

Introduction to Programming with Python - Day 3

`vipin@cbio.mskcc.org`

Rätsch Laboratory, Computational Biology Center
Memorial Sloan Kettering Cancer Center

11 November 2014

List Comprehensions

List comprehensions provide a concise way to create lists.

```
1 squares = []
2 for x in range(10):
3     squares.append(x**2)
4
5 print squares
6 #[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
1 squares = [x**2 for x in range(10)]
```

```
1 # filter the list to exclude less than or equal to 25
2 print [x for x in squares if x >= 25]
```

```
1 # call a method on each element
2 freshfruit = [' banana', ' loganberry ', 'passion fruit
3               ']
4 [weapon.strip() for weapon in freshfruit]
5 #['banana', 'loganberry', 'passion fruit']
```

del Statement

The **del** statement is to remove an item from a list given its index instead of its value.

```
1 xq = [-1, 1, 6, 3, 3, 12]
2 del xq[0]
3 xq
4 #[1, 6, 3, 3, 12]
5
6 del xq[2:4]
7 xq
8 #[1, 6, 12]
9
10 del xq[:]
11 xq
12 #[]
13
14 del xq
```

Tuple - Another Data Type

Tuples are immutable, and usually contain an heterogeneous sequence of elements.

```
1 t = 12, 54, 'hello!'
2 t[0]
3 12
4
5 # Tuples may be nested:
6 u = t, (1, 2, 3, 4, 5)
7 u
8 ((12, 54, 'hello!'), (1, 2, 3, 4, 5))
9
10 # Tuples are immutable:
11 t[0] = 88888
12
13 # but they can contain mutable objects:
14 v = ([1, 2, 3], [3, 2, 1])
15 v
16 ([1, 2, 3], [3, 2, 1])
```

Defining Functions

Little self-contained programs that perform a specific task.
Which you can incorporate into your own, larger programs.

'Calling' a function involves:
giving a function input, and it will return a value as output.

```
print 'Hello Python Class'
```

The keyword `def` introduces a function definition.

Defining Functions

function definition

```
1 def compute_factorial(n):  
2     """  
3     computes factorial of n  
4     """  
5  
6     ret = 1  
7     for i in xrange(n):  
8         ret=ret*(i+1)  
9  
10    return ret
```

Defining Functions

function definition

```
1 def compute_factorial(n):  
2     """  
3     computes factorial of n  
4     """  
5  
6     ret = 1  
7     for i in xrange(n):  
8         ret=ret*(i+1)  
9  
10    return ret
```

function call

```
1  
2 xq = compute_factorial(5)  
3 print xq
```

Modules

A module is a file containing Python definitions and statements.

Modules

A module is a file containing Python definitions and statements.
factorial.py:

```
1 def compute_factorial(n):  
2     """  
3     computes factorial of n  
4     """  
5  
6     ret = 1  
7     for i in xrange(n):  
8         ret=ret*(i+1)  
9  
10    return ret
```

Modules

A module is a file containing Python definitions and statements.
factorial.py:

```
1 def compute_factorial(n):  
2     """  
3     computes factorial of n  
4     """  
5  
6     ret = 1  
7     for i in xrange(n):  
8         ret=ret*(i+1)  
9  
10    return ret
```

Import this module:

```
1  
2 import factorial  
3 print factorial.compute_factorial(5)
```

Regular Expression Operations

The module `re` provides full support for regular expressions.

Regular Expression Operations

The module **re** provides full support for regular expressions.

Meta characters:

```
1 meta_char = '. ^ $ * + ? { } [ ] \ | ( )'
```

Regular Expression Operations

The module **re** provides full support for regular expressions.

Meta characters:

```
1 meta_char = '. ^ $ * + ? { } [ ] \ | ( )'
```

Predefined characters:

```
1 """
2 \d Matches any decimal digit [0-9].
3
4 \D Matches any non-digit character [^0-9].
5
6 \s Matches any whitespace character [ \t\n\r\f\v].
7
8 \S Matches any non-whitespace character [^ \t\n\r\f\v].
9
10 \w Matches any alphanumeric character [a-zA-Z0-9_].
11
12 \W Matches any non-alphanumeric character [^a-zA-Z0-9_].
13 """
```

Regular Expression Operations

The module `re` provides full support for regular expressions.

Meta characters:

```
1 meta_char = '. ^ $ * + ? { } [ ] \ | ( )'
```

Predefined characters:

```
1 """
2 \d Matches any decimal digit [0-9].
3
4 \D Matches any non-digit character [^0-9].
5
6 \s Matches any whitespace character [ \t\n\r\f\v].
7
8 \S Matches any non-whitespace character [^ \t\n\r\f\v].
9
10 \w Matches any alphanumeric character [a-zA-Z0-9_].
11
12 \W Matches any non-alphanumeric character [^a-zA-Z0-9_].
13 """
```

For further references url:

<https://docs.python.org/2/library/re.html>

Match Function

Syntax pattern for match function:

```
1 re.match(pattern , string , flags=0)
```

Match Function

Syntax pattern for match function:

```
1 re.match(pattern, string, flags=0)
```

```
1 import re
2
3 line = "Cats are smarter than dogs"
4
5 matchObj = re.match( r'(.*) are (.*) .*', line, re.I)
6
7 if matchObj:
8     print 1, matchObj.group()
9     print 2, matchObj.group(1)
10    print 3, matchObj.group(2)
11 else:
12    print "No match!!"
```


Matching vs Searching

match checks for a match only at the beginning of the string

Matching vs Searching

match checks for a match only at the beginning of the string

search checks for a match anywhere in the string

Matching vs Searching

match checks for a match only at the beginning of the string

search checks for a match anywhere in the string

```
1 import re
2
3 line = "Cats are smarter than dogs";
4
5 matchObj = re.match( r'dogs', line , re.I)
6 if matchObj:
7     print "match —> : ", matchObj.group()
8 else:
9     print "No match!!"
10
11 searchObj = re.search( r'dogs', line , re.I)
12 if searchObj:
13     print "search —> : ", searchObj.group()
14 else:
15     print "Nothing found!!"
```

Search and Replace

Some of the **re** methods that use regular expressions is **sub**.

Search and Replace

Some of the **re** methods that use regular expressions is **sub**.

```
1 re.sub(pattern , repl , string)
```

Search and Replace

Some of the **re** methods that use regular expressions is **sub**.

```
1 re.sub(pattern, repl, string)
```

```
1 import re
2
3 phone = "646-888-3395 # This is Phone Number"
4
5 # Delete Python-style comments
6 num = re.sub(r'#.*$', "", phone)
7 print "Phone Num : ", num
8
9 # Remove anything other than digits
10 num = re.sub(r'\D', "", phone)
11 print "Phone Num : ", num
```

Errors and Exceptions

Syntax errors.

```
1 while True print 'Hello world', break
```

Errors and Exceptions

Syntax errors.

```
1 while True print 'Hello world', break
```

An exception is an event, which occurs during the execution of a program.

```
1
2 try:
3     fh = open("testfile", "r") # reading the file
4 except IOError:
5     print "Error: can't find file for reading"
```


Errors and Exceptions

Syntax errors.

```
1 while True print 'Hello world', break
```

An exception is an event, which occurs during the execution of a program.

```
1
2 try:
3     fh = open("testfile", "r") # reading the file
4 except IOError:
5     print "Error: can't find file for reading"
```

For further references url:

<https://docs.python.org/2/tutorial/errors.html>

The assert Statement

Python interpreter evaluates the expression,
If the expression is false, raises an AssertionError exception.

```
1 assert Expression [, Arguments]
```

The assert Statement

Python interpreter evaluates the expression,
If the expression is false, raises an AssertionError exception.

```
1 assert Expression [, Arguments]
```

Test case to check for valid user input phone numbers:

```
1
2 import sys
3
4 try:
5     ph = sys.argv[1]
6 except:
7     print 'Provide a phone number'
8
9 assert len(ph)==10, 'Not a valid Phone numnber %s' % ph
```

Would love to hear your experience!

gabow@cbio.mskcc.org

vipin@cbio.mskcc.org