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.

List Comprehensions

List comprehensions provide a concise way to create lists.

```python
squares = []
for x in range(10):
    squares.append(x**2)

print squares
#[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

List Comprehensions

List comprehensions provide a concise way to create lists.

```
squares = []
for x in range(10):
    squares.append(x**2)

print squares
#[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

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

List Comprehensions

List comprehensions provide a concise way to create lists.

```python
squares = []
for x in range(10):
    squares.append(x**2)

print squares
#[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

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

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

List Comprehensions

List comprehensions provide a concise way to create lists.

```
squares = []
for x in range(10):
    squares.append(x**2)

print squares
#[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

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

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

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

del Statement

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

del Statement

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

```
xq = [-1, 1, 6, 3, 3, 12]
del xq[0]
xq
#[1, 6, 3, 3, 12]

del xq[2:4]
xq
#[1, 6, 12]

del xq[:] # deleting the contents
xq
#[]

del xq  # deleting the variable
```

Tuple - Another Data Type

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

Tuple - Another Data Type

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

```
t = 12, 54, 'hello!'
print t[0]
#12

# Tuples may be nested:
u = t, (1, 2, 3, 4, 5)
print u
#((12, 54, 'hello!'), (1, 2, 3, 4, 5))

# Tuples are immutable:
t[0] = 88888

# but they can contain mutable objects:
v = ([1, 2, 3], [3, 2, 1])
print v
#([1, 2, 3], [3, 2, 1])
```

Defining Functions

Little self-contained programs that perform a specific task.

Defining Functions

Little self-contained programs that perform a specific task.

Which you can incorporate into your own, larger programs.

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.

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'

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:

```
def compute_factorial(n):
    """
    computes factorial of n
    """

    ret = 1
    for i in xrange(n):
        ret=ret*(i+1)

    return ret
```

Defining Functions

function definition:

```
def compute_factorial(n):
    """
    computes factorial of n
    """

    ret = 1
    for i in xrange(n):
        ret=ret*(i+1)

    return ret
```

function call:

```

xq = compute_factorial(5)
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:

```
def compute_factorial(n):
    """
    computes factorial of n
    """

    ret = 1
    for i in xrange(n):
        ret=ret*(i+1)

    return ret
```

Modules

A module is a file containing Python definitions and statements.

factorial.py:

```
def compute_factorial(n):
    """
    computes factorial of n
    """

    ret = 1
    for i in xrange(n):
        ret=ret*(i+1)

    return ret
```

Import this module:

```

import factorial
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:

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

Regular Expression Operations

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

Meta characters:

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

Predefined characters:

```
"""
\d Matches any decimal digit [0-9].

\D Matches any non-digit character [^0-9].

\s Matches any whitespace character [ \t\n\r\f\v].

\S Matches any non-whitespace character [^ \t\n\r\f\v].

\w Matches any alphanumeric character [a-zA-Z0-9_].

\W Matches any non-alphanumeric character [^a-zA-Z0-9_].
"""
```

Regular Expression Operations

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

Meta characters:

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

Predefined characters:

```
"""
\d Matches any decimal digit [0-9].

\D Matches any non-digit character [^0-9].

\s Matches any whitespace character [ \t\n\r\f\v].

\S Matches any non-whitespace character [^ \t\n\r\f\v].

\w Matches any alphanumeric character [a-zA-Z0-9_].

\W Matches any non-alphanumeric character [^a-zA-Z0-9_].
"""
```

For further references url:

`https://docs.python.org/2/library/re.html`

Match Function

Syntax pattern for **match** function:

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

Match Function

Syntax pattern for **match** function:

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

```
import re

line = "Cats are smarter than dogs"

matchObj = re.match( r'(.*) are (.*?) .*', line, re.I)

if matchObj:
   print 1, matchObj.group()
   print 2, matchObj.group(1)
   print 3, matchObj.group(2)
else:
   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

```
import re

line = "Cats are smarter than dogs";

matchObj = re.match( r'dogs', line, re.I)
if matchObj:
    print "match --> : ", matchObj.group()
else:
    print "No match!!"

searchObj = re.search( r'dogs', line, re.I)
if searchObj:
    print "search --> : ", searchObj.group()
else:
    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**.

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

Search and Replace

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

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

```
import re

phone = "646-888-3395 # This is Phone Number"

# Delete Python-style comments
num = re.sub(r'#.*$', "", phone)
print "Phone Num : ", num

# Remove anything other than digits
num = re.sub(r'\D', "", phone)
print "Phone Num : ", num
```

Errors and Exceptions

error in syntax:

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

Errors and Exceptions

error in syntax:

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

Exception event, occurs during the execution of a program.

```

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

Errors and Exceptions

error in syntax:

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

Exception event, occurs during the execution of a program.

```

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

For further references:

`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.

The **assert** Statement

Python interpreter evaluates the expression,

If the expression is false, raises an **AssertionError** exception.

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

The **assert** Statement

Python interpreter evaluates the expression,

If the expression is false, raises an **AssertionError** exception.

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

Test case to check for valid user input phone numbers:

```

import sys

try:
    ph = sys.argv[1]
except:
    print 'Provide a phone number'

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