

Introduction to Python

Part 2

COMPSCI 260
Fall 2019

Part 2 Topics:

- Regular expressions (and the *re* package)
- Python code reuse, modules, and *import*
- File I/O
- `range()` function
- Multidimensional arrays
- Some useful Python modules
- Quick note on debugging
- Other resources

Regular Expressions (Regex)

- Useful for finding matches to patterned strings.
- <http://docs.python.org/3.7/library/re.html>

Common Regex Symbols

<code>\</code>	escapes special characters.
<code>.</code>	matches any character
<code>^</code>	matches start of the string (or line if MULTILINE)
<code>\$</code>	matches end of the string (or line if MULTILINE)
<code>[5b-d]</code>	matches any chars '5', 'b', 'c' or 'd'
<code>[^a-c6]</code>	matches any char except 'a', 'b', 'c' or '6'
<code>R S</code>	matches either regex R or regex S.
<code>()</code>	Creates a capture group, and indicates precedence.

<code>*</code>	0 or more	(append ? for non-greedy)
<code>+</code>	1 or more	"
<code>?</code>	0 or 1	"
<code>{m}</code>	exactly 'm'	
<code>{m,n}</code>	from m to n.	'm' defaults to 0, 'n' to infinity
<code>{m,n}?</code>	from m to n,	as few as possible

Regular Expressions in Python

- Make sure you import the module "re".

Method	Purpose
sub()	Search and replace the matched substring with a new string
search()	Scan through a string, looking for any location where this RE matches.
match()	Determine if the RE matches at the beginning of the string.
findall()	Find all substrings where the RE matches, and returns them as a list.
finditer()	Find all substrings where the RE matches, and returns them as an iterator.

Regular Expressions in Python

- MatchObject methods (used with *match* or *search*).

Method/Attribute	Purpose
<code>group()</code>	Return the string matched by the RE
<code>start()</code>	Return the starting position of the match
<code>end()</code>	Return the ending position of the match
<code>span()</code>	Return a tuple containing the (start, end) positions of the match

Examples of Regular Expressions

```
import re
```

sub() — Search and replace the matched substring with a new string

```
input = 'Today is not a good day'
result = re.sub('not', '', input)          # Delete pattern 'not'
```

search() — Scan through a string, looking for any location where the RE matches

```
if re.search('day', 'Sunday'):
    print('Found a match')
```

findall() — Find all substrings where the RE matches, and returns them as a list

finditer() — Find all substrings where the RE matches, and returns them as an iterator

```
exampleString = 'Jessica is 15 years old, and Daniel is 27 years old.
Edward is 97 years old, and his grandfather, Oscar, is 102.'
ages = re.findall(r'\d{1,3}', exampleString)
names = re.finditer(r'[A-Z][a-z]*', exampleString)
print(ages)
for match in names:
    print('Match starts at', match.start())
```

Practice Time:

- Use 10 minutes to play with Part5.py from Tutorial Part 2 in PyCharm.

Import

- You can make use of any Python module or your own code libraries with import.
- `import module_name`
 - `import re` # usually at top of python script
 - You'll have to invoke using `re.findall()`
- `from module_name import function_name`
 - `from re import *` # * means import all functions
 - Can invoke module functions like `findall()` directly

Some Useful Python Modules

- *sys* – useful when using Python scripts on the command line
 - Example:

```
import sys
commandArg1 = sys.argv[1]
commandArg2 = sys.argv[2]
```
- *os* – list the contents of directories, make system calls (e.g. `wc`, `ls`, `mv`, `rm`, etc.)
 - Example:

```
import os
dirContents = os.listdir('./MyDirectory')
for i in range(0, len(dirContents)):
    if i==0:
        os.system("rm ./MyDirectory/%s" % dirContents[i])
```
- *string* – more string manipulations
 - Example:

```
import string
safetyDance = string.join(["You", "can", "dance", "if", "you", "want", "to"], " ")
```
- *math* – simple math functions
 - Example:
 - `math.log`, `math.exp`, `math.sqrt`, etc.

File I/O

- <http://docs.python.org/3.7/tutorial/inputoutput.html>
- Open a file for reading: `f = open('path','r')`
- Open a file for writing: `f = open('path','w')`
- Open a file for appending: `f = open('path', 'a')`

File I/O (cont'd)

- Example 1:

- `f = open("output.txt", "r")` # grab file handle
- `all_lines = f.readlines()` # read all lines into list
- `f.close()`

- `f = open("output.txt", "r")`
- `for fline in f:` # iterate through file with for loop
 - `flist = fline.strip().split()`
 - `print flist`
- `f.close()`

- Example 2: (Can also write using `print()` function)

- `f = open("output.txt", "w")`
- `f.write("We are no strangers to love\n")`
- `f.write("You know the rules ")`
- `f.write("and so do I\n")`
- `f.close()`

Range() function

- An important built-in method: `range(n)`.
- Very useful for iterating the "for" statement.
- `range(50) -> [0,...,49]` # up to n -1
- `range(1,51) -> [1,...,50]`
- `range(1,51, 2) -> [1, 3, ..., 49]`
- `range(10,0, -1) -> [10,...,1]`

Practice Time:

- Use 10 minutes to practice with Part6.py.

Multidimensional Arrays

- In some assignments you will need to create 1D, 2D, 3D arrays with some initial values.
- To create 1D array, use list comprehension:
 - `ex_1D = [str(x) for x in range(5)]`
- To create 2D (or XD) array, use nested list comprehensions:
 - `ex_2D = [[str(x) for x in range(5)] for y in range(10)]`
- Use the pprint module to print 2D,3D arrays easily:
 - `from pprint import *`
`pprint(ex_2D)`
 - alternatively, use `str()` with `print`
`print(str(ex_2D))`

Practice Time:

- Use 15 minutes to practice with Part7.py.

Python in PyCharm/Debugging

- Questions about creating new PyDev projects? Other Python/PyCharm questions?
- Debugging:
 - Print functions are your friend!
 - Breakpoints
 - Interactive Display
- Printing Long Strings

Other Resources

- Software Carpentry – a great general resource on scientific programming (software-carpentry.org)
- learnpython.org – includes a lot of what we've discussed and more advanced topics (generators, data serialization w/ pickle and JSON, decorators, etc.)
- Python 3.7 Documentation/Tutorial (<http://docs.python.org/3.7/tutorial/>) - a bit dense but the go-to reference for understanding what's going on under the hood