Lecture 3

Simple Programming Constructs

if, for, while, etc.
1) def f(x):
    return x.split('(')[1].split(')')[0].strip()

    return x[x.find(')')+1:x.find(')')]

    return x.split()[2].strip(')').strip(')')

    (not quite right)

    a=list(x)
    b=a[a.index(')')+1 :a.index(')')]
    return '""'.join(b)

    return x[8:-15]
Some common misunderstandings

1)
def f(x):
    x="my name, (George Jones) is great"
    return x[x.find("(")+1:x.find(")")]

def f(x):
    print x[x.find("(")+1:x.find(")")]
    return
2) def f(x):
    return list[x].count(',)
    (see anything wrong?)

    return len(x.split(',','))-1

    return x.count(','

    a=str(x)
    b=a.split('','
    return len(b)-1
To perform an action only if some condition is true

```python
>>> def sign(x):
    if x>0 : return 1
    elif x<0 : return -1
    else return 0

>>> f(1000.0)
1
>>> f(-1000.0)
-1
>>> f(0)
0
>>> f("abc")
1
>>> f("-1")
1
```
```python
if

>>> def f(x):
    if not isinstance(x,str) : return “Not a string”
    elif len(s)<5: return “A short string”
    elif len(s)>=5 and len(s)<20 : return “A medium string”
    elif len(s)>=20: return “A long string”

>>> f(“abc”)  
“A short string”
>>> f(“This is a test”)  
“A medium string”
>>> f(5)  
“Not a string”

Could the above function be simplified ?
```
if

exactly equivalent:

```python
>>> def f(x):
    if not isinstance(x, str):
        return "Not a string"
    if len(s) < 5:
        return "A short string"
    if len(s) < 20:
        return "A medium string"
    return "A long string"
```

Useful operators for if:

- `<`, `>`, `==`, `<=`, `>=`, `!=`
- `and`, `or`, `not`, `()`
- `isinstance()`
- `%` (modulus)
try, except, pass

To handle errors in a program gracefully

```python
>>> int("abc")

Traceback (most recent call last):
  File "<pyshell#52>", line 1, in -toplevel-
    int("abc")
ValueError: invalid literal for int(): abc

>>> try: int("abc")     # Try to do something
>>> except: print "an error occurred"    # if an error occurs, do this instead

an error occurred
```
try, except, pass

```python
>>> def f(x):
    if isinstance(x,int) or isinstance(x,float) : return x
    if isinstance(x,str) :
        try: return int(x)
        except: pass
        try: return float(x)
        except: return 0
    try: return len(x)
    except: return 0

>>> f(1)
1
>>> f("1")
1
>>> f("1.0")
1.0
>>> f([1,2,3])
3
>>> f({1:2})
1
```
while

To repeat an action until some condition is met

```python
>>> x=0
>>> y=0
>>> while (x<10):
    # will repeat the next 3 lines (indented the same)
    x+=1
    # until x>=10
    y+=x
    print x,y
```

```
1 1
2 3
3 6
4 10
5 15
6 21
7 28
8 36
9 45
10 55
```
To repeat an action for each item in a list

```python
>>> a=0
>>> for i in [1,2,3]:  # executes a+=i for i=1,2,3
.....  a+=i
>>> print a
6

>>> sum([1,2,3])
6

>>> sum(['a','b','c'])
Traceback (most recent call last):
  File "<pyshell#72>", line 1, in -toplevel-
    sum(['a','b'])
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> a=0
>>> for i in ['a','b','c']:
.....  a+=i

>>> a
"abc"
```
```python
>>> a=0
>>> for i in range(1000): a+=i
>>> a
499500

>>> a=[“Abc”,125.2,100,200,”def”,300,1.0]
>>> for i in a: # finds all integers in a list
    if isinstance(i,int) : print i
    100
    200
    300

>>> b=[]
>>> for i in a:
    if isinstance(i,int) : b.append(i)
>>> b
[100,200,300]
```
A neat trick:

```python
>>> c=[i for i in a if isinstance(i,int)]
>>> c
[100,200,300]
```

Break that down a bit:

```python
>>> a=[1,2,3,4]
>>> a=[i*2 for i in a]
>>> a
[2,4,6,8]
```
Nested Loops

Loops within loops

```python
>>> a=[1,2,3]
>>> b=[“a”,”b”,”c”]
>>> for aa in a:
    for bb in b:
        print bb*aa

a
b
c
aa
bb
cc
aaa
bbb
ccc
```
Nested Loops

Let's figure out what's going on here:

```python
>>> for aa in a:
    for bb in b:
        print bb*aa,"\t",
        print " "
```

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>aa</td>
<td>bb</td>
<td>cc</td>
</tr>
<tr>
<td>aaa</td>
<td>bbb</td>
<td>ccc</td>
</tr>
</tbody>
</table>
Continue and Break

To stop loops, or skip particular steps:

'continue' skips the current cycle of the loop

```python
>>> for i in range(20):
    if i%3==0: continue
    print i,
1 2 4 5 7 8 10 11 13 14 16 17 19
```

'break' stops a loop. 'else' happens only if the loop reaches the end

```python
>>> for i in range(20):
    if i>9 : break
    print i,
else: print “done”
0 1 2 3 4 5 6 7 8 9
```
Homework

Email to me by 10am monday!

1) Multiplication table. Write a program to calculate and print on the screen a multiplication table from 2 to 9

\[
\begin{array}{ccccc}
2 & 3 & 4 & 5 & \ldots \\
2 & 4 & 6 & 8 & 10 \\
3 & 6 & 9 & 12 & 15 \\
4 & 8 & 12 & 16 & 20 \\
\ldots \\
\end{array}
\]

2) Prime numbers. Write a program to find the first prime number (not divisible by anything but 1 and itself) larger than 1,000,000