1 Collection Types

2 Using Other Code
- Modules

3 Problems

Collection Types

- Lists - Fixed ordered Elements
- Tuples -
- Sets - Not in order
- Dictionaries / Dicts - Key Value Pair

Common Methods

- `len(s)` - Give the number of items
- `s.clear()` - Empties the collector

Lists Methods

- Insertion - `list.append(x)` and `list.insert(x, i)`
- Deletion - `list.remove(x)` and `del list[i]`
- Concatenation - `list1 += list2` and list1.extend(list2)
- Membership testing
  - `elem in list` or `elem not in list`
  - `list.index(x)` - returns index
  - `list.count(x)` - # of occurances
- Reverse - `list.reverse()`
- Sort - `list.sort()`

List Slicing

With the slicing operator, we can have the sublists of an existing list.

- list(i:) - The sublist from i till the end
- list(i:j) - Sublist from i till j
- list(i:j:k) - Takes every k^{th} step

```
> numbers = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
> numbers(2:8)
(2, 3, 4, 5, 6, 7)
> numbers(2:8:2)
(2, 4, 6)
> numbers(8:2:-1)
(8, 7, 6, 5, 4, 3)
```

Xmas and Halloween

Why do programmers mix up Christmas and Halloween (Oct 31)?

Tuples

Immutable lists

More efficient than lists

Coordinates

Grouping and Sequence Unpacking

```
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
>>>
... u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
>>> t = 'hello',
>>> t
('hello',)
>>> t = 12, 'x', 'World'
>>> x, y, z = t
```

Sets

- Unordered collection
- One element - only one time
- In comparison with lists
  1. Eliminating double elements
  2. Efficient membership testing
- Only immutable elements. Lists cannot be elements of a set.

Set Operations

- Addition : `set.add(x)`
- Removal : `set.remove(x)` or `set.discard(y)`
- A set from set : `set1.update(set2)`
- Subset / Superset testing `set1.issubset(set2)` and `set1.issuperset(set2)`
- The other operations
  1. Union `set1.union(set2)`
  2. Intersection `set1.intersection(set2)`
  3. Difference `set1.difference(set2)`
  4. Symmetric Difference `set1.symmetric_difference(set2)`

```
1
2 >>> set1.difference_update(set2)
3 set1 -= set2
4 >>> set1.symmetric_difference_update(set2)
5 set1 ^= set2
6 >>> set1.intersection_update(set2)
7 set1 &= set2
```

Exercises From Last Week

- Fibonacci - 4 ways
- n^{th} root
- *atoi* and *itoa*
- Word Combinations

```
1  def fibrec(n):
2      if n < 3:
3          return (n-1)
4      return fibrec(n-1) + fibrec(n-2)
5
6  def fibiter(n):
7      a, b = 0, 1
8      while n > 2:
9          a += b
10         b += a
11         n -= 2
12     if n == 1:
13         return a
14     return b
15
16
17
```

```
18
19 def fibrectab(ft, n):
20     if len(ft) != n+1:
21         for i in range(1, n+2):
22             ft.append(-1)
23         ft[1] = 0
24         ft[2] = 1
25
26     if ft[n] == -1:
27         ft[n] = fibrectab(ft, n-1) + fibrectab(ft,
28
29     return ft[n]
30
31 fibtable = []
32 for i in range(1, 4):
33     x = int(raw_input("Give me num: "))
34     print "Rec: ", fibrec(x),
```

```
35      print "Iter: ", fibiter(x),
36      print "Table: ", fibrectab(fibtable , x)
37
38  (sadanand@lxmayr10 % code)python fibonacci.py
39  Give me num: 12
40  Rec:  89 Iter:  89 Table:  89
41  Give me num: 23
42  Rec:  17711 Iter:  17711 Table:  17711
43  Give me num: 16
44  Rec:  610 Iter:  610 Table:  610
45  (sadanand@lxmayr10 % code)
```

Dictionaries

- Very important data structure
- Key Value Pair
- Find out a value for the given key
- Example - Telephone book

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'sape': 4139, 'guido': 4127, 'jack': 4098}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'guido': 4127, 'irv': 4127, 'jack': 4098}
>>> tel.keys()
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
```

Dictionary Methods

- Membership testing (if a key is in the dict)
  1. `key in dict`
  2. `dict.has_key(key)`
- Deleting a key value pair
  1. `del dict[key]` (returns nothing)
  2. `dict.pop(key)` (returns the value)
- Adding a keyvalue pair
  `dict['key'] = value`

More with dicts

- Updating a dict
  `dict1.update(dict2`
- Listing the keys/values
  1. `dict.keys()`
  2. `dict.values()`

  Be careful that, the order of values/keys being output (as a list) is not deterministic At the same time, the order of keys and values would correspond with each other

```
(sadanand@lxmayr10 % code)python
Python 2.6.1 (r261:67515, Jan 20 2009, 08:31:22)
>>> tlist = ((i, chr(65+i*i%19))
                    for i in range(1,12))
>>> tlist
((1, 'B'), (2, 'E'), (3, 'J'), (4, 'Q'),
(5, 'G'), (6, 'R'), (7, 'L'), (8, 'H'),
(9, 'F'), (10, 'F'), (11, 'H'))
>>> dict(tlist)
{1: 'B', 2: 'E', 3: 'J', 4: 'Q', 5: 'G',
6: 'R', 7: 'L', 8: 'H', 9: 'F',
10: 'F', 11: 'H'}
>>>
```

What to choose

- Fixed order and necessity to change the elements : Lists
- Fixed order but not necessary to change : Tuples (efficient)
- Any order, Mutable : Sets
- Two lists of corresponding values : Dicts

Dead People

Only DEAD people can understand hex. But then you learn it and teach me too. But that makes us all DEAF. Why? How?

Something Useful

- Range: `range(i, j)` - generates all numbers from i to j
- `range(i, j, k)` - takes k as the step

`seq` in **bash**

```
$for i in ‘seq 1 10‘
>do
>print $i
>done
```

Combination Generator

For each character in the word, take that out and then make all the combinations of the rest of word, and append the initial character to each of the combinations.
Make them recursively.

```
"""This function, on accepting a list
of characters and a number, generates
all the combinations of the characters
in the list whose length is specified
by the number, and returns the list of
combinations"""
```

```
1
2  def combinations(set, length):
3
4      if length < 1:
5      return ();
6
7      if length == 1:
8      return set;
9
10     combis = ();
```

```
11
12      for x in set:
13          tmpset = set(0:);
14          tmpset.remove(x);
15          scombis = combinations(tmpset, length - 1);
16          for y in scombis:
17              combis.append(x+y);
18
19      combis.sort();
20      return combis;
21
22 comb = combinations(('a', 's', 'd', 'f'), 3);
23
24 print "The combinations are: ", comb;
```

Fast Fib

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$$

```
1
2
3  def dotprod(row, col):
4      ret = 0
5
6      for i in range(len(row)):
7          ret += row(i) * col(i)
8
9      return ret
10
11 def rearrange(B):
12     R = ()
13     for i in range(len(B(0))):
14         col = ( x(i) for x in B)
15         R.append(col)
16     return R
17
```

```
18
19
20 def matmul(A, B):
21     C = []
22     R = rearrange(B)
23     for row in A:
24         Crow = []
25         for col in R:
26             k = dotprod(row, col)
27             Crow.append(k)
28         C.append(Crow)
29     return C
30
31
32
33
34
```

```
35
36
37
38 def matpow(A, n):
39     if n < 1:
40         print "We'll think about it, Aha!"
41         return ((),())
42     if n == 1:
43         return A
44     if n == 2:
45         return matmul(A, A)
46     if n % 2 == 0:
47         halfpow = matpow(A, n/2)
48         return matmul(halfpow, halfpow)
49     else:
50         return matmul(matpow(A, n-1), A)
51
```

```
52
53
54
55 def printmat(mat):
56     print '-------'
57     for row in mat:
58         print row
59     print '-------'
60
61
62 def specialfib(n):
63     specmat = ((0,1),(1,1))
64     powered = matpow(specmat, n)
65     fibomat = matmul(powered, ((0),(1)))
66     print fibomat(0)(0)
67     printmat(fibomat)
68
```

```
69 specialfib(18);
```

Import Statement

- No one can write all the code he/she needs.
- No need to re-invent the wheel
- Use `import` statement of Python
- Equivalent of `#include` of **C**

```
>>> import math
>>> math.pow(5, 2)
25.0
>>> math.pow(2, 5)
32.0
>>> from math import pow
>>> pow(3, 4)
81.0
```

What are they?

- Modules are collections of classes or functions or definitions or simply python code. (Python files are)
- They are re-usable. One could use the codes from a different module
- There are some standard Python modules (eg. `sys, math`)
- To use the code/module, one has to use the keyword `import` and also should know what exactly is available in the module.

```
1
2  def fib(n):
3      a, b = 0, 1
4      while b < n:
5          print b,
6          a, b = b, a+b
7
8  def fib2(n):
9      result = []
10     a, b = 0, 1
11     while b < n:
12         result.append(b)
13         a, b = b, a+b
14     return result
```

```
>>> import fibo
>>> fibo.fib(1000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
(1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89)
>>> fibo.__name__
'fibo'
>>> fib = fibo.fib
>>> fib(500)
1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Different Ways

- To use the `fibo.py`, one could do straight `import fibo` which imports the module
- One could import functions separately too `from fibo import fib`
- When one tries to import, Python looks for the file in the same directory first, or in the Python path. (usually `/usr/local/lib/python/`)

Running Modules/Files

- One could run the modules straight as scripts (applicable to python files as well) `$$python fibo.py <arguments>`
- When the file is run as above, the `__name__` attribute would be set to `"__main__"`. So a check for the value of the attribute should enable us to choose what to do. (Example follows)
- If the file is made executable, one could also use it as a command `$$./fibo.py <arguments>`

```
1 if __name__ == "__main__":
2     import sys
3     fib(int(sys.argv[1]))
4
5 ..........................
6
7 $ python fibo.py 50
8 1 1 2 3 5 8 13 21 34
```

Dir()

There is a built-in function which lists all the available functions/methods/attributes of a module.
(Example below)

```
>>> import fibo, sys
>>> dir(fibo)
('__name__', 'fib', 'fib2')

>>> dir()
('__builtins__', '__doc__', '__file__',
 '__name__', 'a', 'fib', 'fibo', 'sys')

>>> import __builtin__
>>> dir(__builtin__)
(.....................................)
```

Some nice libraries

`re`	Regular expression operations
`numbers`	Numeric abstract base classes
`math`	Mathematical functions
`cmath`	Functions for complex numbers
`decimal`	Decimal fixed & floating point math
`random`	Generate pseudo-random numbers
`os`	Miscellaneous OS interfaces
`io`	Core tools for streams
`time`	Time access and conversions
`os.path`	Common pathname manipulations

Problems

- Find the Logarithm
- Tower of Hanoi
- Implement Bubblesort and Binary Search
- Number to Word
  1. Read out as in the telephone (reverse too)
  2. Read out the value of the number
- Product of elements

  There is a List $A[n]$ of n integers. You have to create another List *Output* such that $Output[i]$ will be equal to the product of all the elements of A except $A[i]$.