Control Structures

- Branching on different conditions
  `if`
- Looping
  `while, for`

IF

```
if expr:
____ statement1
else:
____ statement2
```

Statement1 is executed if `expr` is true.

What is not TRUE?

False, 0, empty string, empty set, empty list - all are non-true.

All the rest are TRUE.

```
>>> x = int(raw_input("Please enter an integer: "
Please enter an integer: 42
>>> if x < 0:
...         x = 0
...         print 'Negative changed to zero'
... else:
...         print 'Non-negative'
...
```

Switch Case

- No switch case in Python
- Implemented with `if...` `elif...` `elif...` `else`
  (see example)

```
>>> x = int(raw_input("Please enter an integer: "
Please enter an integer: 42
>>> if x < 0:
...         x = 0
...         print 'Negative changed to zero'
... elif x == 0:
...         print 'Zero'
... elif x == 1:
...         print 'Single'
... else:
...         print 'More'
...
```

Where is the else?

```
if expr:
____ if expr2:
________ statement
____ else:
________ statement2
```

```
if expr:
____ if expr2:
________ statement
else:
____ statement2
```

FOR

- Not necessarily Arithmetic progression
- Could iterate on lists, strings etc.

Do NOT modify the list while looping/iterating on a list.
Then how to do?
Make a Slice `for item in mylist[:]`

```
>>>
... a = ('cat', 'window', 'defenestrate')
>>> for x in a:
...         print x, len(x)
...
cat 3
window 6
defenestrate 12
>>>
>>>
>>>
>>>
>>> for x in a[:]:
...     if len(x) > 6: a.insert(0, x)
...
>>> a
('defenestrate', 'cat', 'window', 'defenestrate')
```

While Loops

- The expression is evaluated
- The loop executes as long as it is true.
- Gets out when it becomes false.

```
while expr:
____ statement
```

WHILE ELSE - Unique to Python?

Break and Continue

- Just like in any other programming language
- The `break` statement, like in C, breaks out of the smallest enclosing `for` or `while` loop.
- The `continue` statement, also borrowed from C, continues with the next iteration of the loop.

```
n = 2
while n < 100:
        m = 2
        while m < n:
                if n % m == 0:
                        break
                m += 1
        else:
                print n, 'is a Prime Number'
        n += 1
```

Exercises From Last Week

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

```python
def fibrec(n):
    if n < 3:
        return (n-1)
    return fibrec(n-1) + fibrec(n-2)

def fibiter(n):
    a, b = 0, 1
    while n > 2:
        a += b
        b += a
        n -= 2
    if n == 1:
        return a
    return b




```

```python
def fibrectab(ft, n):
    if len(ft) != n+1:
        for i in range(1, n+2):
            ft.append(-1)
        ft(1) = 0
        ft(2) = 1

    if ft(n) == -1:
        ft(n) = fibrectab(ft, n-1) + fibrectab(ft,

    return ft(n)

fibtable = ()
for i in range(1, 4):
    x = int(raw_input("Give me num: "))
    print "Rec: ", fibrec(x),
    print "Iter: ", fibiter(x),
    print "Table: ", fibrectab(fibtable, x)
```

```
(sadanand@lxmayr10 % code)python fibonacci.py
Give me num: 12
Rec:   89 Iter:   89 Table:   89
Give me num: 23
Rec:   17711 Iter:   17711 Table:   17711
Give me num: 16
Rec:   610 Iter:   610 Table:   610
(sadanand@lxmayr10 % code)
```

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)
```

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)`

```
>>> set1.difference_update(set2)
set1 -= set2
>>> set1.symmetric_difference_update(set2)
set1 ^= set2
>>> set1.intersection_update(set2)
set1 &= set2
```

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"""

def combinations(set, length):

    if length <  1:
    return ();

    if length == 1:
    return set;

    combis = ();

    for x in set:
        tmpset = set(0:);
        tmpset.remove(x);
        scombis = combinations(tmpset, length - 1);
        for y in scombis:
            combis.append(x+y);

    combis.sort();
    return combis;

comb = combinations(('a', 's', 'd', 'f'), 3);

print "The combinations are: ", comb;
```

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

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
```

Fast Fib

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

```python


def dotprod(row, col):
    ret = 0

    for i in range(len(row)):
        ret += row[i] * col[i]

    return ret

def rearrange(B):
    R = []
    for i in range(len(B[0])):
        col = [ x[i] for x in B]
        R.append(col)
    return R


```

```python
def matmul(A, B):
    C = []
    R = rearrange(B)
    for row in A:
        Crow = []
        for col in R:
            k = dotprod(row, col)
            Crow.append(k)
        C.append(Crow)
    return C
```

```
def matpow(A, n):
    if n < 1:
        print "We’ll think about it, Aha!"
        return ((),())
    if n == 1:
        return A
    if n == 2:
        return matmul(A, A)
    if n % 2 == 0:
        halfpow = matpow(A, n/2)
        return matmul(halfpow, halfpow)
    else:
        return matmul(matpow(A, n-1), A)
```

```
def printmat(mat):
    print '--------'
    for row in mat:
        print row
    print '--------'


def specialfib(n):
    specmat = [[0,1],[1,1]]
    powered = matpow(specmat, n)
    fibomat = matmul(powered, [[0],[1]])
    print fibomat[0][0]
    printmat(fibomat)

specialfib(18);
```

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` $A[i]$`.`