Basic Syntax

User Input

For python 2.7: str = raw_input("Please input")

For python 3.5: str = input("Please input")

Functions

```
def function_name(parameters):
        <expressions>
        return variable
```

Loop

1) While loop:

```
while <condition>:        # while True:
        <expression>
```

2) For loop:

```
for item in <list>:
        <expression>
```

3) break / continue

try/except

```
try: <expression>
except: <expression> quit()
```

Data Structure

String (immutable)

#length: len()

#slicing(from n and up to, BUT not including m): str[n:m]

#loop: 1) for letter in str: print letter 2) while i<len(str): …

#stripping: str.lstrip(), str.rstrip(), str.strip()

Str.startwith('text')

Files

handle = open(filename,mode) #mode = 'r','w'

1) for line in handle: prnt line
2) lines = handle.read()

List (mutable)

#element in the list: len(lst)

#sort the list: lst.sort()

max(lst), min(lst), sum(lst), len(lst)

#split(): abc = 'with;3;words' lst = abc.split(';') #space is the default

```
Lst = list(mydict.items())
Lst.sort(reverse = False)
```

Dictionary (mutable, like maps in C++, unordered)

mydict = dict()

#sample code, either construct or add: counts = dict() counts[name] = counts.get(name,0)+1

#return a list: mydict.keys() mydict.values() mydict.items()

#key-value loop: **for k, v in mydict:**

#print sorted (by value) dictionary: **print sorted([(v,k) for k,v in mydict.items()])**

Tuple (immutable & efficient & less resource)

mytup = tuple()

#member function: count() index()

Web

regex

```
Import re
re.search('REGEX',str)
re.findall('REGEX',str)
```

socket

```
import socket
mysock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
mysock.connect(('www.google.com',80))
mysock.send('GET http://www.google.com/sample.txt HTTP/1.0\n\n')
while True:
        data = mysock.recv(512)
        if (len(data)<1):
                break
        print data
mysock.close()
```

urllib

```
import urllib
fhand = urllib.urlopen('http://www.google.com/sample.txt')
for line in fhand:
        print line.strip()
```

BeautifulSoup

```
import urllib
import BeautifulSoup as BS
url = 'www.google.com'
html = urllib.urlopen(url).read()
soup = BS.BeautifulSoup(html)
tags = soup('a')
for tag in tags:
        print tag.get('href',None)
        print tag.contents
```

XML

1. start tag
2. end tag
3. text content
4. attribute
5. self-closing tag

```
<person>
        <name> Jack </name>
        <phone type = "intl"> +1 555 444 3333 </phone>
        <email hide = "yes" />
</person>
```

XML Schema (XSD Structure)

```
<person>
        <lastname> Semiz </lastname>
        <age>18</age>
</person>
```

```
<xs:complexType name = "person">
<xs:sequence>
        <xs:element name = "lastname" type = "xs:string"/>
        <xs:element name = "age" type = "xs:integer"/>
</xs:sequence>
</xs:complexType>
```

XML Sample Codes

```
import urllib
import xml.etree.ElementTree as ET
url = “http://python-data.dr-chuck.net/comments_42.xml”
str = urllib.urlopen(url).read()
stuff = ET.fromstring(str)
lst = stuff.findall(“comments/comment”)     # (“.//count”)
for item in lst: print item.find(‘count’).text
```

API Sample Codes

```
import urllib
import json
serviceurl = 'http://maps.googleapis.com/maps/api/geocode/json?'
while True:
        address = raw_input('Enter location: ')
        if len(address) < 1 : break
        url = serviceurl + urllib.urlencode({'sensor':'false','address': address})
        print 'Retrieving', url
        data = urllib.urlopen(url).read()
        try: js = json.loads(str(data))
        except: js = None
        if 'status' not in js or js['status'] != 'OK':
                print '==== Failure To Retrieve ====='
                continue
        print json.dumps(js, indent=4)
        lat = js["results"][0]["geometry"]["location"]["lat"]
        lng = js["results"][0]["geometry"]["location"]["lng"]
        print 'lat',lat,'lng',lng
        location = js['results'][0]['formatted_address']
        print location
```

Twitter API Sample Codes

```
import urllib
import twurl
import json
TWITTER_URL = 'https://api.twitter.com/1.1/friends/list.json'
acct = raw_input('Enter Twitter Account:')
if ( len(acct) < 1 ) : quit()
url = twurl.augment(TWITTER_URL, {'screen_name': acct, 'count': '5'} )
print 'Retrieving', url
connection = urllib.urlopen(url)
data = connection.read()
headers = connection.info().dict
print 'Remaining', headers['x-rate-limit-remaining']
js = json.loads(data)
print json.dumps(js, indent=4)
for u in js['users'] :
        print u['screen_name']
        s = u['status']['text']
        print ' ',s[:50]
```

SQL

Basic SQL Syntax

- CREATE TABLE users (name VARCHAR(128), email VARCHAR(128))
- INSERT INTO users (name, email) VALUES ('Christ','kf@umich.edu')
- DELETE FROM users WHERE email = 'kf@umich.edu'
- UPDATE users SET name = 'James' WHERE email = 'kf@umich.edu'
- SELECT * FROM users ORDER BY email
- NOT NULL/PRIMARY KEY/AUTOINCREMENT/UNIQUE
- #combined variables: PRIMARY KEY(var1, var2)

sqlite3

sample code

```
import sqlite3
conn = sqlite3.connect('emaildb.sqlite')
cur = conn.cursor()
cur.execute('''DROP TABLE IF EXISTS Counts''')
cur.execute('''CREATE TABLE Counts (email TEXT, count INTEGER)''')
fname = 'mbox-short.txt'
fh = open(fname)
for line in fh:
        if not line.startswith('From: '): continue
        pieces = line.splite()
        email.pieces[1]
        cur.execute('SELECT count FROM Counts WHERE email = ?', (email,))
        try:
                count = cur.fetchone()[0]
                cur.execute('UPDATE Counts SET count = count+1 WHERE email = ?',(email,))
        except:
                cur.execute('''INSERT TNTO Counts (email, count) VALUE (?,1)''',(email,))
        conn.commit()
sqlstr = 'SELECT email, count FROM Counts ORDER BY count DESC LIMIT 10'

for row in cur.execute(sqlstr):
        print str(row[0]), row[1]
cur.close()
```

syntax

```
cur.executescript("<sql codes>")
#either insert or does nothing if the content exists: INSERT OR IGNORE
```

Exceptions

Extensions to try

except: body of this clause is executed when execution of associated **try** body raises exceptions.

else: body of this clause is executed when execution of associated **try** body completes with no exceptions.

finally: body of this clause is always executed after **try**, **else** and **except** clauses, even if they raised another error or executed a **break**, **continue** or **return**.

Assertions

Assert: <expression1>, <expression2, or string>

if expression1 fails, run part2 and raise AssertionError; or just ignore part2

Generators

Fibonacci

```
def genFib():
    fibn_1 = 1
    fibn_2 = 0
    while True:
        next = fibn_1+fibn_2
        yield next
        fibn_2 = fibn_1
        fibn_1 = next
```

```
fib = genFib()
fib.next()
fib.next()
fib.next()
…
```

Prime Number

```
def genPrimes():
    primes = []   # primes generated so far
    last = 1      # last number tried
    while True:
        last += 1
        for p in primes:
            if last % p == 0:
                break
        else:
            primes.append(last)
            yield last
```

numpy

```
from numpy import *
my_array = array([raw_input().split() for i in range(N)], int)
print my_array.shape
reshape(my_array,(row,column))  #reshape the matrix
transpose(my_array)  #transpose the matrix
my_array.flatten()  #turn the matrix into array
concatenate((array1, array2), axis = 0,1,None)  #0: by column; 1: by row
zeros((row, column), dtype = int)
ones((row, column), dtype = int)
identity(Dim)  eye(row, column, k = N)
dot(A,B)  #matrix production
cross(A,B) #cross production
inner(A,B)  #inner production
outer(A,B)  #outer production
linalg.det(matrix)  #determinant
linalg.eig(matrix)  #eigenvalues
linalg.inv(matrix)  #inverse
```

Tree

Depth first search:

- Start with the root
- At any node, if we haven't reached our objectives, take the left branch first
- When get to a leaf, backtrack to the first decision point and take the right branch

```
def DFSBinary(root, fcn):
    stack = [root]
    while len(stack) > 0:
        print 'at node ' + str(stack[0].getValue())
        if fcn(stack[0]):
            return True
        else:
            temp = stack.pop(0)
            if temp.getRightBranch():
                stack.insert(0, temp.getRightBranch())
            if temp.getLeftBranch():
                stack.insert(0, temp.getLeftBranch())
    return False
```

Breath first search:

- Start with the root
- Then proceed to each child at the next level, in order
- Continue until reach objective

```
def BFSBinary(root, fcn):
    queue = [root]
    while len(queue) > 0:
        print 'at node ' + str(queue[0].getValue())
        if fcn(queue[0]):
            return True
        else:
            temp = queue.pop(0)
            if temp.getLeftBranch():
                queue.append(temp.getLeftBranch())
            if temp.getRightBranch():
                queue.append(temp.getRightBranch())
    return False
```

Ordered depth first search:

```
def DFSBinaryOrdered(root, fcn, ltFcn):
    stack = [root]
    while len(stack) > 0:
        if fcn(stack[0]):
            return True
        elif ltFcn(stack[0]):
            temp = stack.pop(0)
            if temp.getLeftBranch():
                stack.insert(0, temp.getLeftBranch())
        else:
            temp = stack.pop(0)
            if temp.getRightBranch():
                stack.insert(0, temp.getRightBranch())
    return False
```