Introduction to Python
Exceptions, Unicode, Text Processing

Christopher Barker

UW Continuing Education / Isilon

July 11, 2012

Table of Contents

title

Lab from end of last class?

LAB

`def count_them(letter):`

- prompts the user to input a letter
- counts the number of times the given letter is input
- prompts the user for another letter
- continues until the user inputs "x"
- returns the count of the letter input

`def count_letter_in_string(string, letter):`

- counts the number of instances of the letter in the string
- ends when a period is encountered
- if no period is encountered – prints "hey, there was no period!"

Questions?

Any Questions about:

- Last class ?
- Reading ?
- Homework ?

Homework review

Homework notes

subprocesses

Subprocesses

```
#easy:
os.popen('ls').read()

#even easier:
os.system('ls')

# but for anything more complicated:
pipe = \
  subprocess.Popen("ls", stdout=subprocess.PIPE).stdout
```

reload

module importing and reloading

```
In [190]: import module_reload

In [191]: module_reload.print_something()
I'm printing something

# change it...
In [196]: reload(module_reload)
Out[196]: <module 'module_reload' from 'module_reload.py'>

In [193]: module_reload.print_something()
I'm printing something else
```

Module Reloading

```
In [194]: from module_reload import this

# change it...

In [196]: reload(module_reload)
Out[196]: <module 'module_reload' from 'module_reload.py'>

In [197]: module_reload.this
Out[197]: 'this2'

In [198]: this
Out[198]: 'this'
```

repr vs. str

repr() vs str()

```
In [200]: s = "a string\nwith a newline"

In [203]: print str(s)
a string
with a newline

In [204]: print repr(s)
'a string\nwith a newline'
```

repr vs. str

```
eval(repr(something)) == something
```

```
In [205]: s2 = eval(repr(s))

In [206]: s2
Out[206]: 'a string\nwith a newline'
```

Strings

A string literal creates a string type

```
"this is a string"
```

Can also use `str()`

```
In [256]: str(34)
Out[256]: '34'
```

or "back ticks"

```
In [258]: `34`
Out[258]: '34'
```

(demo)

The String Type

Lots of nifty methods:

```
s.lower()
s.upper()
     ...
s.capitalize()
s.swapcase()
s.title()
```

`http://docs.python.org/library/stdtypes.html#index-23`

The String Type

Lots of nifty methods:

```
x in s
s.startswith(x)
s.endswith
...
s.index(x)
s.find(x)
s.rfind(x)
```

http://docs.python.org/library/stdtypes.html#index-23

The String Type

Lots of nifty methods:

```
s.split()
s.join(list)
...
s.splitlines()
```

`http://docs.python.org/library/stdtypes.html#index-23`

(demo – join)

String Literals
Common Escape Sequences

`\\`	Backslash (`\`)
`\a`	ASCII Bell (BEL)
`\b`	ASCII Backspace (BS)
`\n`	ASCII Linefeed (LF)
`\r`	ASCII Carriage Return (CR)
`\t`	ASCII Horizontal Tab (TAB)
`\ooo`	Character with octal value ooo
`\xhh`	Character with hex value hh

(http://docs.python.org/release/2.5.2/ref/strings.html)

Raw Strings

Escape Sequences Ignored

```
In [408]: print "this\nthat"
this
that
In [409]: print r"this\nthat"
this\nthat
```

Gotcha:

```
In [415]: r"\"
SyntaxError: EOL while scanning string literal
```

(http://docs.python.org/release/2.5.2/ref/strings.html)

Building Strings

Please don’t do this:

```
'Hello ' + name + '!'
```

(much)

Building Strings

Do this instead:

```
'Hello %s!' % name
```

much faster and safer:
easier to modify as code gets complicated
`http://docs.python.org/library/stdtypes.html#string-formatting-operations`

Joining Strings

The Join Method:

```
In [289]: t = ("some", "words","to","join")

In [290]: " ".join(t)
Out[290]: 'some words to join'

In [291]: ",".join(t)
Out[291]: 'some,words,to,join'

In [292]: "".join(t)
Out[292]: 'somewordstojoin'

In [293]: "\n".join(t)
Out[293]: 'some\nwords\nto\njoin'
```

String Formatting

The string format operator: %

```
In [261]: "an integer is: %i"%34
Out[261]: 'an integer is: 34'

In [262]: "a floating point is: %f"%34.5
Out[262]: 'a floating point is: 34.500000'

In [263]: "a string is: %s"%"anything"
Out[263]: 'a string is: anything'
```

String Formatting

multiple arguments:

```
In [264]: "the number %s is %i"%('five', 5)
Out[264]: 'the number five is 5'

In [266]: "the first 5 numbers are: %i, %i, %i, %i, %i"%(1,
Out[266]: 'the first 5 numbers are: 1, 2, 3, 4, 5'
```

String formatting

Gotcha

```
In [127]: "this is a string with %i formatting item"%1
Out[127]: 'this is a string with 1 formatting item'

In [128]: "string with %i formatting %s: "%2, "items"
TypeError: not enough arguments for format string

# Done right:
In [131]: "string with %i formatting %s"%(2, "items")
Out[131]: 'string with 2 formatting items'

In [132]: "string with %i formatting item"%(1,)
Out[132]: 'string with 1 formatting item'
```

String formatting

Named arguments

```
'Hello %(name)s!'%{'name':'Joe'}
'Hello Joe!'

'Hello %(name)s, how are you, %(name)s!' %{'name':'Joe'}
'Hello Joe, how are you, Joe!'
```

That last bit is a dictionary (next week)

Advanced Formatting

The format method

```
In [283]: 'Hello {0}!'.format(name)
Out[283]: 'Hello Joe!'

In [284]: 'Hello {name}!'.format(**dictionary)
Out[284]: 'Hello Joe!'
```

pick one (probably string formatting):
get comfy with it

LAB

Format operators:

- rewrite:

  `the first 3 numbers are: %i, %i, %i"%(1,2,3)`

  for an arbitrary number of numbers...

Files

Text Files

```
f = open('secrets.txt')
secret_data = f.read()
f.close()
```

`secret_data` is a string

(can also use `file()` – `open()` is preferred)

Files

Binary Files

```
f = open('secrets.txt', 'rb')
secret_data = f.read()
f.close()
```

`secret_data` is still a string

(with arbitrary bytes in it)

(See the `struct` module to unpack binary data)

Files

File Opening Modes

```
f = open('secrets.txt', [mode])
```

```
'r', 'w', 'a'
'rb', 'wb', 'ab'
r+, w+, a+
r+b, w+b, a+b
U
U+
```

Gotcha – w mode always clears the file

Text File Notes

Text is default

- Newlines are translated: `\r\n -> \n`
- – reading and writing!
- Use *nux-style in your code: `\n`
- Open text files with `'U'` "Universal" flag

Gotcha:

- no difference between text and binary on *nix
    - breaks on Windows

File Reading

Reading Part of a file

```
header_size = 4096

f = open('secrets.txt')
secret_data = f.read(header_size)
f.close()
```

File Reading

Common Idioms

```
for line in open('secrets.txt'):
    print line

f = open('secrets.txt')
while True:
    line = f.readline()
    if not line:
        break
    do_something_with_line()
```

File Writing

```
outfile = open('output.txt')

for i in range(10):
    outfile.write("this is line: %i\n"%i)
```

File Methods

Commonly Used Methods

```
f.read() f.readline()  f.readlines()

f.write(str) f.writelines(seq)

f.seek(offset)   f.tell()

f.flush()

f.close()
```

File Like Objects

File-like objects

Many classes impliment the file interface:

- loggers
- `sys.stdout`
- `urllib.open()`
- pipes, subprocesses
- StringIO

`http://docs.python.org/library/stdtypes.html#bltin-file-objects`

StringIO

StringIO

```
In [417]: import StringIO
In [420]: f = StringIO.StringIO()

In [421]: f.write("somestuff")

In [422]: f.seek(0)

In [423]: f.read()
Out[423]: 'somestuff'
```

handy for testing

Unicode

Python Docs Unicode HowTo:
`http://docs.python.org/howto/unicode.html`
Reading Unicode from a file is therefore simple:

```
import codecs
f = codecs.open('unicode.rst', encoding='utf-8')
for line in f:
    print repr(line)
```