Commands:

```
type(x)                  int\str\...
isinstance(x,class)
print x, y, z
len(x)                   __len__(self)
str(x)                   convert to string
int(x)                   convert to integer
float(x)                 convert to float
tuple(x)                 convert to tuple
dict(x)                convert to dictionary
raw_input('text')
abs(x)                   __abs__(self)
global x                 make x global
max(x,y,key=f)
min(x,y.key=f)
cmp(x,y)              x<y:-1, x==y:0, x>y:1
ord(x)                   ASCII # of char
chr(x)                   char of ASCII #
op(enumerate(x))        (index,result)
zip([1,2,3],[4,5,6])   [(1,4),(2,5),(3,6)]
sorted(items to be sorted, cmp =
comparator function, key = pre-comparison
function, reverse = True\False)
```

Operations:

```
x + y          __add__(s,o)
y + x          __radd__(s,o)
a + b        __concat__(s,o)
x - y          __sub__(s,o)
-y             __neg__(s)
x * y          __mul__(s,o)
x **y          __pow__(s,o)
x / y          __div__(s,o)
x % y          __mod__(s,o)
x <  y         __lt__(s,o)
x <= y         __le__(s,o)
x >  y         __gt__(s,o)
x >= y         __ge__(s,o)
x == y         __eq__(s,o)
x != y         __ne__(s,o)
x in y    __contains__(s,o)
x is y
x and y
x or y
not x
 +=
 -=
```

Tuple Handling:

```
w = (x,y,z)
a,b,c = w

a=x, b=y, c=z
```

String Handling:

```
string[1:3]                tr
string[::-1]               gnirts
string[:-1]                strin
string[-3:]                ing
string[:-3]                str
string[-1]                 g
string.find('s')           0
string.upper()             STRING
string.lower()             string
string.replace('str','th')    thing
string.startswith(x)       T\F
'-'.join([a,b,c])          'a-b-c'
string.split()    list split by '_'
string.isalpha()    is letter?
string.rstrip(x)    cut x's from end
```

Random:

```
from random import random
random() - 0-1
from random import randint
randint(x,y) - x-y
from random import choice
choice('abcd') - random from string
```

List Handling:

```
list = [a,b,c,w]
list[0]                  a
list[0] = aa          [aa,b,c,w]
list[0:3:2]           [aa,c]
list.insert(3, v)     [aa,b,c,v,w]
list.append(x)     [aa,b,c,v,w,x]
list.extend([y,z])     [...,y,z]
list[0:6] =[t]         [t,y,z]
list.remove(y)         [t,z]
copy = list[:]
list.pop(x)      remove index x
list.reverse()
list.sort()
list.count(x)    count x in list
```

Dictionary Handling:

```
dict = {k1:v1, k2:v2}
dict[k1]                       v0
dict[k3]                       v3
dict.get(k, default)
dict.has_key(k) \ k in dict
dict.items()    [(k1,v1),(k2,v2)]
dict.keys()
dict.values()
dict.pop(k, default)
dict.update(dict2)
sum(list)  if numbers, returns sum
list[0][1]   index 1 in index 0
list + [x]   same as append(x)
```

Math (Numpy):

```
import numpy as np
np.log(8)/np.log(2)      3
np.sqrt(x)
np.loadtxt('file.txt', delimiter=',')
np.int_(x)
np.uint8(x)
np.arange(1,2.5,0.5)     [1,1.5,2]
np.linspace(1,3,5)     [1,1.5,2,2.5,3]
np.random.random_integers(lo,hi,size)
np.random.rand(2)       [0-1,0-1]
```

Range Handling:

```
range(2,5)      [2,3,4]
range(0,5,2)    [0,2,4]
range(5,0,-2)   [5,3,1]
```

Array Handling:

```
np.array([[1,2,3],[4,5,6]])
x.min\max()        min\max values within array
x[r1:r2:skip,c1:c2:skip]
x.sum(axis=0\1)    sum of column or row
np.sum(x)          sum of entire array
x.sort(axis=0\1)   sort array
np.sort(x,axis=0\1)   instance of sorted array
np.minimum(x,y)    array of min values of x and y
np.maximum(x,y)    array of max values of x and y
np.mean(x)         mean of entire array
x.mean(axis=0)     array of mean of each col\row
np.median(x)       median of entire array
x.median(axis=0)   array of median of each col\row
np.concatenate((x,y),axis=0\1)
np.diag(v)         diagonal of vector
x.T                Transpose
x[a,b]             value at coordinate
x = np.copy(y)     make x a copy of y
np.vdot(x,y)       dot product of x and y
axis=0         along vertical (each column)
axis=1         along horizontal (each row)
x.shape[0]     Rows (including)
x.shape[1]     Columns (including)
x.resize((height including, width including))
x == y         [T,F,T,F,F...]
x[r,:]         row r
x[:,c]         column c
x.any()        are any true?
(x==0).any()   does x contain 0?
x.all()        are all true?
x.nonzero()    get true indices
x op y         array of op results between x and y
x op a         array of op results with a over x
x[mask] = c        where mask is True, change to c
x[(x>a)&(x<b)]=c    where condition is met, change to c
x[x!=a]= c          where value not a, change to c
x = np.zeros_like(y)    make x a zero-copy of y
x = np.full_like(y,val) make x y-shaped and fill with val
```

Graphs:

```
import matplotlib.pyplot as plt
x = np.linspace(-np.pi,np.pi)
y,z = np.cos(x),np.sin(x)
plt.plot(x,y)
plt.plot(x,z)
plt.show()
```

Image Handling:

```
import matplotlib.pyplot as plt
from scipy import misc
image = misc.imread('pic.jpg')
misc.imsave('new.bmp',image)
image.rotate(deg)
```

File Handling:

```
url = urllib.urlopen(x).read()
file = open('file.txt', 'w\r\a')
file.read()
for line in file
file.readline()
file.readlines()
file.close()
file.write(str)
'\n' - new line
```

Monte Carlo Pi Calculation

```
from random import random
n = 10000
circle = Circle(Point(0.5, 0.5), 0.5)
count = 0
points = [Point(random(), random()) for
i in range(n)]
count = sum([1 for p in points if p in
circle])
print count * 4.0 / n
```

Show Image:

```
plt.figure()
plt.imshow(image,cmap=plt.cm.gray)
plt.show()
```

Empty B&W Image:

```
x = np.zeros((height incl,width
incl),dtype=np.uint8)
```

Equation Solver:

```
from scipy import linalg
def solve_by_inverse(A, b):
___return linalg.inv(A).dot(b)
A=np.array([[1,2],[3,4]])
b=np.array([17,39])
solve_by_inverse(A, b) - [5,6]
```

Vertical Gradient:

```
zero = np.zeros((1,im.shape[1]), dtype=np.uint8)
shift = np.concatenate((im[1:], zero), axis=0)
diff = np.int_(shift)-np.int_(im)
imDy = np.abs(diff)
```

Get Neighbours:

```
def neighbours(matrix, x, y, nx=1, ny=1):
return matrix[max(x-nx, 0) : min(x+nx+1,
matrix.shape[0]), \
max(y-ny, 0) : min(y+ny+1,
matrix.shape[1])]
```

Optimal Cost w/ Memo:

```
def optimal_cost_memo (N, prices, start=1):
___if start not in memo:
______min_price = prices[start, N]
_____________for i in range(N - 1, start, -1):
_____________opt_i = optimal_cost_memo (N, prices, i, memo)
_____________min_price = min(min_price, prices[start, i] + opt_i)
______memo[start] = min_price
___return memo[start]
```

Squeeze Image:

```
def squeeze_image(im,factor):
___new_n = im.shape[0]
___new_m = im.shape[1] / factor
___new_mat = np.zeros((new_n,new_m))
___for j in range(new_mat.shape[1]):
______curr=range(j*factor,min((j+1)*factor,im.shape[1]))
______new_mat[:,j] = im[:,curr].mean(axis=1)
___return new_mat
```

Condition:

```
if condition:
___statement
elif:
___statement
else:
___statement
```

Loops:

```
for i in iterable:
___statement
while expression:
___statement
___count += 1
___break - exits loop
___continue - skips cycle
```

Functions:

```
def function(x,y):
___body
___return value
def rec (x,y):
___if x == 0: #stop
______return 0
___return rec (x-1) + y
```

Memoization:

```
def memo (x,y,mem=None):
___if mem==None:
______mem={}
___if n not in mem:
______rec(x-1,mem) + y
______mem[n]=value
___return mem[n]
```

Greatest Common Divider:

```
def GCD(x,y):
___if y==0:
______return x
___return GCD(y,x%y)
```

Error Handling:

```
raise errortype('text')

ValueError
TypeError
ZeroDivisionError

try:
___
except exception as err1:
___print err1.args[0]
finally: #runs anyway
___
```

Object Handling:

```
class classname:
___def __init__(self,x,y):
______self.a = x
______self.b = y
___def method(self,z):
______body
______return value
__def __repr__(self):
______return str(self.a)
__def__str__(self)
______return str(self.a)

object = classname(x,y)
object.a = w
object.method(z)
```

Reminders:

- Define parameters before using them.
- Use functions from previous questions.
- Use class properties from previous questions.
- Be sure to use 'self' in method definition and calling.
- Treat methods as functions (specific input and output).
- When validating array entries: (0<=i<M).
- Length\Height\Width are including, range is not.
- Use range(len()) in loops if possible.
- In loops, make new data and re-assign rather than edit.
- Use == for comparison, not =.
- Note difference between arrays and matrix-like lists.
- Do not forget calling of non-returning functions when required.

Search & Sort:

Binary Search - in sorted list - if value<middle, search lower half of list.
Bubble Sort - compare sequentially and swap if needed
Merge Sort - split in two until length of one, sort each by parts, merge and sort.
Counting Sort - histogram and rebuild in order.
Selection Sort - find index of minimal\maximal value in sublist and swap with start\end if needed.
Insertion Sort - assume sorted so far, insert in correct place and continue until end.
Bucket Sort - put each x into k buckets by [x/k], sort each bucket and merge sequentially.
Shell Sort - variant of Insertion; sort first objects in each half, divide by four, sort again and so on until sorted.

Distance:

Hamming Distance - number of positions at which corresponding symbols are different.
QWERTY Distance - horizontal and vertical distances between keys on keyboard representing each corresponding symbol.
Levenshtein Distance - number of edits (insert, delete or substitute) required to transform one word into another.

Polynomial Handling:

```
import numpy.polynomial.
polynomial as poly
p1= poly.Polynomial([6,-5,1])

p1.coef,  p1.roots(),  p1(x)
```

Cumulative Sum:

```
def cum_sum(lst):
___a = []
___for i in range(len(lst)):
______a.append(sum(lst[:i])+lst[i])
___return a
```

Permutation:

```
def permute(elems):
___if len(elems) == 0:
______return [[]]
___perms = []
___for curr in elems:
______elems_minus_curr = elems[:]
______elems_minus_curr.remove(curr)
______sub_perms =
permute(elems_minus_curr)
______for perm in sub_perms:
_________perm.append(curr)
______perms.extend(sub_perms)
___return perms
```

Bucket Sort:

```
def bucket_sort(lst):
___buckets=[]
___for i in range(10):
______buckets.append([])
___for x in lst:
______buckets[x / 10].append(x)
___sorted_list = [ ]
___for b in buckets:
______if len(b) > 0:
_________b = selection_sort(b)
______sorted_list.extend(b)
___return sorted_list
```

Selection Sort:

```
def selection_sort(lst):
___lst = lst[:]
___for i in range(len(lst)):
______m = min_index(lst, i)
______lst[m],lst[i] = lst[i], lst[m]
___return lst
```

Bubble Sort:

```
def bubble_sort(lst):
___for i in range(len(lst)-1, 0, -1):
______for j in range(i):
_________if lst[j] > lst[j+1]:
_____________lst[j], lst[j+1]= lst[j+1], lst[j]
```

Histogram:

```
def histogram(string):
___d = { }
___for char in string:
______count = d.get(char,0)
______d[char] = count + 1
___return d
```

Anagram Check:

```
def anagrams(s1,s2):
___if len(s1) != len(s2):
______return False
___d1 = histogram(s1)
___d2 = histogram(s2)
___return d1 == d2
```

Binary Search:

```
def bin_iter(key,vals):
___n = len(vals)
___lo = 0
___up = n-1
___while up >= low:
______mid = (up + lo) / 2
______if key == vals[mid]:
_________return True
______elif key< vals[mid]:
_________up = mid - 1
______else:
_________lo = mid + 1
___return False
```

Fibonacci w\ Memo:

```
def fib(n,mem=None):
___if n<2:
______return n
___if mem == None:
______mem = {}
___if n not in mem:
______mem[n] = \
fib(n-1,mem)+fib(n-2,mem)
___return mem[n]
```

Integrate Between 0 & 1:

```
from scipy import integrate
integrate.quad(f,0,1)[0]
```

Sort Dictionary Keys by Values:

```
def tuc(tup):
___return tup[1]
def keys_by_values(d):
___items = list(d.items())
___sort_list=sorted(items,key=tuc)
___sort_keys = []
___for item in sort_list:
______sort_keys.append(item[0])
___return sorted_keys
```

Hamming Distance:

```
def hamming_dist(w1,w2):
___score = abs(len(w1) - len(w2))
______for c1,c2 in zip(w1,w2):
_________if c1 != c2:
_____________score += 1
___return score
```

Levenstein Distance:

```
def lev(s1,s2):
___if len(s1)==0 or len(s2)==0:
______return max(len(s1),len(s2))
___d1 = lev(s1[:-1],s2)+1
___d2 = lev(s1,s2[:-1])+1
___last = 0 if s1[-1]==s2[-1] else 1
___d3 = lev(s1[:-1],s2[:-1])+last
___return min(d1,d2,d3)
```

Reverse Digits:

```
def reverse_digits(num):
___if num==0:
______return 0
___d = len(str(num))-1
___last_num = num%10
___last_num = last_num*(10**d)
___return last_num+reverse_digits(num/10)
```