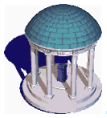


The University of North Carolina at Chapel Hill

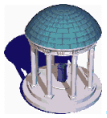
COMP 144 Programming Language Concepts

Spring 2002

Lecture 8: Python's Files, Modules, Classes and Exceptions

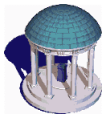
Felix Hernandez-Campos

Jan 28



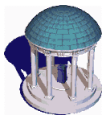
Files

- Creating file object
  - Syntax: file_object = open(filename, mode)
    - `Input = open("d:\inventory.dat", "r")`
    - `Output = open("d:\report.dat", "w")`
- Manual close
  - Syntax: close(file_object)
    - `close(input)`
- Reading an entire file
  - Syntax: string = file_object.read()
    - `content = input.read()`
  - Syntax: list_of_strings = file_object.readlines()
    - `lines = input.readline()`



Files

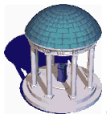
- Reading one line at time
  - Syntax: list_of_strings = file_object.readline()
    - » `line = input.readline()`
- Writing a string
  - Syntax: file_object.write(string)
    - » `output.write("Price is %(total)d" % vars())`
- Writing a list of strings
  - Syntax: file_object.writelines(list_of_string)
    - » `output.write(price_list)`
- This is very simple!
  - Compare it with `java.io`



Modules

- Long programs should be divided into different files
  - It makes them easier to maintain
- Example: mandelbrot.py

```
# Mandelbrot module
def inMandelbrotSet(point):
  """
  True iff point is in the Mandelbrot Set
  """
  X, t = 0 + 0j, 0
  while (t < 30):
    if abs(X) >= 2: return 0
    X, t = X ** 2 + point, t + 1
  return 1
```



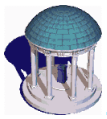
Using Modules

- Importing a module
  - Syntax: import module_name

```
import mandelbrot

p = 1+0.5j

if mandelbrot.inMandelbrotSet(p):
    print "%f+%fj is in the set" % (p.real, p.imag)
else:
    print "%f+%fj is NOT in the set" % (p.real, p.imag)
```

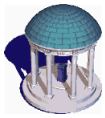



Using Modules

- Importing functions within a modules
  - Syntax: from module_name import function_name

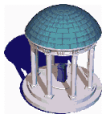
```
from mandelbrot import inMandelbrotSet
p = 1+0.5j
if inMandelbrotSet(p):
    print "%f+%fj is in the set" % (p.real, p.imag)
else:
    print "%f+%fj is NOT in the set" % (p.real, p.imag)
```

- Importing all the functions within a module
  - Syntax: from module_name import *



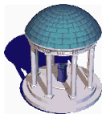
Standard Modules

- Python has a very comprehensive set of standard modules (a.k.a. libraries)
  - See Python library reference
    - » http://www.python.org/doc/current/lib/lib.html



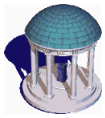
`string`

- Reference
  - http://www.python.org/doc/current/lib/module-string.html
- Some very useful functions
  - find(s, sub[, start[,end]])
  - split(s[, sep[, maxsplit]])
  - strip(s)
  - replace(str, old, new[, maxsplit])



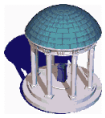
Regular expressions

- The `re` module provides Perl-style REs
- References:
  - HOWTO
    - » http://py-howto.sourceforge.net/regex/regex.html
  - Module reference
    - » http://www.python.org/doc/current/lib/module-re.html



Regular Expressions

- A regular expression (RE) is:
  - A single character
  - The empty string, ε
  - The concatenation of two regular expressions
    - » *Notation:* RE_1 RE_2 (*i.e.* RE_1 followed by RE_2)
  - The union of two regular expressions
    - » *Notation:* $RE_1 \mid RE_2$
  - The closure of a regular expression
    - » *Notation:* RE*
    - » * is known as the *Kleene star*
    - » * represents the concatenation of 0 or more strings

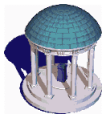


Defining Regular Expression

- Regular expression are created using compile(re)
  - *E.g.*

```
import re
regex_object = re.compile('key')
```

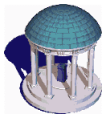
- Basic syntax
  - Concatenation: sequence
  - Union: |
  - Closure: *



Regular Expression Matching

- Matching at the beginning of strings
  - Syntax: math_object = re.match(string)
  - *E.g.*

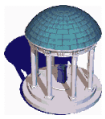
```
>>> import re
>>> regex = re.compile('key')
>>> match = regex.match("I have a key")
>>> print match
None
>>> match = regex.match("keys open doors")
>>> print match
<_sre.SRE_Match object at 0x009B6068>
```



Regular Expression Matching

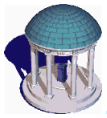
- Matching within strings
  - Syntax: math_object = re.search(string)
  - *E.g.*

```
>>> regex = re.compile('11*')
>>> match = regex.match("I have $111 dollars")
>>> print match
None
>>> match = regex.search("I have $111 dollars")
>>> print match
<_sre.SRE_Match object at 0x00A036F0>
>>> match.group()
'111'
```

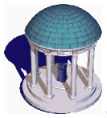
Matching Object Methods

- Return matched string
  - Syntax: `string = match_object.group()`
- Return starting position of the match
  - Syntax: `m = match_object.start()`
- Return ending position of the match
  - Syntax: `n = match_object.end()`
- Return a tuple with start and end positions
  - Syntax: `m, n = match_object.span()`



Extended Syntax

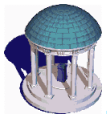
- Ranges
  - Any character within `[]` (*e.g.* `[abc]`)
  - Any character not within `[]` (*e.g.* `[^abc]`)
- Predefined ranges
  - `\d` Matches any decimal digit; this is equivalent to the set `[0-9]`.
  - `\D` Matches any non-digit character; equivalent to the set `[^0-9]`.
  - `\s` Matches any whitespace character; this is equivalent to the set `[\t\n\r\f\v]`.
  - `\S` Matches any non-whitespace character; this is equivalent to the set [^ \t\n\r\f\v].
- One or more closure + (*e.g.* `a+`)



Grouping

- Groups are defined using parentheses
  - *E.g.*

```
>>> regex = re.compile('(ab*)(c+)(def)')
>>> match = regex.match("abbbdefhhggg")
>>> print match
None
>>> match = regex.match("abbbcdefhhggg")
>>> print match
<_sre.SRE_Match object at 0x00A35A10>
>>> match.groups()
('abbb', 'c', 'def')
```

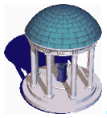


Classes

- Defined using `class` and indentation
  - E.g.

```
class MyClass:
        "A simple example class"
        i = 12345
        def f(self):
            return 'hello world'
```

- *Methods* are functions defined within the class declaration or using the *dot* notation
- *Attributes* are variables defined within the the class declaration or using the *dot* notation



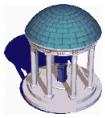
Class Constructor

- `__init__` method
  - E.g.

```
class MyClass:
    def __init__(self):
                self.data = []
```

- Creation of instances is straightforward
  - E.g.

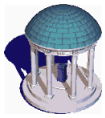
```
x = MyClass()
```



Class Examples

- Example

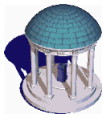
```
>>> class Complex:
...     def __init__(self, realpart, imagpart):
...         self.r = realpart
...         self.i = imagpart
...
>>> x = Complex(3.0, -4.5)
>>> x.r, x.i
(3.0, -4.5)
```

Exceptions

- Try/except/raise

```
>>> while 1:
...     try:
...         x = int(raw_input("Please enter a number: "))
...         break
...     except ValueError:
...         print "Oops! That was not valid.  Try again"
...
```



Reading Assignment

- Guido van Rossum and Fred L. Drake, Jr. (ed.), *Python tutorial*, PythonLabs, 2001.
  - Read chapters 6 to 9
  - http://www.python.org/doc/current/tut/tut.html
  - Write some simple programs
- Glance at the RE HOWTO and reference
  - You will need REs in the next assignment
  - HOWTO
    - » http://py-howto.sourceforge.net/regex/regex.html
  - Module reference
    - » http://www.python.org/doc/current/lib/module-re.html