python Workshop

2nd Session

BY-

CHANDAN MITTAL

STUDENT, CSE DISCIPLINE

IIITDM JABALPUR

filter(function, sequence)

- **Returns sequence consisting of those items from the sequence for which function(item) is true**

```
>>> def f(x): return x%3 == 0 or x%5 == 0

>>> filter(f, range(2, 25))

[3, 5, 6, 9, 10, 12, 15, 18, 20, 21, 24]
```

map(function, sequence)

- **map(function, sequence) calls function(item) for each of the sequence items and returns a list of the returned values.**

```
>>> def cube(x): return x**3

>>> map(cube, range(1, 11))

[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]

>>>

>>> seq  = range(8)

>>> def add(x, y): return x+y

>>> map(add, seq, seq)

[0, 2, 4, 6, 8, 10, 12, 14]
```

reduce(function, sequence)

- reduce(function, sequence) returns a single value constructed by calling the binary function *function* on the first 2 items of the sequence, then on the result and the next item, and so on.

```
>>> def add(x, y): return x+y
>>> reduce(add, range(1, 11))
55
>>>def sum(seq):
        def add(x, y): return x+y
        return reduce(add, seq, 0)
>>> sum(range(1, 11))
55
```

Lambda functions

```
>>> def f(x): return x**2
>>> print f(4)
16
>>> #anonymous fn
>>> g = lambda x: x**2
>>> print g(4)
16
```

```
>>> #prog to print primes from 2-50
>>> nums = range(2, 50)
>>> for i in range(2, 8):
...         nums = filter(lambda x: x == i or x%i, nums)
...
>>> print nums
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

File Handling

```
>>> f = open(‘filename.txt’, ‘r’)
>>> f.readline()
>>> f.close()


>>> f = open(‘filename.txt’, ‘w’)
>>> f.write(‘hello world’)
>>> f.close() #don’t forget to close the file
```

File Handling (continued..)

```
>>>for line in open(“filename.txt”)
...         print line
>>> #here the file remains open for indeterminate amount of time


>>> with open(“filename.txt”) as f:
 ...        for line in f:
 ...                    print line
>>> #after the statement executed, file is always closed.
```

Errors and Exceptions

```
>>> 10 * (1/0)
ZeroDivisionError
>>> 2*x
NameError
>>> '2' + 2
TypeError
```

- Even if a statement is syntactically correct, it may cause an error when an attempt is made to execute it.
- Errors detected during execution are called **exception**.

```
>>> import sys
>>> try:
        x, y = map(int, raw_input().split())
        print x/y
        print 'Division successful'
 except IOError as e:
      print "I/O Error " + e.reason
 except ZeroDivisionError:
      print 'Divide by Zero Error'
 except:
      print 'Unexpected error:', sys.exc_info()[0]
      raise
```

Errors and Exceptions (continued..)

```
>>> try:
...         raise NameError(‘Hi there’) #intentionally raise an exception
 except NameError:
...         print ‘An exception occurred’
...         raise
...
An exception occurred
NameError: Hi There
```

Errors and Exceptions (continued..)

```
>>> def divide(x, y):
...     try:
...         result = x/y
...     except ZeroDivisionError:
...         print ‘Divide by zero error’
...     else:
...         print ‘result is ’, result
...     finally:
...         print “executing finally clause”
```

```
>>> divide(2, 1)
result is 2
executing finally clause
>>> divide(2, 0)
 Divide by zero error
 executing finally clause
>>> divide(“2”, “1”)
 executing finally clause
 TypeError
```

OOPS in python

```
>>> class MyClass:
        i = 12345
        def f(self):
            return 'hello world'
```

Attribute Reference

MyClass.i and MyClass.f are valid attribute references

Instantaniation

x = MyClass()

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

OOPS in python

```
>>> class Employee(object):
        empcount = 0
        def __init__(self, name, salaray):
                self.name = name  #self.name is variable of this object
                self.salary = salary    #self.salary is variable of this object
                Employee.empcount += 1 #
        def displayCount(self):
                print ‘Total Employee ’ + str(Employee.empcount)
        def displayEmployee(self):
                print ‘Name ’ + str(self.name) + ‘\nSalary’ + str(self.salary)

>>> if __name__ == ‘__main__’:
…       emp = Employee()
```

OOPS and py modules(continued..)

```python
# file one.py
def func():
    print("func() in one.py")

print("top-level in one.py")

if __name__ == "__main__":
    print("one.py is being run directly")
else:
    print("one.py is being imported into another module")
```

```python
# file two.py
import one

print("top-level in two.py")
one.func()

if __name__ == "__main__":
    print("two.py is being run directly")
else:
    print("two.py is being imported into another module")
```

OOPS and modules (continued..)

Invoke interpreter as

```
>>> python one.py


top-level in one.py
one.py is being run directly
```

Invoke interpreter as

```
>>> python two.py


top-level in one.py
one.py is being imported into another module
top-level in two.py
func() in one.py
two.py is being run directly
```

What’s new in python3?

Changed Syntax

Old: print "The answer is", 2*2

New: print("The answer is", 2*2)

Range() behave like xrange()

Map() and filter() return iterators