Introduction to Python - 2

Summer Projects & Workshop 2017

Ujjawal Soni

Department of Computer Science & Engineering
IIT Madras

June 2, 2017

Outline

Functions

- functions are constructs that allow you to structure your programs
- functions group a set of statements so that they can run more than once in a program
- general format of a python function:

```
def <function-name>(<parameter-list>):
    <statements>          # the function body
```

  - `def` keyword is used to define a function in python
  - `<parameter-list>` contain none or more parameters
  - parameters can be mandatory or optional
  - function body can contain an optional `return` statement
- python also supports function which can take arbitrary number of parameters

Functions

Examples

- ```
  def times(x,y):          # create and assign function
      return x*y           # body executed when function is called
  ```
- ```
  def times(x,y,z=1):      # function with optional parameter z
      return x*y*z
  ```

```
>>> print times(2,2)        # returns 2*2*1
4
>>> print times(2,2,4)      # returns 2*2*4
16
```

- alternatively you can use keyword parameters to make function calls

```
def func(w,x,y=1,z=1):
    return w*x*y+z
>>> print func(1,1,z=3)
4
>>> print func(1,1,z=3,y=4)
7
```

Higher-order Functions

`lambda` function

- lambda function is a way to create small anonymous functions, i.e. functions without a name
- lambda functions are mainly used in combination with the functions `filter()`, `map()` and `reduce()`
- general syntax of a `lambda` function is:

```
lambda argument_list: expression
```

- you can assign the function to a variable to give it a name
- Example:

```
>>> f = lambda x, y : x + y
>>> f(1,2)
3
```

Higher-order Functions

map function

- `map()` is a function with two arguments: `r = map(func, seq)`
  - first argument `func` is the name of a function and second a sequence (e.g. a list) `seq`
  - `map()` applies the function `func` to all the elements of the sequence `seq` and returns a new list with the elements changed by `func`
- Example:

```
>>> def inc(x): return x + 10
>>> map(inc, [1,2,3,4])
[11,12,13,14]
```

- `map()` can also be used with more than one list (lists need to be of same length):

```
>>> map((lambda x,y: x+3*y), [0,1,2], [1,2,3])
[3,7,11]
```

Higher-order Functions

`filter` function

- `filter(func,lst)` function filters out all the elements of `lst`, for which the function `func` returns `True`
- `func` is a function which returns a Boolean value, i.e. either `True` or `False`
- `func` will be applied to every element of the list `lst` and if `func` returns `True` for that element, it will be included in the resultant list
- Example,

```
>>> lst = [5,2,11,14,9,3,21,31,6]
>>> filter(lambda x: x < 10, lst)
[5,2,9,3,6]
```

Higher-order Functions

reduce function

- The function `reduce(func,seq)` continually applies the function `func()` to the sequence `seq` and returns a single value
- if `seq = [s1,s2,s3,...,sn]`, calling `reduce(func,seq)` works like this:
  - first the first two elements of `seq` will be applied to `func`, i.e. `func(s1,s2)`
  - the list on which `reduce()` works looks like this now: `[func(s1,s2),s3,...,sn]`
  - next `func` will be applied on the previous result and the third element of the list, i.e. `func(func(s1,s2),s3)`, and so on till only one element is left
- Example,

```
>>> reduce(lambda x,y: x+y, range(1,101))
5050
```

List Comprehension

- list comprehension is an elegant way to define and create list in python
- these lists have often the qualities of sets, but are not in all cases sets
- substitute for higher order functions
- Consider the following `for` loop example on a list:

```
>>> L = [1,2,3,4,5]
>>> for i in range(5): L[i] += 10
>>> L
[11,12,13,14,15]
```

- you can replace the loop with a single expression using list comprehension:

```
>>> L = [x+10 for x in L]
>>> L
[11,12,13,14,15]
```

- list comprehensions are more concise to write and run more faster than `for` loop statements

List Comprehension

- Consider the following example:

```
l = []
for x in range(1,11):
    for y in range(1,11):
        if 2*x == y:
            l.append((x,y))
print l
```

```
[(1,2),(2,4),(3,6),(4,8),(5,10)]
```

- using list comprehension, this can equivalently be written as:

```
>>> [(x,y) for x in range(1,11) for y in range(1,11) if 2*x == y]
[(1,2),(2,4),(3,6),(4,8),(5,10)]
```

- similar commands are supported for sets and dictionaries:

```
>>> d = {1:1, 2:2, 3:3, 4:4}
>>> {v:2*k for k,v in d.items()}
{1:2, 2:4, 3:6, 4:8}
```

File Management

- Reading from a file:

```
# create a file object
fin = open('words.txt', 'r')
for line in fin:
    print line.strip()
fin.close()
```

- Writing to a file:

```
# readlines() reads lines in the file to a list
fin = open('words1.txt', 'r').readlines()
fout = open('words2.txt', 'w')
i=1
for line in fin:
    print line.strip()
    # write the new string to output file
    fout.write(str(i) + ": " + line)
    i+=1
fout.close()
```

Exception Handling

- an exception is an error that happens during the execution of a program
- exception handling is a construct to deal with errors automatically
- common errors are 'division by zero', 'file open error', 'invalid literals for functions'
- the code which harbours the risk of an exception, is embedded in a `try` block, and the exception is handled in an `except` statement
- example,

```
while True:
    try:
        n = raw_input("Enter an integer: ")
        n = int(n)
        break
    except ValueError:
        print("Not a valid integer! try again ..")
print "Great! you successfully entered an integer!"
```

Exception Handling

Example

- multiple except clauses for handling different exceptions

```
try:
    f = open('words.txt')
    s = f.readline()
    i = int(s.strip())
except IOError:
    print "An I/O error occurred. Please check your file .."
except ValueError:
        print "Invalid input! Not an integer .."
except:
    print "An unexpected error occurred"
    raise
finally:
    # optional clause, executes regardless if an exception occurred
    if f:
        f.close()
print "Great! You file contains an integer %d!" %(i)
```

For Further Reading I


Mark Lutz.
Learning Python, Third Edition.
OReilly Media, 2008.


http://www.python-course.eu