

## Python - 2

- **Walking a directory structure**
  - Using the `os` module
- **Using regular expressions**
  - Using the `re` module
  - Forming regular expressions
  - Doing replacement for regular expressions
  - Forming patterns from regular expressions

## How to walk a directory?

```
import os
for root, dirs, files in
    os.walk('/Users/ryder/Rutgers/classes/314/314f07/python/Project/testHandin'):
    print " root= ", root, "\n diry= ",
    for d in dirs:
        print d,
    print "\n files= ",
    for f in files:
        print f.
    print "\n"
```

```
1 1 barbaras-computer!uploaded> python os-walk.py
root=
/Users/ryder/Rutgers/classes/314/314f07/python/Project/testHandin
diry= Prolog Python Scheme
files=
```

```

root=
  /Users/ryder/Rutgers/classes/314/314f07/python/Project/testHandin/
  Prolog
diry= hari ryder
files=
root=
  /Users/ryder/Rutgers/classes/314/314f07/python/Project/testHandin/
  Prolog/hari
diry=
files= proj3 proj4 proj5
root=
  /Users/ryder/Rutgers/classes/314/314f07/python/Project/testHandin/
  Prolog/ryder
diry=
files= proj1 proj2
root=
  /Users/ryder/Rutgers/classes/314/314f07/python/Project/testHandin/
  Python
diry= ryder smith
files=
root=
  /Users/ryder/Rutgers/classes/314/314f07/python/Project/testHandin/
  Python/ryder
diry=

```

```

root=
  /Users/ryder/Rutgers/classes/314/314f07/python/Project/testHandin/
  /Python/smith
diry=
files= proj6
root=
  /Users/ryder/Rutgers/classes/314/314f07/python/Project/testHandin/
  /Scheme
diry= detlef ryder
files=
root=
  /Users/ryder/Rutgers/classes/314/314f07/python/Project/testHandin/
  /Scheme/detlef
diry=
files= proj9
root=
  /Users/ryder/Rutgers/classes/314/314f07/python/Project/testHandin/
  /Scheme/ryder
diry=
files= proj7 proj8

```

```

6 1 barbaras-computer!testHandin> ls -R
total 0
drwxrwx---  4 ryder  admin  136 Nov 18 20:47 Prolog
drwxrwx---  4 ryder  admin  136 Nov 18 20:48 Python
drwxrwx---  4 ryder  admin  136 Nov 18 20:49 Scheme
./Prolog:
total 0
drwxrwx---  5 ryder  admin  170 Nov 18 20:48 hari
drwxrwx---  4 ryder  admin  136 Nov 18 20:48 ryder
./Prolog/hari:
total 0
-rw-rw----  1 ryder  admin  0 Nov 18 20:48 proj3
-rw-rw----  1 ryder  admin  0 Nov 18 20:48 proj4
-rw-rw----  1 ryder  admin  0 Nov 18 20:48 proj5
./Prolog/ryder:
total 0
-rw-rw----  1 ryder  admin  0 Nov 18 20:47 proj1
-rw-rw----  1 ryder  admin  0 Nov 18 20:48 proj2
./Python:
total 0
drwxrwx---  3 ryder  admin  102 Nov 18 20:48 ryder
drwxrwx---  3 ryder  admin  102 Nov 18 20:48 smith

```

Actual  
directory  
structure

Python-2, CS314 Fall 2007 © BGRyder

5

```

./Python/ryder:
total 0
-rw-rw----  1 ryder  admin  0 Nov 18 20:48 proj5
./Python/smith:
total 0
-rw-rw----  1 ryder  admin  0 Nov 18 20:48 proj6
./Scheme:
total 0
drwxrwx---  3 ryder  admin  102 Nov 18 20:49 detlef
drwxrwx---  4 ryder  admin  136 Nov 18 20:49 ryder
./Scheme/detlef:
total 0
-rw-rw----  1 ryder  admin  0 Nov 18 20:49 proj9
./Scheme/ryder:
total 0
-rw-rw----  1 ryder  admin  0 Nov 18 20:49 proj7
-rw-rw----  1 ryder  admin  0 Nov 18 20:49 proj8

```

Python-2, CS314 Fall 2007 © BGRyder

6

## Regular Expressions in Python

- **Standard pattern constructors**
  - . Matches any char
  - ^ Matches beginning of string
  - \$ Matches end of string (just before \n)
  - \* 0 or more repetitions of preceding RE
  - + 1 or more repetitions of preceding RE
  - ? Matches 0 or 1 of preceding RE
  - [ ] delimits a set of characters as possible matches
    - [a-zA-Z] matches any alphabetic symbol
    - [^a-z] matches any non-alphabetic symbol
  - | A|B where A,B are REs, means A or B
    - matches tried from left to right in or expression
  - \s matches any whitespace character
  - \d matches [0-9]

Python-2, CS314 Fall 2007 © BGRyder

7

### Ex1-simple substitution, `addr.py`

Cf Dive into Python

*Suppose you have a list of addresses and you want to convert the street address to use abbreviations for 'Avenue' (Ave.), 'Road' (Rd.), and Street (St.); how to do this with regular expressions in Python*

```
import re #RE module
f1=open("/Users/ryder/Rutgers/classes/314/314f07/python/addr")
f2=open("/Users/ryder/Rutgers/classes/314/314f07/python/addr")
for line in f1: # $ in pattern means end of string
    x = re.sub(r'ROAD$', 'Rd.',line)
    print line, x
for line in f2:
    x = re.sub('\bROAD$', 'Rd.', line) #\b requires word
    to start at this point
    y = re.sub(r'\bROAD$', 'Rd.', line) #r is raw string
    mode, to prevent any escapes due to backslashes in
    string, '\t' is a tab char; r'\t' is a backslash
    followed by a t in the string.
    z = re.sub(r'\bROAD\b', 'Rd.', line) #new pattern
    does not require ROAD to be at end of string
    print line,x,y,z
```

Python-2, CS314 Fall 2007 © BGRyder

8

**Input**

```
100 BROAD ROAD
100 FALLS ROAD, Apt 10
100 FALLS ROAD, 101 FALLS ROAD
```

```
addrs.out
```

**Output (exerpts)**

```
100 BROAD ROAD
100 BROAD Rd. #pattern worked and didn't change ROAD in
    BROAD
100 FALLS ROAD, Apt 10
100 FALLS ROAD, Apt 10 #pattern failed
100 FALLS ROAD, 101 FALLS ROAD
100 FALLS ROAD, 101 FALLS Rd. #pattern worked on last 'ROAD'
...
100 FALLS ROAD, 101 FALLS ROAD
100 FALLS ROAD, 101 FALLS Rd.
100 FALLS ROAD, 101 FALLS Rd.
100 FALLS Rd., 101 FALLS Rd.
```

**Ex2- forming and using patterns and matches**

```
import re
s = 'cabcd'
t = 'aabbaabb'
print s,t #cabcd aabbaabb
patt = re.compile(r'^a+') #creates pattern
x = patt.search(s) #searches for pattern in string s
if x != None: #if no match, returns None
    print x.group() #else print what matched pattern
print (patt.search(t)).group() #aa
patt2=re.compile(r'a+')
print (patt2.search(s)).group() #a
print (patt2.search(t)).group() #aa

for matchObj in patt2.finditer(t):
    print matchObj.end() #prints 2 and 6
```

## Ex2- forming and using patterns and matches

```
patt3=re.compile(r'(?P<x1>a+) | (?P<x2>b+) ')
print (patt3.search(s)).group("x1") #a
print (patt3.search(s)).group("x2") #None

for matchObj in patt3.finditer(t):
    print matchObj.end() #prints 2,4,6,8
```