PYTHON LAB BOOK

Python For Programmers

UCSC Extension Online

Lab 19 re Module

Topics

- re - Regular Expressions (Optional)
- Search and replace (Optional)
- Named groups (Optional)

v.4.0

lab18_1.py

```
#!/usr/bin/env python
"""lab18_1.py
Make an UpIt(str) function that returns the input string, but
with all caps.  Your UpIt function will be different from
str.upper() in that, if any of the characters in the input string
are already uppercase, it raises an exception.  Invent your own
exception and put it in a reasonable spot in the exception
hierarchy.
"""
import sys

class CaseError(ValueError):
    pass

def UpIt(s):
    if not s.islower():
        raise CaseError
    return s.upper()

if __name__ == '__main__':
    try:
        print UpIt(sys.argv[1])
    except CaseError:
        print sys.argv[1], 'has at least one upper case letter.'
    except IndexError:
        print 'Usage:', sys.argv[0], 'string_of_lower_case_letters'

"""
$ lab18_1.py HI
HI has at least one upper case letter.
$ lab18_1.py hi
HI
$ lab18_1.py Hi
Hi has at least one upper case letter.
$
"""
```

lab18_2.py

```
#!/usr/bin/env python
"""lab18_2.py -- collects 2 sides of a right triangle and prints
the hypotenuse."""

import math

def GetXY(prompt):
    """Returns a tuple of 2 floats, or None if the user just hits the
    enter key or enters q."""

    while True:
        try:
            incoming = raw_input(prompt)
        except (KeyboardInterrupt, EOFError):
            return None
        if incoming == '' or incoming[0].lower() == 'q':
            return None
        try:
            x, y = [float(x) for x in incoming.split(',')]
        except (TypeError, NameError, ValueError, SyntaxError):
            print "Two numbers are required."
        else:
            return x,y

def Hypot(x, y):
    """Returns sqrt(x**2 + y**2)"""
    return math.sqrt(x * x + y * y)

def main():
    while True:
        xy = GetXY("Give me 2 sides of a right triangle: (x, y) ")
        if not xy:
            print
            break
        answer = Hypot(*xy)
        if answer < 0:
            continue
        print 'Hypotenuse is %.2f' % (answer)

if __name__ == '__main__':
    main()

"""
$ lab18_2.py
```

```
Give me 2 sides of a right triangle: (x, y) 4, 5
Hypotenuse is 6.40
Give me 2 sides of a right triangle: (x, y) a
Two numbers are required.
Give me 2 sides of a right triangle: (x, y) 1
Two numbers are required.
Give me 2 sides of a right triangle: (x, y) 1,a
Two numbers are required.
Give me 2 sides of a right triangle: (x, y) 1,2,3
Two numbers are required.
Give me 2 sides of a right triangle: (x, y) 2,,3
Two numbers are required.
Give me 2 sides of a right triangle: (x, y) 1,2
Hypoteneuse is 2.24
Give me 2 sides of a right triangle: (x, y)
$ lab18_2.py
Give me 2 sides of a right triangle: (x, y)
$ lab18_2.py
Give me 2 sides of a right triangle: (x, y) Q
$"""
```

Answers for Lab 18_3

1.
```
>>> a_list = ['loop']
>>> a_list += a_list
>>> a_list
['loop', 'loop']
>>> a_list.append(a_list)
>>> a_list
['loop', 'loop', [...]]
```

So `a_list` has itself in it, that `a_list` inside has `a_list` inside, has `a_list` inside, ... Python uses **...** to indicate this infinite pattern.

I can't explain the difference between `append` and `+=`.

2.
```
>>> bottles = 100
>>> def HowMany():
...     print bottles
...
>>> HowMany()
100
>>> def HowMany():
...     bottles -= 1
...     print bottles
...
>>> HowMany()
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "<stdin>", line 2, in HowMany
UnboundLocalError: local variable 'bottles' referenced before
assignment
```

While interpreting, Python sees the `bottles -= 1` as an assignment and decides that `bottles` is a local name and sets it up that way. But at run-time, the interpreter tries to access `bottles` and doesn't find the local variable.

How can you fix this so that the `HowMany`'s `bottles` is really the global `bottles`?

```
>>> def HowMany():
...     global bottles
...     bottles -= 1
...     print bottles
    ...
>>> HowMany()
99
```

3.
```
>>> def Snake(rattle=[]):
...     rattle += ["hiss"]
...     print rattle
...
>>> Snake([100])
[100, 'hiss']
```

No surprises there but:

```
>>> Snake()
['hiss']
```

As expected.

```
>>> Snake()
['hiss', 'hiss']
>>> Snake()
['hiss', 'hiss', 'hiss']
```

Woah, since it used the mutable default in the last call, and then it altered it, it altered that default!

4.
```
>>> class X:
...     pass
...
>>> class Y:
...     pass
...
>>> X.a = 1
>>> X.b = 2
>>> X.c = 3
>>> Y.a = X.a + X.b + X.c
>>> for X.i in range(Y.a):
...     print X.i
...
0
1
2
3
4
5
>>> dir (X)
['__doc__', '__module__', 'a', 'b', 'c', 'i']
```

Handling Regular Expressions in Python (Labs 19-20 are optional)

```
>>> import re

>>> mo = re.search("needle", "Is there a needle in this haystack?")
>>> print mo                            mo == None if there's no match
<_sre.SRE_Match object at 0xb7ea5f00>

>>> mo.span()
(11, 17)
>>> mo.start()
11
>>> mo.end()
17

>>> re.findall("needle", "Is there a needle in this haystack of needles?")
['needle', 'needle']

>>> re.sub("needle", "thread", "Is there a needle in this haystack of needles?")
'Is there a thread in this haystack of threads?'

>>> re.subn("needle", "thread", "Is there a needle in this haystack of needles?")
('Is there a thread in this haystack of threads?', 2)

>>> c = re.compile("needle")

>>> c.findall("Is there a needle in this haystack of needles?")
['needle', 'needle']
```

Raw Strings!

```
"(\d+\.\d*|\d*\.\d+|\d+)"
```

is a regular expression that matches numbers.

All those '\' need escaping:

```
"(\\d+\\.\\d*|\\d*\\.\\d+|\\d+)" TOO HARD!
```

```
r"(\d+\.\d*|\d*\.\d+|\d+)" is the "raw string" notation.
```

```
>>> c = re.compile(r"(\d+\.\d*|\d*\.\d+|\d+)")
>>> c.findall("3.2, a .4 and a 5. x.x .y z. . .")
['3.2', '.4', '5.']
```

However, your pattern might contain special re characters that you are looking for as literals.

```
>>> re.findall("cat?", "Where's the cat?  Is the cat in the car?")
['cat', 'cat', 'ca']
```

It didn't find that question mark but thought we were looking for 0 or 1 `t`s, so it just matched the *cat*. In this case, you need to put the escape character before the '?'. `re.escape` does it for you:

```
>>> re.findall(re.escape("cat?"), "Where's the cat?  Is the cat in the car?")
['cat?']
```

```
>>> import re

Setting up some capturing groups:

>>> p = re.compile('(a(b)c)d')  >>> p = re.compile(r'\b((\w+) ?(\w?\.?)) Jones')
>>> m = p.search('abcd')        >>> m = p.search("John J. Jones")
>>> m.group()                   >>> m.group()
'abcd'                          'John J. Jones'
>>> m.groups()                  >>> m.groups()
('abc', 'b')                    ('John J.', 'John', 'J.')
>>> m.group(1)                  >>> m.group(2)
'abc'                           'John'
                                >>> p.findall("John J. Jones, Mary Jones")
                                [('John J.', 'John', 'J.'), ('Mary', 'Mary', '')]
Names Groups:

>>> r'\b((?P<first>\w+) ?(?P<middle>\w?\.?)) Jones')
>>> m = p.search("John J. Jones")
>>> m.groups(2)
'John'
>>> m.group('first')
'John'
>>> m.group('middle')
'J.'

Works nicely with sub:

>>> p = re.compile('section{(?P<which>[^}]*)}')
>>> p.sub(r'subsection{\g<which>}', 'section{First}')
'subsection{First}'
>>> p.sub(r'subsection{\1}', 'section{First}')
'subsection{First}'
>>> p.sub(r'subsection{\g<1>}', 'section{First}')
'subsection{First}'

With p.sub or re.sub, instead of a replacement string, you can give a
function name!
```

re_swap.py

```
#!/usr/bin/env python
"""Swapping cats and dogs again, but this time doing a better job with
regular expressions.  The old, faulty solution is:

def DoSwap(text, apple, orange):
    """Swap apple for orange and orange for apple in the text.

    Return the swapped text.
    """

    dummy = 'wxyz'
    while True:
        if text.find(dummy) == -1:
            break
        dummy *= 2
    text = text.replace(apple, dummy)
    text = text.replace(orange, apple)
    text = text.replace(dummy, orange)
    return text

"""

import re

def DoSwap(text, apple, orange):
    """Swaps all occurances of apple for orange, and orange
    for apple in the text."""
    # Using VERBOSE and named groups for readability
    compiled_re = re.compile(r"""
    \b               # matches a word boundary
    (?P<found>%s|%s) # matches apple or orange and puts the match
                     # in a group named "found" if the whole thing matches
    (?P<rest>s?)\b   # matches a word boundary or 's' and a word boundary
                     # and puts the 's', or not, into a group named "rest"
    """ % (re.escape(apple), re.escape(orange)), re.IGNORECASE | re.VERBOSE)

    def MatchCase(answer, like_string):
        if like_string.isupper():
            return answer.upper()
        if like_string.islower():
            return answer.lower()
        if like_string.istitle():
            return answer.title()
        return answer

    def SwapMatch(match_object):
        found = match_object.group('found')
        if found.lower() == apple.lower():
            x = MatchCase(orange, found) + match_object.group('rest')
            return x
        x = MatchCase(apple, found) + match_object.group('rest')
        return x

    fixed = compiled_re.sub(SwapMatch, text)
    return fixed

def main():
    print DoSwap("DOGS: 14dogs are a lot of Dogs.", 'dog', 'elephant')
    print DoSwap("3 Categories of CATS.", 'cat','dog')

if __name__ == '__main__':
    main()
"""
$ re_swap_caseless.py
ELEPHANTS: 14dogs are a lot of Elephants.
3 Categories of DOGS.
$
"""
```

quiz.py

```
#!/usr/bin/env python
"""Quiz answer."""

import sys
sys.path.insert(0, '..')
import pet_store.pet_def

class Rock(pet_store.pet_def.Pet):
    no_rocks = 0

    def __init__(self, name, weight):
        super(Rock, self).__init__(name)
        pet_store.pet_def.Pet.__init__(self, name)
        Rock.no_rocks += 1
        self.__weight = weight

def main():
    rocky = Rock("Rocky", "3 oz")
    gravel = Rock("Gravel", "1/2 oz")
    diamond = Rock("Diamond", ".03 oz")
    print "Made %d pet rocks." % Rock.no_rocks
    print rocky.__weight

if __name__ == '__main__':
    main()
"""
$ quiz.py
Made 3 pet rocks.
Traceback (most recent call last):
  File "./quiz.py", line 23, in <module>
    main()
  File "./quiz.py", line 20, in main
    print rocky.__weight
AttributeError: 'Rock' object has no attribute '__weight'
$ tree.py mall
|-- /mall
|   |-- /basket
|   |   |-- quiz.py
|   |-- /pet_store
|   |   |-- pet_def.py
|   |   |-- __init__.py
$ """
```

Lab 19_re_Module (Optional)

Check out `http://regexlib.com` to find some helpful regular expressions and other tools.

1. Find `labs/lab_11_Packages/numbers.txt` again. Use this regular expression to add up all the numbers:

   ```
   "\d+\.\d*|\d*\.\d+|\d+"
   ```

2. Say you have a dictionary of search and replace words:

   ```
   replace_d = {
   'orchestra leader':'singer',
   'china':'mexico',
   'zen':'mariachi',
   'master':'teacher',
   'sword':'baton',
   'through':'around'}
   ```

   Find file `labs/lab_19_re_Module/zen.story` and change all those words. Copy the dictionary from `zen.dictionary` in the same directory.

3. A big optional project! A few potentially useful, but imperfect, regular expressions:

   For email addresses:

   ```
   r"[\s,:;(<\['\"]+([^@\s,;:\"]+)@([^@\-\s,:;>\)'\"]+\.[a-z,A-Z]{2,3})[\s,;:\"'>\)]"
   ```

   For finding URL's:

   ```
   r"http://[^\s\"<>()']*\b"
   ```

   Copy and paste these from `labs/lab_19_re_Module/lab.data` Or, if you know better ones, use them, and please tell me.

   With the urllib module, you can:

   ```
   stream = urllib.urlopen("http://www.pythontrainer.com")
   ```

   and then handle `stream` as if it was an open file.

   If there is an error opening the page, the exception raised is `httplib.InvalidURL` so you need to `import httplib` if you want to catch such an exception. If the page asks for a password, it'll hang. So, you might want to catch that by:

   ```
   import labs/lab_12_Function_Fancies.time_out_decorator as time_out
   ```

Then decorate over any function that you expect will hang:

```
@time_out.TimeOut(3) # This make the following function
                     # timeout in 3 seconds.
def CollectPage(url):
    whatever that should do

try:
    CollectPage(some_url)
except RuntimeError:
    whatever that should do
```

Write an evil program that takes a starting url ('www.ngc.com' behaves pretty well), finds all the email addresses on that page, and all the links to urls on that page, and then reads all the links, and their links, etc., collecting email addresses.