Python Programming

Chapter 3

Control Structures

Topics Covering

Control structures:

- if statement
- if - else statement
- if - elif statement
- Nest if-else
- Multiple if
- Which control structure to choose?
- Looping statements
    - while loop
    - for loop
        - range()
        - xrange()
        - Iterator and generator Introduction
    - for - else
    - When to use for-else ?
- Interview Questions
- Exercise Programs
- Summary

Control Structures

Control structures are used to control the flow of execution. Some, times we may require to skip the execution of some statements. Some times we may want to execute two blocks of statements exclusively. Some times may need to execute a block of statements repeatedly. Below are the control statments we are going to discuss now.

1. if
2. if-else
3. if-elif ladder
4. multiple if
5. nested if-else
6. while loop
7. for loop

if statement:

Syntax:

```
if condition:
    statements
```

Program: Read a number from keyboard, and if the number is even, print 'EVEN'.

In[]
```
x = int(raw_input("Enter x:"))

if x%2 == 0:
    print '----'
    print "Even"
    print '----'

print 'The End'
```

```
Enter x:3
The End
```

If the given number is not even, we do not see any ouput.

Identation: Python identifies code-blocks using indentation. In the above program, we can see that, all the statatements which needs to be excuted when condition is True, should be placed after 'if condition:' preceeded by one tab(generally 4 spaces). All the statments after if statment with one tab indentation, are considered as if block. lines 4, 5, 6 belong to if block, but not line 8

if-else statement:

When we want to execute two blocks of code exclusively we use if-else statement. Only one block of code is executed all the time.

Syntax:

```
if condition:

else:
```

Program: Read a number from keyboard, and if the number is even, print 'EVEN' else print 'ODD'.

In[]
```
x = int(raw_input("Enter x:"))

if x%2 == 0:
    print "Even"
else:
    print "Odd"

print 'The End'
```

```
Enter x:4
Even
The End
```

if-elif ladder - more exclusive cases

When we have mulitple exclusive cases, we use if-elif ladder.

Syntax:

```
if condition:

elif condition:

elif condition:


else:
```

Program: Read a character from keyboard. If the given input is,

'a' - print 'Apple'
'b' - print 'Ball'
'c' - print 'Cat'
'd' - print 'Dog'
Any other character, print 'Unknown'

In[]
```
char = raw_input("Enter character:")

if char == 'a':
    print "Apple"
elif char == 'b':
    print "Ball"
elif char == 'c':
    print "Cat"
elif char == 'd':
    print "Dog"
else:
    print "Unknown"

print 'The End'
```

```
Enter character:d
Dog
The End
```

Mutiple if: a special case

Eventhough it is not a seperate control statement, there is a special use case for this. When the given input falls under multiple categories we use mutiple ifs.

Program: Which of the following numbers [3, 7, 11] are the factors for a given number

In[]
```
x = int(raw_input("Enter x:"))

if x%3 == 0 :
    print "3 is a factor"

if x%7 == 0 :
    print "7 is a factor"

if x%11 == 0 :
    print "11 is a factor"
```

```
Enter x:33
3 is a factor
11 is a factor
```

Nested if-else: one more special case

Like multiple if, nested if-else is also a basic control structure it has a special use-case. When we have multiple sub-categories under a condition we use nesed if-else.

Program: Read a number from keyboard,
if the number is even square it
if it is odd multiple of 5 cube it
if it is odd non-multiple of 5 mutiply with 10

In[]
```
x = int(raw_input("Enter x:"))

if x%2 == 0:
    print x**2
else:
    if x%5 == 0:
        print x**3
    else:
        print x*10
```

```
Enter x:17
170
```

Looping statements

When a block of statements is required to be executed, repeatedly, we use looping staments. There are three looping stataments in python.

1. while
2. for
3. for-else

While loop:

As log as condition is true, block of staments under while are executed repeatedly. Once condition is false, loop stops exection and control goes immediate next stament after while block.

Syntax:

```
while condition:
    statements
```

Program: Print first 10 natural numbers.

In[]
```
i = 1
while i <= 10:
    print i
    i += 1
```

```
1
2
3
4
5
6
7
8
9
10
```

Observe the behaviour of above while loop carefully. 1 is the start value of 'i', and then checking the condition, 'i <= 10'. I if it is True, conteol is going inside the loop. Inside while, 'i' is getting incremented by 1(step) in each iteration of the loop. Actually, 'i' is moving towards the end value 10. start, end and step are the three factors on which loop execution depends. Now, we can understand how string slicing is wroking in python.

Program: Print multiples of 3 below 10

In[]
```
i = 1
while i <= 10:
    if i%3 == 0:
        print i
    i += 1
```

```
3
6
9
```

Exercises for non-programmers

Program: Count number of multiples of 3 below 100.

In[]
```
i = 1
c = 0
while i <= 100:
    if i%3 == 0:
        c += 1
    i += 1
print "Number of multiples of 3 below 100 : ", c
```

```
Number of multiples of 3 below 100 :  33
```

Program: Count number of multiples of 7 from 100 to 135.

In[]
```
i = 100
c = 0
while i <= 135:
    if i%7 == 0:
        c += 1
    i += 1
print "Number of multiples of 7 between 100 and 135: ",  c
```

```
Number of multiples of 7 between 100  and 135:  5
```

Program: Sum of the multiples of 3 below 100

In[]
```
i = 1
s = 0
while i <= 100:
    if i%3 == 0:
        s += i
    i += 1
print "sum of the multiples of 3 below 100 : ", s
```

```
sum of the multiples of 3 below 100 :  1683
```

Program: Multiples of n below 100.

In[]
```
n = int(raw_input("Enter n value:"))
i = 1
while i <= 100:
    if i%n == 0:
        print i
    i+=1
```

```
Enter n value:13
13
26
39
52
65
78
91
```

Program: Printing factors of a given number.

In[]
```
n = int(raw_input("Enter n:"))
i = 1
while i <= n:
    if n%i ==0:
        print i
    i+=1
```

```
Enter n:12
1
2
3
4
6
12
```

Program: Printing number of factors of a given number

In[]
```
x = int(raw_input("Enter x:"))
i = 1
c = 0
while i <= x:
    if x%i ==0:
        c += 1
    i+=1
print "Number of factors of {} is {}".format(x, c)
```

```
Enter x:12
Number of factors of 12 is 6
```

Program: Print the digits in the given number in reverse order.

In[]
```
n = int(raw_input("Enter x:"))

while n > 0:
    r = n%10
    print r,
    n /= 10
```

```
Enter x:12345
5 4 3 2 1
```

Program: Print count of the digits in the given number.

In[]
```
n = int(raw_input("Enter x:"))
c = 0
b = n
while n > 0:
    n /= 10
    c += 1

print 'Number of digits in {} is {}'.format(b, c)
```

```
Enter x:12345
Number of digits in 12345 is 5
```

Program: Sum of the digits in the number

In[]
```
n = int(raw_input("Enter x:"))
b = n
s = 0

while n > 0:
    r = n%10
    n /= 10
    s += r

print 'Sum of the digits in {} is {}'.format(b, s)
```

```
Enter x:12345
Sum of the digits in 12345 is 15
```

Program: Magic number of the given number. Sum all the digits, until you get single digit sum.

In[]
```
n = int(raw_input("Enter x:"))
b = n
s = n

while s > 9:
    n = s
    s = 0
    while n > 0:
        r = n%10
        s += r
        n /= 10

print 'Magic number of {} is {}'.format(b, s)
```

```
Enter x:12345
Magic number of 12345 is 6
```

Program: Reverse the given number.

In[]
```
n = int(raw_input("Enter x:"))
b = n
rev = 0

while n > 0:
    r = n%10
    rev = rev*10 + r
    n = n/10

print 'reverse of {} is {}'.format(b, rev)
```

```
Enter x:12345
reverse of 12345 is 54321
```

Program: Check the given number is prime or not.

In[]
```
n = int(raw_input("Enter x:"))
i = 1
c = 0
while i <= n:
    if n%i == 0:
        c += 1
    i+=1

if c == 2:
    print "Prime"
else:
    print "Not Prime"
```

```
Enter x:117
Not Prime
```

In[]
```
n = int(raw_input("Enter x:"))
i = 2
c = 0
while i <= n//2:
    if n%i == 0:
        c += 1
    i+=1

if c == 0:
    print "Prime"
else:
    print "Not Prime"
```

```
Enter x:117
Not Prime
```

In[]
```
n = int(raw_input("Enter x:"))
i = 2
c = 0
rt = n**0.5

while i <= rt:
    if n%i == 0:
        c += 1
    i+=1

if c == 0:
    print "Prime"
else:
    print "Not Prime"
```

```
Enter x:117
Not Prime
```

Program: Check the given number is prime or not. [Effcicent way]

In[]
```
n = int(raw_input("Enter x:"))
i = 2
rt = int(n**0.5)

while i <= rt:
    if n%i ==0:
        print 'Not Prime'
        break
    i+=1

if i > rt:
    print 'Prime'
```

```
Enter x:117
Not Prime
```

for loop:

Unlike other programming languages, for loop in python completely works on iteration protocol. iterable in the below syntax should provide a next value in each iteration to iter_var.

Syntax:

```
for iter_var in iterable:
    statements
```

In[]
```
for i in range(5):
    print i
```

```
0
1
2
3
4
```

Using range() function in for loops

range() is a function which constructs and returns an iterable list, with the given range.

Syntax:

```
range(start, end, step)
```

Some examples of range:

In[]
```
range(5)
```

Output: [0, 1, 2, 3, 4]

In[]
```
range(1, 6)
```

Output: [1, 2, 3, 4, 5]

In[]
```
range(1, 11, 2)
```

Output: [1, 3, 5, 7, 9]

Program: Printing first 10 natural numbers using for

In[]
```
for i in range(1, 11):
    print i
```

```
1
2
3
4
5
6
7
8
9
10
```

This is equivalent to below while loop.

In[]
```
i = 1
while i < 11:
    print i
    i += 1
```

```
1
2
3
4
5
6
7
8
9
10
```

Note: In python 2.x range() returns a list, where as in 3.x it returns a generator

In[]
```
for x in xrange(1,11,2):
    print x
```

```
1
3
5
7
9
```

In[]
```
a = range(1, 100)
b = xrange(1,  100)

print type(a), type(b)
```

```
<type 'list'> <type 'xrange'>
```

In[]
```

s = 0
for x in range(1,11):
    s += x

print s


```

```
55
```

In[]
```
for i in range(1,6):
    print i
```

```
1
2
3
4
5
```

Below loop will not print anything as end=1, which is exclusive.

In[]
```
for i in range(1, 1):
    print i
```

If we want to iterate the loop one time, 'end' must be 2 for the below loop.

In[]
```
for i in range(1, 1+1):
    print i
```

```
1
```

In[]
```
for i in range(4):
    for j in range(3):
        print "Apple",
```

```
Apple Apple Apple Apple Apple Apple Apple Apple Apple Apple Apple
Apple
```

Printing in the same line

Each print stament takes the control to the net line. If you to keep the control in the same line, use comma(,) at the end of print statement line.

In[]
```
print 'Apple',
print 'Orange',
```

```
Apple Orange
```

Program: Print the below trianlge using for loop.

```
*
* *
* * *
* * * *
* * * * *
```

In[]
```
for i in xrange(1, 6):
    for j in xrange(1, i+1):
        print '*',
    print ''
```

```
*
* *
* * *
* * * *
* * * * *
```

Exercises for non-programmers

Program: Print the below trianlge using for loop.

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

In[]
```
for i in xrange(1,6):
    for j in xrange(1, i+1):
        print i,
    print ''
```

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

Program: Print the below trianlge using for loop.

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

In[]
```
for i in xrange(1,6):
    for j in xrange(1, i+1):
        print j,
    print ''
```

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Program: Print the below trianlge using for loop.

```
5
4 4
3 3 3
2 2 2 2
1 1 1 1 1
```

In[]
```
for i in xrange(5,0, -1):
    for j in xrange(5,i-1, -1):
        print i,
    print ''
```

```
5
4 4
3 3 3
2 2 2 2
1 1 1 1 1
```

Program: Print the below trianlge using for loop.

```
5
5 4
5 4 3
5 4 3 2
5 4 3 2 1
```

In[]
```
for i in xrange(5,0, -1):
    for j in xrange(5,i-1, -1):
        print j,
    print ''
```

```
5
5 4
5 4 3
5 4 3 2
5 4 3 2 1
```

In[]
```
c = 1
for i in xrange(1, 6):
    for j in xrange(1, i+1):
        print c,
        c += 1
    print ''
```

```
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
```

break:

break statement breaks the execution of while or for loops, control trnasfered to immediate next statement of the loop. Refer the above example.

In[]
```
n = int(raw_input("Enter Value:"))
rt = int(n ** 0.5)

i = 2
while i <= rt:
    if n%i == 0:
        print "Not Prime"
        break
    i += 1

if i > rt:
    print "Prime"
```

```
Enter Value:117
Not Prime
```

In[]
```
n = int(raw_input("Enter Value:"))
rt = int(n ** 0.5)
flag = True
i = 2

for i in range(2, rt+1):
    if n%i == 0:
        print "Not Prime"
        flag = False
        break

if flag:
    print "Prime"
```

```
Enter Value:117
Not Prime
```

for-else

**IQ. 'else' block gets executed on the successfull completion of for loop

In[]
```
from math import sqrt
n = int(raw_input("Enter Value:"))
rt = int(sqrt(n))

for i in range(2, rt+1):
    if n%i == 0:
        print "Not Prime"
        break
else:
    print "Prime"
```

```
Enter Value:117
Not Prime
```

Program: Print the prime numbers between 50 to 100.

In[]
```
from math import sqrt

for j in range(50, 101):
    for i in range(2, int(sqrt(j)+1)):
        if j%i == 0:
            break
    else:
        print j
```

```
53
59
61
67
71
73
79
83
89
97
```

continue:

continue skips the execution to next iteration of the loop.

Program:
Sum all the elements in the range() which are non-multiples of 3

In[]
```
s = 0
for x in range(1, 11):
    if x%3 == 0:
        continue
    s += x

print "sum of non-multiples of 3: ", s
```

```
sum of non-multiples of 3:  37
```

Interview Questions

1. How do you swap two values?
2. range() vs xrange() ?
3. How for-else construct works?

Practice programs

Program 1: A Company insures its drivers in the following cases

1. If the driver is married
2. If the driver is unmarried, male and above 30 years of age
3. If the driver is unmarried, female and above 25 years of age
   in all other cases, the driver is not insured. If the marital status, gender, age of the drivers are the inputs, write a program to determine whether the driver is isured or not.(use 'nested - if')

Solution:

In[]
```
status = raw_input('Enter marrital status[M/U]:')
gender = raw_input('Enter gender[M/F]:')
age    = int(raw_input('Enter age:'))

if status == 'M':
    print 'Inssured'
else:
    if gender == 'M' and age >= 30:
        print 'Inssured'
    elif gender == 'F' and age >= 25:
        print 'Inssured'
    else:
        print 'Not Inssured'
```

```
Enter marrital status[M/U]:U
Enter gender[M/F]:M
Enter age:26
Not Inssured
```

Program 2: Indian income tax calcuation:
0% tax - 2.5 lac for men, 3.0 lac for women
5% tax - till 5.0 lac
20% tax - till 10 lac
30% tax - for above 10 lac

In[]
```
actual_salary = float(raw_input('Enter salary:'))
gender = raw_input('Enter gender[M/F]:')

tax = 0.0
salary = actual_salary

if salary > 1000000:
    slab = salary - 1000000
    tax = slab * 0.3
    salary -= slab

if salary > 500000:
    slab = salary - 500000
    tax += slab * 0.2
    salary -= slab

if salary > 300000:
    slab = salary - 300000
    tax += slab * 0.05
    salary -= slab

if gender == 'M':
    slab = salary - 250000
    tax += slab * 0.05

print 'tax for {} is {} if gender is {}'.format(actual_salary, tax,
gender)
```

```
Enter salary:1370000
Enter gender[M/F]:M
tax for 1370000.0 is 223500.0 if gender is M
```

Program 3: Write a programe to read the characters continuosly until '$' is given and display the number of characters entered.

Program 4: Write a programe to read a character and find out whether it is uppercase or lowercare

Solution:

In[]
```
ch = raw_input('Enetr a charcater:')

if ch.isdigit():
    print 'Digit Character'
elif ch.islower():
    print 'Lower case Character'
elif ch.isupper():
    print 'Upper case Character'
else:
    print 'Special Character'

print 'The End'
```

```
Enetr a charcater:%
Special Character
The End
```

Programe 5: Write a program to print the uppercase letter of a given lowercase

Program 6: Write a program to check whether the given input is digit or lowercase character or uppercase character or a special character(use 'if-else-if' lasser)

Program 7: Write a program to print all prime numbers between 100 to 200 using only for and for-else looping statements.

Solution:

"s1">'Data1'), ('196.131.56.65', 'Data2'), ('196.131.56.71', 'Data1')] dict_set = defaultdict(set) for ip, data in packets: dict_set[ip].add(data) for ip, pkts in dict_set.items(): print ip, '->', sorted(pkts)

```
196.131.56.78 -> ['Data1']
196.131.56.71 -> ['Data1', 'Data2']
196.131.56.65 -> ['Data1', 'Data2']
196.131.56.44 -> ['Data1']
196.131.56.35 -> ['Data1']
```

Heapq

Python heapq is min-heap data structure implemenation. Internals of this data structures is out of scope for this materail. But we discuss operations and Use-Cases of this data structure. It is physically stored as dynamic array, but logically a binary tree. Always maintains smallest element as root. root is stored at o'th index always. Each time we pop an element from heapq, it returns the smallest. Each time we push an element, heapq, rearranges elements and pushes the smallest to 0'th index (in just O(log n) time).

We need to import heapq module to use this functionality.

```
import heapq
```

Frequenctly used heapq operations:

heapq.heapify(x): This function transforms list x into a heap.

In[]
```
import heapq
l = [4, 20, 6, 9, 2, 15]
heapq.heapify(l)
print l
```

```
[2, 4, 6, 9, 20, 15]
```

heapq.heappush(heap, item): adding an element to heap.

In[]
```
import heapq
l = [4, 20, 6, 9,  2, 15]
heapq.heapify(l)
heapq.heappush(l,  3)
print l
heapq.heappush(l,  1)
print l
```

```
[2, 4, 3, 9, 20, 15, 6]
[1, 2, 3, 4, 20, 15, 6, 9]
```

heapq.heappop(heap, item): deleting smallest element from heap.

In[]
```
import heapq
l = [4, 20, 6, 9, 2, 15]
heapq.heapify(l)
heapq.heappush(l, 3)
print l
heapq.heappush(l, 1)
print l
print  'poped:',  heapq.heappop(l)
print  'poped:',  heapq.heappop(l)
print l
```

```
[2, 4, 3, 9, 20, 15, 6]
[1, 2, 3, 4, 20, 15, 6, 9]
poped: 1
poped: 2
[3, 4, 6, 9, 20, 15]
```

heapq.heapreplace(heap, item): pops smallest element from heap and pushes new element

In[]
```
import heapq
l = [4, 20, 6, 9, 3, 15]
heapq.heapify(l)
heapq.heapreplace(l, 2)
print l
```

```
[2, 4, 6, 9, 20, 15]
```

heapq.heappushpop(heap, item): pushes new item then pops smallest from the heap.

In[]
```
import heapq
l = [4, 20, 6, 9, 3, 15]
heapq.heapify(l)
heapq.heappushpop(l, 2)
print l
```

```
[3, 4, 6, 9, 20, 15]
```

heap.nsmallest(n, iterable[, key]): returns a list with n smallest elements from the list (dataset) defined, and key if provided.

In[]
```
import heapq
l = [4, 20, 6, 9, 3, 15]
heapq.heapify(l)
print heapq.nsmallest(4, l)
```

```
[3, 4, 6, 9]
```

heap.nlargest(n, iterable[, key]): returns a list with n largest elements from the list (dataset) defined, and key if provided

In[]
```
import heapq
l = [4, 20, 6, 9, 3, 15]
heapq.heapify(l)
print heapq.nlargest(4, l)
```

```
[20, 15, 9, 6]
```

Use-Cases:

1. When we have streaming data, and always want to maintain n-largest or n-smallest, heapq is is used.
2. This is also used as priority queue, which is very popular in operating system process scheduling.
3. Widely used in Operating System, Compiler and Interpreters for memory management.

Interview Questions

1. tuple vs list?
2. Reversing a list, tuple and string?
3. list append vs extend ?
4. Internal data structure of a a list ?
5. Output (5,) * 3?
6. What is tuple packing and unpacking?
7. similarities between str and tuple?
8. Sorting algorithm used in list.sort() method?
9. How do you check a string is pallindrome or not ?
10. Two strings are anagarams or not ?
11. How do you remove duplicates in a list?
12. find the word with most number of occurances in a large list of 1 billion words?
13. What is the data structures which is used for caching?
14. Explain how defaultdict works?
15. When do you use heapq?
16. Is set is hashable?
17. When do you use a dict?
18. What is the pupose of OrderedDict?
19. print all the pairs from the list which makes sum n
20. Print all the anagrams from a text file.
21. When do you use FrozenSet()?
22. When do you use tuple, instead of a list?
23. Check the given array is sorted in ascending order or not.

Exercises

Program 1: A list is supposed to contain first n natural numbers in it. But one number is missing in the list. so , we have n-1 elements in the list. How do you find missing element.

Program 2: How do you check two lists are having same set of values and same number of values.

Program 3: A list of email ids(strings) given, and assume there is a third-party function _send_email(email*id, msg)* is available. 'send_email()' takes destination email address and email message. Example: send_email('john@abc.com', 'Welcome to my birthday!') sends message 'Welcome to my birthday!' to 'john@abc.com'.

1. Send an invitation message, to all the email ids in the list.
2. Do not send message to email ids having hotmail.com in it

Program 4:

- Below are the customer ids and deposites done by the customers in a typical day of a bank.

> 1. Find how many customers, acutally deposited the money.
> 2. Who deposited maximum number of times.
> 3. Who deposited maximum sum of deposites.
> 4. Who deposited max amount in a single transaction.
> 5. Customer Id - and list of deposites done by each cutomer.
> 6. Total balance at the end of the day

```
trans = [(1237, 87522),
 (1234, 1000),
 (1236, 6754),
 (1234, 200),
 (1236, 1700),
 (1234, 400),
 (1234, 600),
 (1236, 788),
 (1234, 500),
 (1237, 126653)]
```

Solution

In[]
```
from collections import defaultdict
trans = [(1237, 87522),
  (1234, 1000),
  (1236, 6754),
  (1234, 200),
  (1236, 1700),
  (1234, 400),
  (1234, 600),
  (1236, 788),
  (1234, 500),
  (1237, 126653)]

ledger = defaultdict(list)
total_balance = 0

for cust_id, amount in trans:
    ledger[cust_id].append(amount)
    total_balance += amount

print 'Number of customers', len(ledger)
custid, amounts = max(ledger.items(), key=lambda x:len(x[1]))
print "Customer who did max number of txns is {} and the txns count is {}
".format(custid, len(amounts))
custid, amounts = max(ledger.items(), key=lambda x:sum(x[1]))
print "Customer who did max sum of the amount is {} and the amount is {}"
.format(custid, sum(amounts))
custid, amounts = max(ledger.items(), key=lambda x:max(x[1]))
print "Customer who did max desposit in a single txn {} and the amount is
{}".format(custid, max(amounts))
print 'Total Balance: ', total_balance
```

```
Number of customers 3
Customer who did max number of txns is 1234 and the txns count is 5
Customer who did max sum of the amount is 1237 and the amount is 214175
Customer who did max desposit in a single txn 1237 and the amount is 126
653
Total Balance:  226117
```

In[]
```
ledger.items()
```

Output:
```
[(1234, [1000, 200, 400, 600, 500]),
 (1236, [6754, 1700, 788]),
 (1237, [87522, 126653])]
```

Program 5:

- List of votes and contestant's age are given.

> 1. Find the contestant who won the elections.
> 2. If there is a tie, older contenstant wins.

```
votes = ['Kenny',    'Amanda', 'John',      'Vicky', 'Alex',
         'Amanda',   'John',   'Alex',      'Kenny', 'Vicky',
         'Charles', 'Alex',    'Kenny',     'Eric',  'Charles',
         'Eric',     'Laura',  'Michelle', 'Eric',   'Vicky']

age = {'Kenny': 61,
       'Amanda': 54,
       'Alex': 79,
       'John': 80,
       'Vicky': 34,
       'Eric': 50,
       'Laura': 55,
       'Michelle': 42,
       'Charles': 70}
```

In[]
```
from collections import Counter, defaultdict

votes = ['Kenny',    'Amanda', 'John',      'Vicky', 'Alex',
         'Amanda',   'John',   'Alex',      'Kenny', 'Vicky',
         'Charles', 'Alex',    'Kenny',     'Eric',  'Charles',
         'Eric',     'Laura',  'Michelle', 'Eric',   'Vicky']

age = {'Kenny': 61,
       'Amanda': 54,
       'Alex': 79,
       'John': 80,
       'Vicky': 34,
       'Eric': 50,
       'Laura': 55,
       'Michelle': 42,
       'Charles': 70}

name_count = Counter(votes)

print name_count
print
count_names = defaultdict(list)

for name, count in name_count.items():
    count_names[count].append(name)
print count_names
print
count, candidates = max(count_names.items())

winner = max(candidates, key=lambda name:age[name])

print 'Winner is : ', winner
```

```
Counter({'Alex': 3, 'Kenny': 3, 'Vicky': 3, 'Eric': 3, 'John': 2, 'Charl
es': 2, 'Amanda': 2, 'Michelle': 1, 'Laura': 1})

defaultdict(<type 'list'>, {1: ['Michelle', 'Laura'], 2: ['John', 'Charl
es', 'Amanda'], 3: ['Alex', 'Kenny', 'Vicky', 'Eric']})

Winner is :  Alex
```

Program 3: Check the given two strings are anagrams or not

In[]
```
from collections import Counter

s1 = 'aacbb'
s2 = 'aabcb'

if Counter(s1) == Counter(s2):
    print 'Anagrams'
else:
    print 'Not Anagrams'
```

```
Anagrams
```

Summary

1. list is dynamically resizable array, also called as vector in other programming languages.
2. tupele is immutable and hashable
3. Set doesn't allow duplicates
4. deque retains only last n elements
5. Using we can heapq efficiently retrieve nsmallest elements in streaming data
6. OrderedDict maintains insertion order
7. defaultdict has a default zero value for every key