS_Store
    first.py

ex_09/ex_09.py 11:17
```

```
1 fname = input('Enter File: ')
2 if len(fname) < 1 : fname = 'clown.txt'
3 hand = open(fname)
4
5 for lin in hand:
6     lin = m-c02m92uxfd57:ex_09 py4e$ python3 ex_09.py
7     # print(lin)
8     wds = lin.split()
9     print(wds)
10    for w in wds:
11        print(w)
12
```

```
m-c02m92uxfd57:ex_09 py4e$ python3 ex_09.py
Enter File:
['the', 'clown', 'ran', 'after', 'the', 'car', 'and', 'the', 'car', 'ran', 'into',
 'the', 'tent', 'and', 'the', 'tent', 'fell', 'down', 'on', 'the', 'clown', 'a
nd', 'the', 'car']

the
clown
ran
after
the
car
and
the
car
ran
into
the
tent
and
the
tent
fell
down
```

That's the words that have
been split from the line.

7:56

... 🔍 ⌂ H 99

The screenshot shows a Mac desktop environment. In the top right corner, there are system status icons. The desktop background is blue with icons for 'Macintosh HD' and 'the'. A terminal window titled 'ex_09.py — /Users/py4e/Desktop/py4e' is open, displaying the following Python code:

```
1 fname = input('Enter File: ')
2 if len(fname) < 1 : fname = 'clown.txt'
3 hand = open(fname)
4
5 for lin in hand:
6     lin = lin.rstrip()
7     # print(lin)
8     wds = lin.split()
9     print(wds)
10    for w in wds:
11        print(w)
```

The code reads a file named 'fname' (or 'clown.txt' if none is provided), strips whitespace from each line, splits each line into words using the space character as a delimiter, and then prints each word on a new line.

Below the terminal, a text editor window titled 'ex_09/ex_09.py' is open, showing the same code. The status bar at the bottom of the text editor indicates 'ex_09/ex_09.py 11:17' and file encoding options 'LF' and 'UTF-8'. The text editor has tabs for 'clown.txt' and 'intro.txt'. A message box is overlaid on the text editor, containing the text: "that w is going to successively take on literally all the words of this file."

7:58 ... 🔍 ⌂ H 99

The screenshot shows a Mac desktop environment. In the center is a terminal window titled "ex_09.py — /Users/py4e/Desktop/py4e". The code inside the terminal is:

```
1 fname = input('Enter File: ')
2 if len(fname) < 1 : fname = 'clown.txt'
3 hand = open(fname)
4
5 di = dict()
6 for lin in hand:
7     lin = lin.rstrip()
8     # print(lin)
9     wds = lin.split()
# print(wds)
11    for w in wds:
12        if w in di :
13            di[w] |
14        print(w)
15
```

Below the terminal, a message box displays the text: "which is our key in the key-value store of the dictionary,". The desktop background shows icons for "Macintosh HD" and "Time Machine". The bottom of the screen shows the Dock with various application icons.

7:59 ... HD 1x Macintosh HD

Worked Exercise: Dictionary

```
1 ex_09.py — /Users/py4e/Desktop/py4e
2 if len(fname) < 1 : fname = 'clown.txt'
3 hand = open(fname)
4
5 di = dict()
6 for lin in hand:
7     lin = lin.rstrip()
8     # print(lin)
9     wds = lin.split()
# print(wds)
11    for w in wds:
12        if w in di:
13            di[w] = di[w] + 1
14        else:
15            di[w] = 1
16    print('***NEW***')
17
18
```

then print w and the current value of
the counter for w as it's going through.

07:34

absurd
it.
m-c02m92uxfd57:ex_09_py4e\$

LF UTF-8 Python

24:23

8:01 ... ⚡ ⌂ H 99

The screenshot shows a Mac desktop environment. In the center is a terminal window titled "ex_09.py — /Users/py4e/Desktop/py4e". The terminal displays the following Python script and its execution:

```
ex_09.py
1 if len(fname) < 1 : fname = 'clown.txt'
2 hand = open(fname)
3
4 di = dict()
5 for lin in hand:
6     lin = lin.rstrip()
7     # print(lin)
8     wds = lin.split()
9     # print(wds)
10    for w in wds:
11        print(w)
12        if w in di :
13            di[w] = di[w] + 1
14            print('**Existing**')
15        else:
16            di[w] = 1
17            print('**NEW**')
18
19 print(di[w])
20
```

Below the terminal, the system menu bar shows the date and time (8:01), battery status (99%), and signal strength. To the right of the terminal is a sidebar with icons for "Macintosh HD" and "Time Machine". At the bottom of the screen, there are several application icons, including Finder, Mail, Safari, and others.

Terminal Output:

```

z
ran
**Existing**
```

Text at the bottom of the screen:

then we added di sub the = di sub the + 1.

8:01 ... HD 1x Macintosh HD

Worked Exercise: Dictionary

```
the <-- py4e
> ex_02_03
> ex_03_01
> ex_03_02
> ex_04_06
> ex_05_01
> ex_06_05
> ex_07_01
<-- ex_08
  .DS_Store
  dow.py
  mbox-short.txt
<-- ex_09
  .DS_Store
  clown.txt
  ex_09.py
  intro.txt
  first.py
ex_09/ex_09.py 13:19 (1, 8)
```

```
ex_09.py --> /Users/py4e/Desktop/py4e
1
1
# pr
**NEW**
1
after
**NEW**
1
the
**Existing**
2
car
**NEW**
1
and
**NEW**
1
the
**Existing**
3
car
**Existing**
09:05
ran
**Existing**
```

And so that's why we added 1 to it.

10

24:23

8:02 ... ⚡ ⌂ H 98

The screenshot shows a Mac desktop environment. In the center is a terminal window titled "ex_09.py" with the command "/Users/py4e/Desktop/py4e/ex_09/ex_09.py". The terminal output displays a Python script that reads lines from files and counts words. The script uses a dictionary to track word counts, distinguishing between new words and existing ones. The terminal window also shows the path "/Users/py4e/Desktop/py4e/ex_09/ex_09.py" and the line number "21:10". Below the terminal is a file browser window titled "the" showing a directory structure under "py4e". The "ex_09" folder contains files like "clown.txt", "intro.txt", and "first.py". The "ex_09.py" file itself is visible in the browser. The desktop background shows icons for "Macintosh HD" and "Time Machine". The status bar at the bottom includes icons for battery, signal strength, and a clock.

```
4
5  di = dict()
6  for lin in hand:
7      lin = lin.rstrip()
8      # print(lin)
9      wds = down
10     # pr
11     for w in
12         if w not in down:
13             down[w] = 1
14             continue
15             print(w)
16             print("**NEW**")
17             print(w)
18             print("the")
19             print("and")
20             print("the")
21             print(di)
22
{'ran': 2, 'after': 1, 'into': 1, 'tent': 2, 'and': 3, 'on': 1, 'the': 7, 'down': 1, 'fell': 1, 'clown': 2, 'car': 3}
m-c02m92uxfd57:ex_09 py4e$
```

And so that will give us the counts.

21:38

... ⌂ 41

The screenshot shows a Mac OS X desktop environment. A terminal window is open, displaying Python code and its execution output. The terminal window title is "ex_09.py — /Users/py4e/Desktop/py4e". The code reads a file named "clown.txt" and processes its contents to print word counts. The output shows that the word "and" has a count of 3, while other words like "car" and "clown" have a count of 1. The word "fall" is explicitly mentioned as having a count of 1. The terminal window also shows the file path "ex_09/ex_09.py" and the current date and time "12-34 (1, 3)". The system tray at the top right shows battery level at 41% and signal strength. The desktop background features icons for "Macintosh HD" and "Time Machine".

```
ex_09.py
1 fname = input('enter file: ')
2 if len(fname) < 1 : fname = 'clown.txt'
3 hand = open(fname)
4
5 di = dict()
6 for lin in hand:
7     lin = lin.rstrip()
8     # print(lin)
9     wds = lin.split()
10    # print(wds)
11    for w in wds:
12        print('**',w,di.get(w,-99))
13        if w in di :
14            di[w] = di[w] + 1
15        else:
16            di[w] = 1
17
18    print(w, di[w])
19
```

LF - UTF-8 Python
{'tan': 2, 'after': 1, 'into': 1, 'tear': 2, 'and': 3, 'on': 1, 'the': 1, 'down': 1, 'fell': 1, 'clown': 2, 'car': 3}

99 is the default value that we get if the key doesn't exist.

21:40

... 🔔 41

The screenshot shows a Mac OS X desktop environment. On the left, a file browser window titled 'the' displays a directory structure under 'py4e'. The 'ex_09' folder contains files: ex_09.py, clown.txt, ex_09.py, intro.txt, .DS_Store, and first.py. The 'ex_09.py' file is open in a code editor window titled 'ex_09.py — /Users/py4e/Desktop/py4e'. The code in the editor is as follows:

```
nano = open(tname)
di = dict()
for lin in hand:
    lin = lin.rstrip()
    # print(lin)
    wds = lin.split()
    # print(wds)
    for w in wds:
        print('**',w,di.get(w,-99))

        if w in di :
            di[w] = di[w] + 1
        else:
            di[w] = 1

    # print(w, di[w])
```

The terminal window at the bottom has tabs for 'clown.txt' and 'intro.txt'. The command 'python ex_09.py' is running, with the output:

```
** clown 1
** clown 1
```

A message box in the foreground says: "And so that's just this get mechanism allows us to get the new value."

21:41

... ⌂ 41

The screenshot shows a Mac desktop environment. A terminal window is open, displaying Python code for a word frequency counter. The code reads from a file named 'ex_09.py' located at '/Users/py4e/Desktop/py4e/ex_09/ex_09.py'. The code uses a dictionary to count word occurrences. A status bar message at the bottom of the terminal window says: 'Look up using the key w, which is the, and if don't get it, give me back 0.' The desktop background shows icons for 'Macintosh HD', 'Mac SD', 'py4e', and 'iShowU-Capture.mov'.

```
ex_09/ex_09.py* 13:28 (1, 1) LF UTF-8 Python
ex_09.py — /Users/py4e/Desktop/py4e
3 nano = open(rname)
4
5 di = dict()
6 for lin in hand:
7     lin = lin.rstrip()
8     # print(lin)
9     wds = lin.split()
10    # print(wds)
11    for w in wds:
12
13        oldcount = di.get(w,0)
14
15        if w in di :
16            di[w] = di[w] + 1
17        else:
18            di[w] = 1
19
20        # print(w, di[w])
21
```

** the 5
** clown 1
**

clown.txt intro.txt Show All X

16:16 ... ⌂ H 96

The screenshot shows a Mac desktop environment. On the left, there's a file browser window titled 'the' showing a directory structure under 'py4e'. In the center, a terminal window is open with the command 'ex_09.py' running. The terminal output shows the program's progress: 'ran new 2 into old 0' followed by 'int'. Below the terminal, a status bar displays 'clown.txt' and 'intro.txt'. To the right of the terminal is a dock with various application icons. The desktop background is blue, and a sidebar on the right lists external drives: 'Macintosh HD', 'Mac SD', and 'py4e'. A video file 'iShowU-Capture.mov' is also visible in the sidebar.

```
ex_09.py -- /Users/py4e/Desktop/py4e
1 if len(fname) < 1 : fname = 'clown.txt'
2 hand = open(fname)
3
4 di = dict()
5 for lin in hand:
6     lin = lin.rstrip()
7     # print(lin)
8     wds = lin.split()
9     # print(wds)
10    for w in wds:
11        # if the key is not there the count is zero
12        oldcount = di.get(w,0)
13        print(w,'old',oldcount)
14        newcount = oldcount + 1
15        di[w] = newcount
16
17        di[w] = di.get(w,0) + 1
18        print(w,'new',newcount)
19
20
21
22
23
24
25
26
27
28
```

clown.txt intro.txt

because that really combines all of
these lines into a single line, okay?

16:18

... ⊞ H 95

The screenshot shows a Mac desktop environment at 16:18. On the left is a file browser window titled 'the' showing a directory structure under 'py4e'. In the center is a terminal window titled 'ex_09.py' with the following code:

```
1 fname = input('Enter File: ')
2 if len(fname) < 1 : fname = 'clown.txt'
3 hand = open(fname)
4
5 for line in hand:
6     print(line, end='')
```

The terminal then shows two runs of the script:

```
m-c02m92uxfd57:ex_09 py4e$ python3 ex_09.py
Enter File:
{'ran': 2, 'clown': 2, 'after': 1, 'into': 1, 'and': 3, 'down': 1, 'the': 7, 'fe
ll': 1, 'tent': 2, 'on': 1, 'car': 3}
m-c02m92uxfd57:ex_09 py4e$ python3 ex_09.py
Enter File: intro.txt
```

On the right side of the screen is a dock containing several icons: 'Macintosh HD', 'Mac SD', 'py4e' (a folder icon), and 'iShowU-Capture.mov' (a video camera icon). The status bar at the bottom shows 'Show All'.

And let's run it with intro.txt.

16:18 ... ⌂ H 95

The screenshot shows a Mac desktop environment. On the left, a file browser window displays a directory structure under 'the'. It includes subfolders like 'py4e' containing files such as 'ex_02_02.py', 'ex_02_03.py', etc., and 'ex_09' containing 'clown.txt', 'ex_09.py', 'intro.txt', and 'first.py'. To the right of the file browser is a terminal window titled 'ex_09.py — /Users/py4e/Desktop/py4e'. The terminal contains the following Python code:

```
1 fname = input('Enter File: ')
2 if len(fname) < 1 : fname = 'clown.txt'
3 hand = open(fname)
4
5 di = dict()
6 for lin in hand:
7     lin = lin.rstrip()
8     # print(lin)
9     wds = lin.split()
10    # print(wds)
11    for w in wds:
12        # idiom: retrieve/create/update counter
13        di[w] = di.get(w,0) + 1
14        # print(w, 'new', di[w])
15
16 print(di)
```

The terminal window also shows the command 'ex_09/ex_09.py 16:10' at the bottom left and status indicators 'LF' and 'Python' at the bottom right. Below the terminal, a message box displays the output of the program: 'So, that was a lot of work to get to this line 16 that has the dictionary in it.' At the very bottom of the screen, there's a dock with icons for various applications like Finder, Mail, and Safari.

16:20 ... ⌂ H 95

The screenshot shows a Mac desktop environment. On the left is a file browser window titled 'the' showing a directory structure under 'py4e'. In the center is a terminal window titled 'ex_09.py — /Users/py4e/Desktop/py4e'. The terminal contains the following Python code:

```
1  fname = input('enter file: ')
2  if len(fname) < 1 : fname = 'clown.txt'
3  hand = open(fname)
4
5  di = dict()
6  for lin in hand:
7      lin = lin.rstrip()
8      wds = lin.split()
9      for w in wds:
10          # idiom: retrieve/create/update counter
11          di[w] = di.get(w,0) + 1
12
13  print(di)
14
15 #now we want to find the most common word
16 for k,v in di.items():
17
18
19
```

The terminal window also displays the output of the script:

```
: 1, 'car.': 1, 'program.': 8, 'communicate': 2, 'disk': 1, 'input/output': 1,
names': 2, 'languages.': 1)
m-c02m92uxfd57:ex_09 py4e$
```

A tooltip at the bottom of the terminal window says: "key and value, in the dictionary's name .items()."

On the right side of the desktop, there is a sidebar with icons for 'Macintosh HD', 'Mac SD', 'py4e' (a folder), and 'iShowU-Capture.mov' (a video file).

16:20 ... ⌂ H 95

The screenshot shows a Mac desktop environment. On the left is a file browser window titled 'the' showing a directory structure under 'py4e'. In the center is a terminal window titled 'ex_09.py — /Users/py4e/Desktop/py4e'. The terminal contains the following Python code:

```

ex_09.py
1 fname = input('enter file: ')
2 if len(fname) < 1 : fname = 'clown.txt'
3 hand = open(fname)
4
5 di = dict()
6 for lin in hand:
7     lin = lin.rstrip()
8     wds = lin.split()
9     for w in wds:
10         # idiom: retrieve/create/update counter
11         di[w] = di.get(w,0) + 1
12
13 print(di)
14
15 #now we want to find the most common word
16 for k,v in di.items():
17
18
19

```

The terminal also displays the output of the script, which is a dictionary of word counts:

```

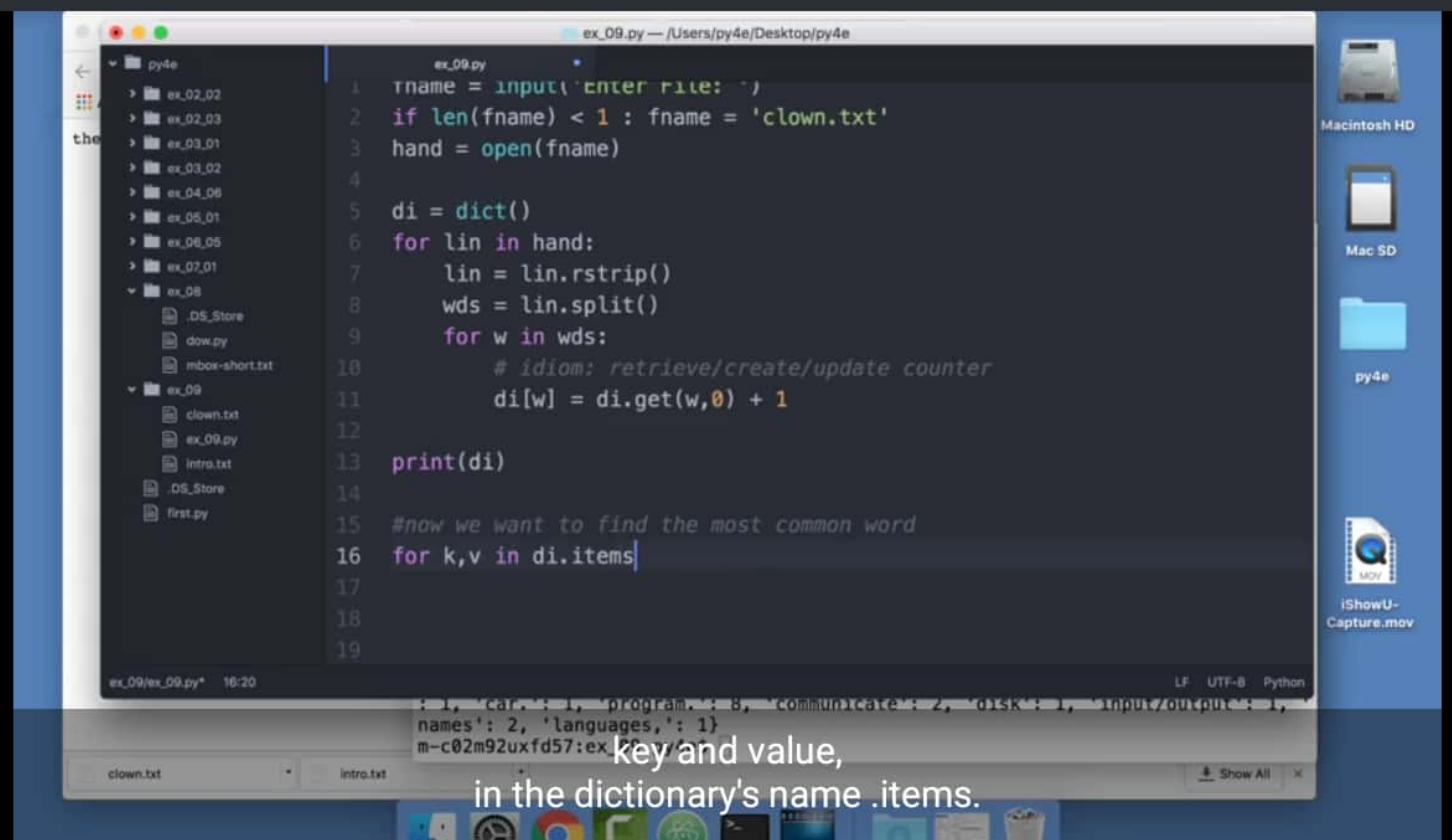
: 1, 'car.': 1, 'program.': 8, 'communicate': 2, 'disk': 1, 'input/output': 1,
names': 2, 'languages': 1}
m-c2014-07-14-16:44:42

```

A tooltip at the bottom of the terminal window says: "And items is a method inside of all dictionaries that says give me the".

On the right side of the screen, there is a dock with several icons: Macintosh HD, Mac SD, py4e, and iShowU-Capture.mov.

16:20 ... ⌂ H 95



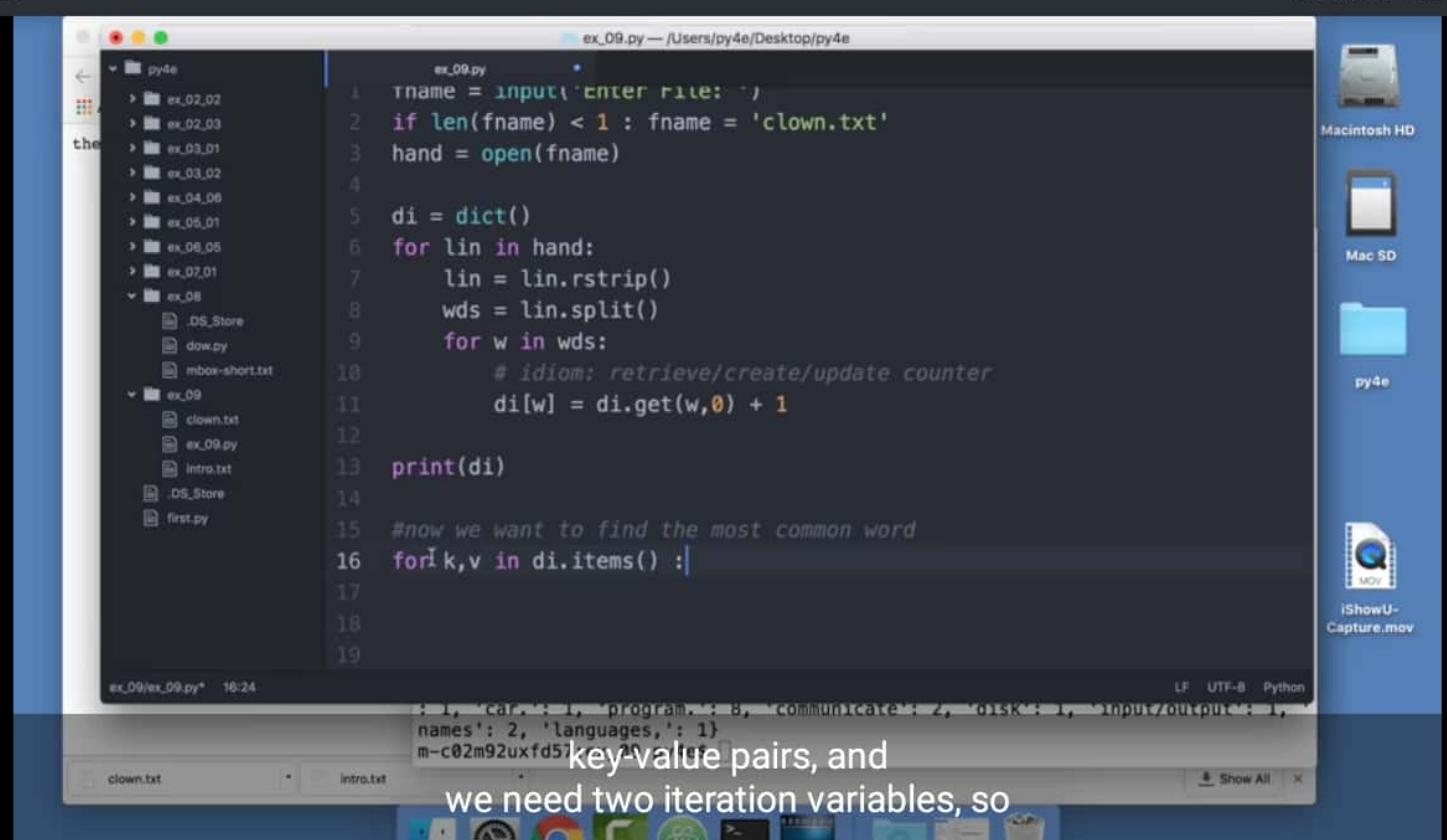
```

ex_09.py -- /Users/py4e/Desktop/py4e
1 fname = input('enter file: ')
2 if len(fname) < 1 : fname = 'clown.txt'
3 hand = open(fname)
4
5 di = dict()
6 for lin in hand:
7     lin = lin.rstrip()
8     wds = lin.split()
9     for w in wds:
10         # idiom: retrieve/create/update counter
11         di[w] = di.get(w,0) + 1
12
13 print(di)
14
15 #now we want to find the most common word
16 for k,v in di.items():
17
18
19
ex_09/ex_09.py* 16:20: LF UTF-8 Python
: 1, 'car.': 1, 'program.': 8, 'communicate': 2, 'disk': 1, 'input/output': 1,
names': 2, 'languages.': 1)
m-c02m92uxfd57:ex_09 py4e
key and value,
in the dictionary's name .items.

```

The screenshot shows a Mac desktop environment. A terminal window is open, displaying Python code for reading files and creating a dictionary. The code uses the `dict` class to count words in files like 'clown.txt' and 'intro.txt'. The terminal window has tabs for 'clown.txt' and 'intro.txt'. The desktop background shows icons for 'Macintosh HD', 'Mac SD', 'py4e', and 'iShowU-Capture.mov'. The status bar at the top right shows the time as 16:20 and battery level as 95%.

16:20 ... ⌂ H 95



```

ex_09.py -- /Users/py4e/Desktop/py4e
1  tname = input('enter file: ')
2  if len(fname) < 1 : fname = 'clown.txt'
3  hand = open(fname)
4
5  di = dict()
6  for lin in hand:
7      lin = lin.rstrip()
8      wds = lin.split()
9      for w in wds:
10          # idiom: retrieve/create/update counter
11          di[w] = di.get(w,0) + 1
12
13  print(di)
14
15  #now we want to find the most common word
16  for k,v in di.items() :
17
18
19
ex_09/ex_09.py* 16:24 LF UTF-8 Python
: 1, 'car.': 1, 'program.': 8, 'communicate': 2, 'DISK': 1, 'input/output': 1,
names': 2, 'languages.': 1}
m-c02m92uxfd57
key-value pairs, and
we need two iteration variables, so

```

The screenshot shows a Mac desktop environment. A terminal window titled 'ex_09.py' is open, displaying Python code for reading files and counting word frequencies using a dictionary. The code includes imports, file handling, and a loop that iterates over words in each line of the file to update a counter. The terminal window also shows the output of the program, which lists key-value pairs and concludes with a note about needing two iteration variables. The desktop background shows icons for Macintosh HD, Mac SD, py4e, and iShowU-Capture.mov.

I need parentheses for my print.

16:21

... ⌂ H 95

The screenshot shows a Mac desktop environment. On the left is a file browser window titled 'ex_09.py' showing a directory structure under 'the'. In the center is a terminal window titled 'ex_09.py' with the command 'python3 ex_09.py' run. The terminal output lists words from a file named 'clown.txt'. On the right is a dock with various icons, and above it is a sidebar with icons for 'Macintosh HD', 'Mac SD', 'py4e', and 'iShowU-Capture.mov'.

```
the
├── py4e
│   ├── ex_02_02
│   ├── ex_02_03
│   ├── ex_03_01
│   ├── ex_03_02
│   ├── ex_04_06
│   ├── ex_05_01
│   ├── ex_06_05
│   ├── ex_07_01
│   └── ex_08
        ├── .DS_Store
        ├── dow.py
        └── mbox-short.txt
└── ex_09
    ├── clown.txt
    ├── ex_09.py
    ├── intro.txt
    ├── .DS_Store
    └── first.py

ex_09/ex_09.py 17:16
```

```
m-c02m92uxfd57:ex_09 py4e$ python3 ex_09.py
Enter File:
the 7
on 1
after 1
down 1
tent 2
fell 1
car 3
and 3
ran 2
clown 2
into 1
# m-c02m92uxfd57:ex_09 py4e$
```

And it's kind of the same thing except it's pretty,

16:24 ... ⌂ H 95

The screenshot shows a Mac desktop environment. In the center is a terminal window titled "ex_09.py — /Users/py4e/Desktop/py4e". The code inside the terminal is as follows:

```
ex_09.py
1 lin = lin.rstrip()
2 wds = lin.split()
3 for w in wds:
4     # idiom: retrieve/create/update counter
5     di[w] = di.get(w,0) + 1
6
7 # now we want to find the most common word
8 largest = -1
9 theword = None
10 for k,v in di.items() :
11     print(k,v)
12     if v > largest :
13         largest = v
14         theword = w # capture/remember the word that was largest
15
16 print('Done',theword,largest)
```

The terminal window also displays the file path "ex_09/ex_09.py+ 24:22". At the bottom of the terminal window, there are tabs for "clown.txt" and "intro.txt", and a status bar with "LF" and "UTF-8".

On the right side of the desktop, there is a sidebar with icons for "Macintosh HD", "Mac SD", "py4e" (a folder), and "iShowU-Capture.mov".

A tooltip message is overlaid on the desktop: "So now I can print out at the end, theword and thelargest, and that's the count."

16:24

... ⌂ H 95

The screenshot shows a Mac desktop environment. On the left, a file browser window titled 'the' displays a directory structure under 'py4e'. In the center, a terminal window titled 'ex_09.py' shows the execution of a Python script. The script reads words from a file and increments their count in a dictionary. The terminal output shows the words 'clown', 'car', 'down', 'la', 'the', 'tent', 'into', 'after', 'and', 'ran', and 'on', each followed by its count. The status bar at the bottom indicates the file is 24/22 pages long. A status bar at the top right shows signal strength and battery level.

```
the
├── py4e
│   ├── ex_02_02
│   ├── ex_02_03
│   ├── ex_03_01
│   ├── ex_03_02
│   ├── ex_04_06
│   ├── ex_05_01
│   ├── ex_06_05
│   ├── ex_07_01
│   └── ex_08
        ├── .DS_Store
        ├── dow.py
        └── mbox-short.txt
└── ex_09
    ├── clown.txt
    ├── ex_09.py
    ├── intro.txt
    ├── .DS_Store
    └── first.py

ex_09/ex_09.py 24/22
```

```
ex_09.py
1 lin = lin.rstrip()
2 wds = lin.split()
3 for w in wds:
4     # idiom: retrieve/create/update counter
5     di[w] = di.get(w,0) + 1
6
7 m-c02m92uxfd57:ex_09 py4e$ python3 ex_09.py
8 Enter File:
9 clown 2
10 car 3
11 down 1
12 fell 1
13 la 1
14 the 7
15 tent 2
16 into 1
17 after 1
18 and 3
19 ran 2
20 on 1
21 Done car 7
22 m-c02m92uxfd57:ex_09 py4e$
```

Okay, and so now we know that,
Oops, did we make a mistake here?

16:25 ... ⌂ H 95

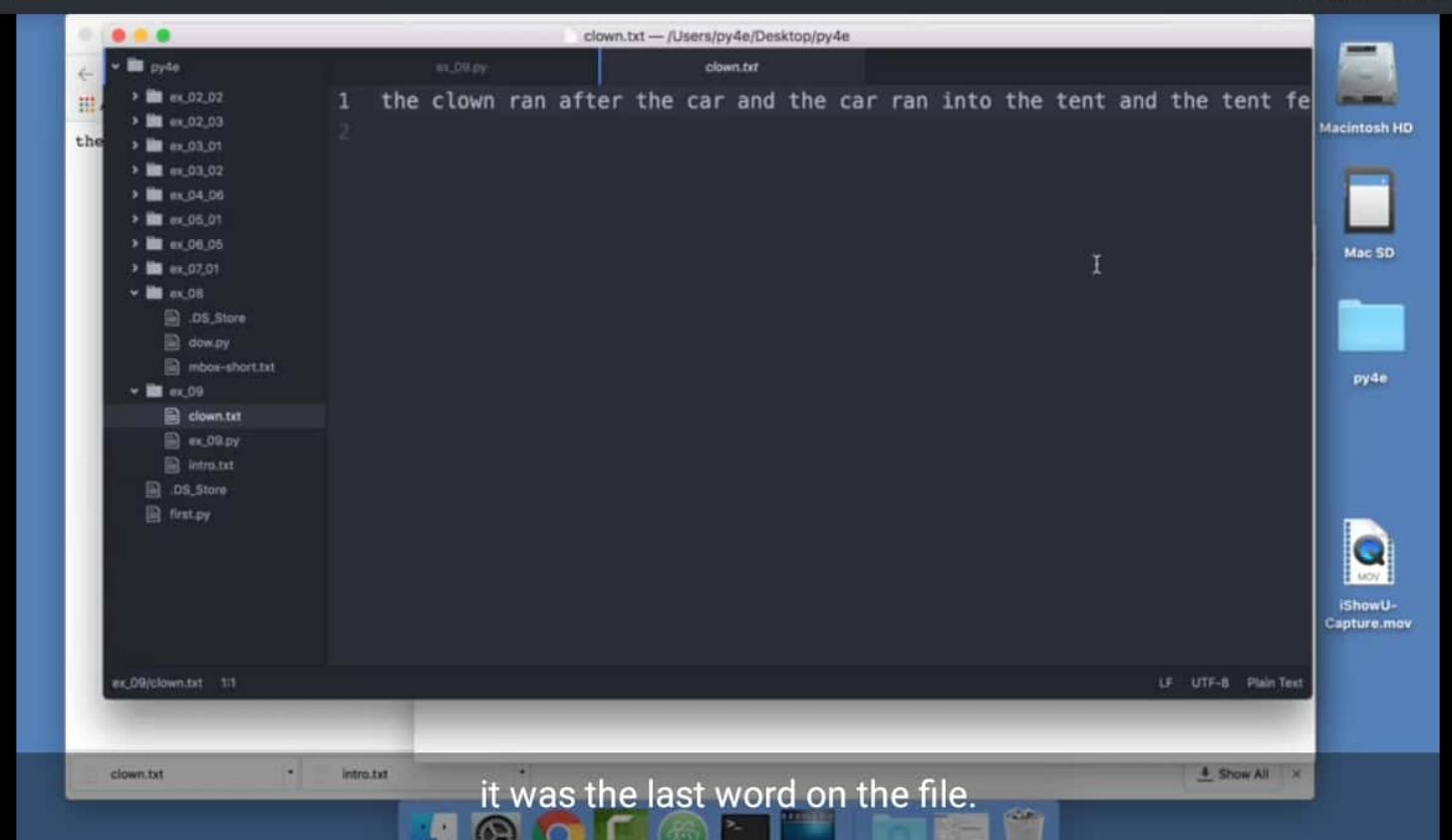
The screenshot shows a Mac desktop environment. On the left, a file browser window displays a directory structure under 'the' folder. Inside 'the' are subfolders 'py4e' and 'ex_09'. 'py4e' contains files like 'ex_02_02.py', 'ex_02_03.py', etc., and a file 'mbox-short.txt'. 'ex_09' contains files like 'clown.txt', 'ex_09.py', 'intro.txt', and 'first.py'. A terminal window titled 'ex_09.py — /Users/py4e/Desktop/py4e' is open, showing the following Python script:

```
lin = lin.rstrip()
wds = lin.split()
for w in wds:
    # idiom: retrieve/create/update counter
    di[w] = di.get(w,0) + 1
# print(di)
# now we want to find the most common word
largest = -1
theword = None
for k,v in di.items() :
    print(k,v)
    if v > largest :
        largest = v
        theword = k #capture/remember the key that was largest
print('Done',theword,largest)
```

The terminal status bar indicates 'LF UTF-8 Python'. Below the terminal, a message box says 'Key. I was going to say that was quite the bug.' The desktop background is blue, and icons for 'Macintosh HD', 'Mac SD', 'py4e', and 'iShowU-Capture.mov' are visible on the right.

16:25

... ⊞ H 95



16:25 ... ⌂ H 95

The screenshot shows a Mac desktop environment. On the left, a file browser window titled 'the' displays a directory structure under 'py4e'. The 'ex_09' folder is selected, showing files like 'clown.txt', 'intro.txt', and 'ex_09.py'. The 'ex_09.py' file is open in a terminal window titled 'ex_09.py — /Users/py4e/Desktop/py4e'. The code in the terminal is:

```
lin = lin.rstrip()
wds = lin.split()
for w in wds:
    # idiom: retrieve/create/update counter
    di[w] = di.get(w,0) + 1
# print(di)
# now we want to find the most common word
largest = -1
theword = None
for k,v in di.items() :
    print(k,v)
    if v > largest :
        largest = v
        theword = k # capture/remember the key that was largest
print('Done',theword,largest)
```

The terminal window also shows the status bar with 'ex_09/ex_09.py 22:20 (1, 1)' and encoding information 'LF UTF-8 Python'. To the right of the terminal is a dock with icons for 'Macintosh HD', 'Mac SD', 'py4e' (a folder), and 'iShowU-Capture.mov' (a video file). The desktop background is blue.

The and 7.

16:25

...  H 95

The screenshot shows a Mac desktop environment. On the left, a file browser window titled 'py4e' is open, showing a directory structure with files like ex_02_02, ex_02_03, ex_03_01, ex_03_02, ex_04_06, ex_05_01, ex_06_05, ex_07_01, ex_08, ex_09, clown.txt, intro.txt, first.py, and ex_09.py. The file 'ex_09.py' is selected. In the center, a terminal window titled 'ex_09 -- bash -- 80x24' is running the command 'python3 ex_09.py'. The output shows the program asking for an input file, which is then processed. On the right side of the desktop, there are icons for 'Macintosh HD', 'Mac SD', 'py4e' (a folder), and 'iShowU-Capture.mov' (a video file). At the bottom, the Dock contains icons for various applications.

```
ex_09.py -- /Users/py4e/Desktop/my4e
the
py4e
  ex_02_02
  ex_02_03
  ex_03_01
  ex_03_02
  ex_04_06
  ex_05_01
  ex_06_05
  ex_07_01
  ex_08
    .DS_Store
    dow.py
    mbox-short.txt
  ex_09
    clown.txt
    ex_09.py
    intro.txt
    .DS_Store
    first.py

ex_09/ex_09.py 23:7

ex_09.py
1   for lin in name:
2       lin = lin.rstrip()
3       wds = lin.split()
4       for w in wds:
5           # idiom: retrieve/create/update counter
6           print(w)
7
8
9
10
11 m-c02m92uxfd57:ex_09 py4e$ python3 ex_09.py
Enter File:
the 7
m-c02m92uxfd57:ex_09 py4e$
```

Tuples Are Like Lists

Tuples are another kind of sequence that functions much like a list
- they have elements which are indexed starting at 0

```
' '
>>> x = ('Glenn', 'Sally', 'Joseph')      >>> for iter in y:
>>> print(x[2])                         ...     print(iter)
Joseph                                ...
>>> y = ( 1, 9, 2 )                      1
>>> print(y)                           9
(1, 9, 2)                            2
>>> print(max(y))                     >>>
9
```

So three-tuples,
it's a set of three things, basically.



16:29

... 🔍 4G H 94



The place where there are differences
are that tuples are not changeable,

M

but... Tuples are “immutable”

Unlike a list, once you create a **tuple**, you cannot alter its contents - similar to a string

```
>>> x = [9, 8, 7]
>>> x[2] = 6
>>> print(x)
>>> [9, 8, 6]
```

```
>>> y = 'ABC'
>>> y[2] = 'D'
Traceback: 'str' object does not support item Assignment
>>>
```

```
>>> z = (5, 4, 3)
>>> z[2] = 0
Traceback: 'tuple' object does not support item Assignment
>>>
```

And it says sorry, you cannot change,
so these are not mutable.



A Tale of Two Sequences

```
>>> l = list()  
>>> dir(l)  
['append', 'count', 'extend', 'index', 'insert', 'pop',  
'remove', 'reverse', 'sort']  
  
>>> t = tuple()  
>>> dir(t)  
['count', 'index']
```

you can't do append or extend or
insert, pop, remove.



Tuples Are More Efficient

- Since Python does not have to build tuple structures to be modifiable, they are simpler and more efficient in terms of memory use and performance than lists
- So in our program when we are making “temporary variables” we prefer tuples over lists

If you need a list, use a list, but if you can get away with a tuple we tend to

03:14

17:54

10

K

Tuples and Assignment

- We can also put a tuple on the left-hand side of an assignment statement
- We can even omit the parentheses



```
>>> (x, y) = (4, 'fred')
>>> print(y)
fred
>>> (a, b) = (99, 98)
>>> print(a)
99
```

So it's the same as saying x equals 4,
and y equals 'fred'.



Tuples and Assignment

- We can also put a tuple on the left-hand side of an assignment statement
- We can even omit the parentheses

```
>>> (x, y) = (4, 'fred')
>>> print(y)
fred
>>> (a, b) = (99, 98)
>>> print(a)
99
```

(x, y) = (4, 'fred')

x = 4

y = "fred"

(a, b) = 9

It's going to be, I'm unhappy about that,
because it expects if there's a tuple on

Tuples and Dictionaries

The `items()` method in dictionaries returns a list of (key, value) tuples

```
>>> d = dict()
>>> d['csev'] = 2
>>> d['cwen'] = 4
>>> for (k,v) in d.items():
...     print(k, v)
...
csev 2
cwen 4
>>> tups = d.items()
>>> print(tups)
dict_items([('csev', 2), ('cwen', 4)])
```

`d.items` gives you a list of tuples.



Tuples are Comparable

The comparison **operators** work with **tuples** and other sequences. If the first item is equal, Python goes on to the next element, and so on, until it finds elements that differ.

```
>>> (0, 1, 2) < (5, 1, 2)
True
>>> (0, 1, 2000000) < (0, 3, 4)
True
>>> ('Jones', 'Sally') < ('Jones', 'Sam')
True
>>> ('Jones', 'Sally') > ('Adams', 'Sam')
True
```

That's like the most significant element of the tuple, like the most significant digit of

Tuples are Comparable

The comparison **operators** work with **tuples** and other sequences. If the first item is equal, Python goes on to the next element, and so on, until it finds elements that differ.

```
>>> (0, 1, 2) < (5, 1, 2)
True
>>> (0, 1, 2000000) < (0, 3, 4)
True
>>> ('Jones', 'Sally') < ('Jones', 'Sam')
True
>>> ('Jones', 'Sally') > ('Adams', 'Sam')
True
```



It compares these two



Tuples are Comparable

The comparison **operators** work with **tuples** and other sequences. If the first item is equal, Python goes on to the next element, and so on, until it finds elements that differ.

```
>>> (0, 1, 2) < (5, 1, 2)
True
>>> (0, 1, 2000000) < (0, 3, 4)
True
>>> ('Jones', 'Sally') < ('Jones', 'Sam')
True
>>> ('Jones', 'Sally') > ('Adams', 'Sam')
True
```



And then at some point goes oh,
there's the l and that m, and so

Tuples are Comparable

The comparison **operators** work with **tuples** and other sequences. If the first item is equal, Python goes on to the next element, and so on, until it finds elements that differ.

```
>>> (0, 1, 2) < (5, 1, 2)
True
>>> (0, 1, 2000000) < (0, 3, 4)
True
>>> ('Jones', 'Sally') < ('Jones', 'Sam')
True
>>> ('Jones', 'Sally') > ('Adams', 'Sam')
True
```

Adams, then it never looks at the second piece whatsoever.



Sorting Lists of Tuples

- We can take advantage of the ability to sort a list of tuples to get a sorted version of a dictionary
- First we sort the dictionary by the key using the `items()` method and `sorted()` function

```
>>> d = {'a':10, 'b':1, 'c':22} ↴
>>> d.items()
dict_items([('a', 10), ('c', 22), ('b', 1)])
>>> sorted(d.items())
[('a', 10), ('b', 1), ('c', 22)]
    [           |           ]
```

You can't have two a's or two b's.



16:37

... 🔍 4G H 94



you can't put the same
key in more than once.

Using sorted()

We can do this even more directly using the built-in function `sorted` that takes a sequence as a parameter and returns a sorted sequence

```
>>> d = {'a':10, 'b':1, 'c':22}
>>> t = sorted(d.items())
>>> t
[('a', 10), ('b', 1), ('c', 22)]
>>> for k, v in sorted(d.items()):
...     print(k, v)
...
a 10
b 1
c 22
```

So this is a way to say I want to loop through this dictionary in key order.



Using sorted()

We can do this even more directly using the built-in function `sorted` that takes a sequence as a parameter and returns a sorted sequence

```
>>> d = {'a':10, 'b':1, 'c':22}
>>> t = sorted(d.items())
>>> t
[('a', 10), ('b', 1), ('c', 22)]
>>> for k, v in sorted(d.items()):
...     print(k, v)
...
a 10
b 1
c 22
```

I N
K E Y
O R D S

So this is a way to say I want to loop through this dictionary in key order.



Sort by Values Instead of Key

- If we could construct a list of tuples of the form (value, key) we could sort by value
- We do this with a for loop that creates a list of tuples

```
>>> c = {'a':10, 'b':1, 'c':22}
>>> tmp = list()
>>> for k, v in c.items() :
...     tmp.append( (v, k) )
...
>>> print(tmp)
[(10, 'a'), (22, 'c'), (1, 'b')]
>>> tmp = sorted(tmp, reverse=True)
>>> print(tmp)
[(22, 'c'), (10, 'a'), (1, 'b')]
```

then it's going to compare these things and then sort, and when those match.



Sort by Values Instead of Key

- If we could construct a list of tuples of the form (value, key) we could sort by value
- We do this with a for loop that creates a list of tuples

```
>>> c = {'a':10, 'b':1, 'c':22}
>>> tmp = list()
>>> for k, v in c.items() :
...     tmp.append( (v, k) )
...
>>> print(tmp)
[(10, 'a'), (22, 'c'), (1, 'b')]
>>> tmp = sorted(tmp, reverse=True)
>>> print(tmp)
[(22, 'c'), (10, 'a'), (1, 'b')]
```

then it's going to compare these things and then sort, and when those match.



Sort by Values Instead of Key

- If we could construct a list of tuples of the form (value, key) we could sort by value
- We do this with a for loop that creates a list of tuples

```
>>> c = {'a':10, 'b':1, 'c':22}
>>> tmp = list()
>>> for k, v in c.items() :
...     tmp.append( (v, k) )
...
>>> print(tmp)
[(10, 'a'), (22, 'c'), (1, 'b')]
>>> tmp = sorted(tmp, reverse=True)
>>> print(tmp)
[(22, 'c'), (10, 'a'), (1, 'b')]
```

then it will look to the key as the next thing to sort based on.



```

fhand = open('romeo.txt')
counts = dict()
for line in fhand:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1

lst = list()
for key, val in counts.items():
    newtup = (val, key)
    lst.append(newtup)

lst = sorted(lst, reverse=True)
for val, key in lst[:10] :
    print(key, val)

```

The top 10 most common words



right here, we've got the completed histogram in no particular order.

```

fhand = open('romeo.txt')
counts = dict()
for line in fhand:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1

lst = list()
for key, val in counts.items():
    newtup = (val, key)
    lst.append(newtup)
lst = sorted(lst, reverse=True)

for val, key in lst[:10]:
    print(key, val)

```

The top 10 most common words



So that's 0 through 9 which is the first ten.

Even Shorter Version

```
>>> c = {'a':10, 'b':1, 'c':22}  
>>> print( sorted( [ (v,k) for k,v in c.items() ] ) )  
[(1, 'b'), (10, 'a'), (22, 'c')]
```

List comprehension creates a dynamic list. In this case, we make a list of reversed tuples and then sort it.

<http://wiki.python.org/moin/HowTo/Sorting>

It's a three-tuple list
flipped with values and keys.

16:51

... ⌂ Thu 1:13 PM

The screenshot shows a Mac OS X desktop environment. In the foreground, a terminal window titled 'ex_10.py' is open, displaying Python code. The code reads a file name from input, sets it to 'clown.txt' if empty, opens the file, creates a dictionary, and prints its items. A command is run to execute the script. In the background, a file browser window titled 'py4e' is visible, showing a directory structure with files like 'ex_02_02', 'ex_02_03', etc., and sub-directories 'ex_09'. A file named 'ex_10.py' is selected in the browser. The desktop background shows icons for 'Macintosh HD' and 'Time Machine'.

```
py4e
├── ex_02_02
├── ex_02_03
├── ex_03_01
├── ex_03_02
├── ex_04_06
├── ex_05_01
├── ex_06_05
├── ex_07_01
└── ex_08
    ├── .DS_Store
    ├── down.py
    └── mbox-short.txt

Why
and
man
a d
som
eve
you
to
We
fro
as
on
is
Wha
Pro
to
Ass
us
Our
cou
spe
do
com
Int
are often the kinds of things that we humans f
and mind-numbing.

ex_09/ex_10.py 16:51
```

```
1 fname = input('Enter File: ')
2 if len(fname) < 1 : fname = 'clown.txt'
3 hand = open(fname)
4
5 di = dict()
6 for line in hand:
7     line = line.rstrip()
8     wds = line.split(' ')
9     for word in wds:
10         if word in di:
11             di[word] += 1
12         else:
13             di[word] = 1
14
15 print(di)
16
17
```

```
m-c02m92uxfd57:ex_09 py4e$ python3 ex_10.py
Enter File:
{'ran': 2, 'down': 1, 'and': 3, 'clown': 2, 'car': 3, 'into': 1, 'tent': 2, 'the': 7, 'on': 1, 'after': 1, 'fell': 1}
wds = dict_items([('ran', 2), ('down', 1), ('and', 3), ('clown', 2), ('car', 3), ('into', 1), ('tent', 2), ('the', 7), ('on', 1), ('after', 1), ('fell', 1)])
m-c02m92uxfd57:ex_09 py4e$
```

use items it gives us the key-value pairs.

16:52

... ⌂ Thu 1:14 PM

The screenshot shows the Atom code editor on a Mac OS X desktop. The window title is "ex_10.py — /Users/py4e/Desktop/py4e". The file content is as follows:

```
1 fname = input('Enter File: ')
2 if len(fname) < 1 : fname = 'clown.txt'
3 hand = open(fname)
4
5 di = dict()
6 for lin in hand:
7     lin = lin.rstrip()
8     wds = lin.split()
9     for w in wds:
10         # idiom: retrieve/create/update counter
11         di[w] = di.get(w,0) + 1
12
13 print(di)
14
15 x = sorted(di.items())
16 print(x[:5])
```

The sidebar on the left lists several Python files in the "py4e" directory, including "ex_02_02", "ex_02_03", "ex_03_01", "ex_03_02", "ex_04_06", "ex_05_01", "ex_06_05", "ex_07_01", "ex_08", "ex_09", "clown.txt", "ex_09.py", "ex_10.py", "intro.txt", ".DS_Store", and "first.py". The status bar at the bottom shows "ex_09/ex_10.py 16:51".

16:53

... ⌂ Thu 1:14 PM

The screenshot shows a Mac OS X desktop environment. In the foreground, a terminal window titled "ex_10.py" is open, displaying Python code. The code reads a file name from input, sets it to "clown.txt" if empty, opens the file, and then processes its contents into a dictionary. The terminal window also shows the output of running the script with "python3 ex_10.py". In the background, a file browser window titled "py4e" is visible, showing a directory structure with files like "ex_02_02", "ex_02_03", etc., and sub-directories "ex_09" and "ex_10". A file named "intro.txt" is selected in the browser. The desktop background features a Macintosh HD icon.

```
py4e
> ex_02_02
> ex_02_03
> ex_03_01
> ex_03_02
> ex_04_06
> ex_05_01
> ex_06_05
> ex_07_01
> ex_08
>   .DS_Store
>   down.py
>   mbox-short.txt
> ex_09
>   clown.txt
>   ex_09.py
> ex_10.py
>   intro.txt
>   .DS_Store
> first.py

ex_10.py
1 fname = input('Enter File: ')
2 if len(fname) < 1 : fname = 'clown.txt'
3 hand = open(fname)
4
5 di = dict()
6 for line in hand:
7     line = line.rstrip()
8     words = line.split()
9     for word in words:
10        if word in di:
11            di[word] += 1
12        else:
13            di[word] = 1
14
15 x = sorted(di.items())
16
17 print(x)

m-c02m92uxfd57:ex_09 py4e$ python3 ex_10.py
Enter File:
clown.txt
{'after': 1, 'tent': 2, 'down': 1, 'ran': 2, 'clown': 2, 'car': 3, 'into': 1, 'on': 1, 'fell': 1, 'and': 3, 'the': 7}
wds = [('after', 1), ('ang', 3), ('car', 3), ('clown', 2), ('down', 1), ('fell', 1), ('into', 1), ('on', 1), ('ran', 2), ('tent', 2), ('the', 7)]
m-c02m92uxfd57:ex_09 py4e$
```

Int. ex_09/ex_10.py 16:23
are often the kinds of things that we humans f
and mind-numbing.

And now is after, and, car,
it's in alphabetical order by key.

16:55

... ⌂ Thu 1:16 PM

The screenshot shows a Mac desktop with the Atom code editor open. The window title is "ex_10.py — /Users/py4e/Desktop/py4e". The code editor displays the following Python script:

```
ex_10.py
4
5  di = dict()
6  for lin in hand:
7      lin = lin.rstrip()
8      wds = lin.split()
9      for w in wds:
10          # idiom: retrieve/create/update counter
11          di[w] = di.get(w,0) + 1
12
13  print(di)
14
15  tmp = list()
16  for k,v in di.items() :
17      # print(k,v)
18      newt = (v,k)
19      tmp.append(newt)
20
21  print('Flipped',tmp)
22
```

The status bar at the bottom indicates "ex_09/ex_10.py* 21:17". Below the editor, a terminal window shows the output of the script:

```
I mean, it's not sorted,
it's flipped, let's print it.
```

16:56

... ⌂ Thu 1:17 PM

The screenshot shows the Atom code editor window. The title bar says "ex_10.py — /Users/py4e/Desktop/py4e". The left sidebar shows a file tree with a "py4e" folder containing several "ex_xx.py" files and some text files like "mbox-short.txt", "clown.txt", and "intro.txt". The main editor area contains the following Python code:

```
for lin in nando:
    lin = lin.rstrip()
    wds = lin.split()
    for w in wds:
        # idiom: retrieve/create/update counter
        di[w] = di.get(w,0) + 1

print(di)

tmp = list()
for k,v in di.items() :
    # print(k,v)
    newt = (v,k)
    tmp.append(newt)

print('Flipped',tmp)
```

The status bar at the bottom shows "ex_09/ex_10.py* 23:4". A tooltip at the bottom center says "We can say tmp equals sorted(tmp)".

16:56

... ⌂ Thu 1:17 PM

The screenshot shows a Mac OS X desktop environment. In the foreground, a terminal window titled 'ex_10.py' is open, displaying Python code. The code reads a file 'clown.txt', processes it, and prints the result. A second terminal window titled 'ex_09' is visible in the background, showing the output of the script. To the left of the terminals, a file browser window is open, showing a directory structure with files like 'ex_01.py' through 'ex_10.py'. The status bar at the bottom of the screen displays the text 'Okay, so here's the first print. When we flip it, we've got 2 tent.'

```
ex_10.py
lin = lin.rstrip()
wds = lin.split()
for w in wds:
    # idiom: retrieve/create/update counter
    di[...] = di.get(w, 0) + 1
print(di)

Enter File:
m-c02m92uxfd57:ex_09 py4e$ 
m-c02m92uxfd57:ex_09 py4e$ python3 ex_10.py
{'tent': 2, 'after': 1, 'down': 1, 'ran': 2, 'into': 1, 'the': 7, 'on': 1, 'clown': 2, 'and': 3, 'fell': 1, 'car': 3}
Flipped [(2, 'tent'), (1, 'after'), (1, 'down'), (2, 'ran'), (1, 'into'), (7, 'the'), (1, 'on'), (2, 'clown'), (3, 'and'), (1, 'fell'), (3, 'car')][]
Sorted [(1, 'after'), (1, 'down'), (1, 'fell'), (1, 'into'), (1, 'on'), (2, 'clown'), (2, 'ran'), (2, 'tent'), (3, 'and'), (3, 'car'), (7, 'the')]
newt = tmp.ap
tmp.ap
m-c02m92uxfd57:ex_09 py4e$ 

print('Fl')
print('Sorted')
tmp = sorted(tmp.items())
print('Sorted')
for k, v in tmp:
    print(k, v)
print('')

print('Flipped')
tmp = sorted(tmp.items(), reverse=True)
print('Flipped')
for k, v in tmp:
    print(k, v)
print('')

print('Newt')
newt = dict((v, k) for k, v in tmp)
print('Newt')
for k, v in newt.items():
    print(v, k)
print('')

print('Intro')
with open('intro.txt') as f:
    intro = f.read()
print(intro)

print('Clown')
with open('clown.txt') as f:
    clown = f.read()
print(clown)

print('Done')
print('Done')
```

Okay, so here's the first print.
When we flip it, we've got 2 tent.

16:58

... ⌂ Thu 1:18 PM

The screenshot shows the Atom code editor on a Mac desktop. The file ex_10.py is open, displaying Python code. A tooltip for the 'try' keyword is visible, listing four related suggestions: 'try', 'trye', 'tryef', and 'tryf'. The code itself includes imports from re and collections, defines a function to flip a list of words, and prints the sorted and reversed result. The status bar at the bottom indicates the file is 2329 bytes long and contains a note about intro.txt.

```
ex_10.py
lin = lin.rstrip()
wds = lin.split()
for w in wds:
    # idiom: retrieve/create/update counter
    di[w] = di.get(w,0) + 1

print(di)

tmp = list()
for k,v in di.items() :
    # print(k,v)
    newt = (v,k)
    tmp.append(newt)

print('Flipped',tmp)

tmp = sorted(tmp, reverse=True)
print('Sorted',tmp)
```

Int 2329
are often the kinds of things that we humans f and mind-numbing.

try Try/Except
trye Try/Except/EL
tryef Try/Except/EL
tryf Try/Except/Fi

And we just say,

hey sorted, do this backwards,

16:58

... ⌂ Thu 1:18 PM

The screenshot shows the Atom code editor on a Mac desktop. The window title is "ex_10.py — /Users/py4e/Desktop/py4e". The code editor displays the following Python script:

```
    ex_10.py
    lin = lin.rstrip()
    wds = lin.split()
    for w in wds:
        # idiom: retrieve/create/update counter
        di[w] = di.get(w,0) + 1

    print(di)
    tmp = list()
    for k,v in di.items() :
        # print(k,v)
        newt = (v,k)
        tmp.append(newt)

    print('Flipped',tmp)
    tmp = sorted(tmp, reverse=True)
    print('Sorted',tmp[:5])
```

The file path in the sidebar is "/Users/py4e/Desktop/py4e/ex_10.py". The sidebar also lists other files like ex_02_02, ex_02_03, etc. The status bar at the bottom shows "ex_09/ex_10.py 24:23". A message in the status bar says "are often the kinds of things that we humans f and mind-numbing.". The bottom of the screen has a decorative footer with various icons.

We can say up to but not including 5.

16:58

... ⌂ Thu 1:18 PM

The screenshot shows a Mac OS X desktop environment. In the foreground, a terminal window titled 'ex_10.py' is open, displaying Python code. The code reads words from a file, splits them into a list, and then prints the sorted list. A message box is overlaid on the terminal window, showing the output of the script. The output shows the words 'clown', 'down', 'tent', 'on', 'into', 'car', 'fell', 'ran', 'after', and 'the' sorted by frequency. Below the terminal, the desktop background features a dark blue gradient with various icons.

```
py4e
> ex_02_02
> ex_02_03
> ex_03_01
> ex_03_02
> ex_04_06
> ex_05_01
> ex_06_05
> ex_07_01
> ex_08
>   .DS_Store
>   down.py
>   mbox-short.txt
> ex_09
>   clown.txt
>   ex_09.py
> ex_10.py
>   intro.txt
>   .DS_Store
>   first.py
> ex_09/ex_10.py 24:23
Int
are often the kinds of things that we humans f
and mind-numbing.

m-c02m92uxfd57:ex_09 py4e$ python3 ex_10.py
Enter File:
('clown': 2, 'down': 1, 'tent': 2, 'on': 1, 'into': 1, 'car': 3, 'fell': 1, 'ran': 2, 'after': 1, 'the': 7, 'and': 3}
Flipped [(2, 'clown'), (1, 'down'), (2, 'tent'), (1, 'on'), (1, 'into'), (3, 'car'), (1, 'fell'), (2, 'ran'), (1, 'after'), (7, 'the'), (3, 'and')]
Sorted [(7, 'the'), (3, 'car'), (3, 'and'), (2, 'tent'), (2, 'ran')]

m-c02m92uxfd57:ex_09 py4e$
```

So the sorted one is that's the top five.

17:00

... ⌂ Thu 1:20 PM

The screenshot shows the Atom code editor window. The title bar says "ex_10.py — /Users/py4e/Desktop/py4e". The left sidebar shows a file tree with a folder "py4e" containing several subfolders and files, including "ex_02_02", "ex_02_03", "ex_03_01", "ex_03_02", "ex_04_06", "ex_05_01", "ex_06_05", "ex_07_01", "ex_08", "ex_09", "ex_09.py", "ex_10.py", "intro.txt", ".DS_Store", and "first.py". The right sidebar shows a file list with "Macintosh HD" at the top, followed by "clown.txt" and "after.txt". The main editor area contains the following Python code:

```
    ex_10.py
    wds = lin.split()
    for w in wds:
        # idiom: retrieve/create/update counter
        di[w] = di.get(w,0) + 1
    # print(di)
    tmp = list()
    for k,v in di.items() :
        newtI= (v,k)
        tmp.append(newtI)
    # print('Flipped',tmp)
    tmp = sorted(tmp, reverse=True)
    # print('Sorted',tmp[:5])
    for v,k in tmp[:5] :
        print(k,v)
```

The status bar at the bottom shows "ex_09/ex_10.py 17:17 (1,6)". Below the editor, a message box displays:

Then we flip them back with key-value and
print them out.

17:00

