Introduction to Python

Lecture 2

Arul Lakshminarayan, 28/9/17

Another fibonacci code

```
# Fibonacci Series using while:
# sum of two consecutive integers = next.
a,b=0,1
while b<1000:
    print(b,end=',') #inserts a , and continues horizontally
    a,b=b,a+b
```

```
======== RESTART: /Users/arul/Desktop/Python/Smallcodes/fibonacci2.py ========
1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,
>>>
```

```
# Fibonacci Series using while:
# sum of two consecutive integers = next.
a,b=0,1
while b<1000:
    print(b,end='\n') #What happens? \n enforces line break and this is default
    a,b=b,a+b
```

Modules and importing

```
# Fibonacci numbers module

def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b
    #print()

def fib2(n):   # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

1. **Save as say fibo.py**
2. **import fibo**
3. **fibo.fib(1000)**
4. **fibo.fib2(1000)**

```
>>> import fibo
>>> fibo.fib(1000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(1000)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987]
>>>
```

Modules and importing

```
>>> import fibo
>>> fibo.fib(1000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(1000)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987]
>>>
```

```
#Create a file in the same directory as fibo.py with the
#following content and run it:

import fibo
fibo.fib(100)
print('\n')
print(fibo.fib2(1000))
```

Modules and importing: other formats

```
#Create a file in the same directory as fibo.py with the
#following content and run it:

from fibo import fib
fib(100)
print('\n')
print(fib2(1000))
```

```
1 1 2 3 5 8 13 21 34 55 89

Traceback (most recent call last):
  File "/Users/arul/Desktop/Python/Smallcodes/callfibo2.py", line 7, in <module>
    print(fib2(1000))
NameError: name 'fib2' is not defined
>>>
```

Modules and importing: other formats

```
#Create a file in the same directory as fibo.py with the
#following content and run it:

from fibo import fib, fib2
fib(100)
print('\n')
print(fib2(1000))
```

```
#Create a file in the same directory as fibo.py with the
#following content and run it:

import fibo as f
f.fib(100)
print('\n')
print(f.fib2(1000))
```

```
1 1 2 3 5 8 13 21 34 55 89


[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987]
>>>
```

Importing Modules from other directories

```
>>> import fibo
Traceback (most recent call last):
  File "<pyshell#40>", line 1, in <module>
    import fibo
ModuleNotFoundError: No module named 'fibo'
>>>
```

```
>>> import sys
>>> sys.path
['', '/Users/arul/Documents', '/Library/Frameworks/Python.framework/Versions/3.6/lib/
python36.zip', '/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6', '/Library/
Frameworks/Python.framework/Versions/3.6/lib/python3.6/lib-dynload', '/Library/Frameworks/
Python.framework/Versions/3.6/lib/python3.6/site-packages']
>>> sys.path.append('/Users/arul/Desktop/Python/Smallcodes')
>>> sys.path
['', '/Users/arul/Documents', '/Library/Frameworks/Python.framework/Versions/3.6/lib/
python36.zip', '/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6', '/Library/
Frameworks/Python.framework/Versions/3.6/lib/python3.6/lib-dynload', '/Library/Frameworks/
Python.framework/Versions/3.6/lib/python3.6/site-packages', '/Users/arul/Desktop/Python/
Smallcodes']
>>> import fibo
>>>
```

The importance of “Returning”

```
# Fibonacci numbers module

def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b
    #print()

def fib2(n):   # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

```
>>> fib(100)
1 1 2 3 5 8 13 21 34 55 89
>>> x=fib(100)
1 1 2 3 5 8 13 21 34 55 89
>>> x
>>>
>>> type(fib(100))
1 1 2 3 5 8 13 21 34 55 89 <class 'NoneType'>
>>>
```

```
>>> fib2(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> x=fib2(100)
>>> x
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> type(fib2(100))
<class 'list'>
>>> x[3]
3
>>>
```

Built-in Modules

- Python has an extensive set of built-in modules
- These can be imported when needed
- For scientific work: Numpy, math, cmath, SciPy, SymPy

Exercises

1. **Import math and cmath:**

find

a) **sin(pi/4), cos(pi/8), tan(3 pi/5),cosh(23),cosh(2), sinh(0)**

b) **sin(3+4i), cos(4i), sin(2 pi i), sin(6.2 i), exp(2 pi i)**

c) **find the real an imaginary parts in answers to (b)**

More containers

Python has general “container objects” on which the “in” operator can be used.

We have already come across Lists and Strings.
In addition there is Tuples and Dictionaries.

```
>>> u=[1,3.0,'python']
>>> v=[2.3,9.0,34,'foo1','foo2']
>>> len(u)
>>> len(v)
>>> u+v
>>> u*2
>>> u
>>> v.append('foo3')
>>> v
>>> v.remove(9.0)
>>> v
>>> v.remove(34.0)
>>> v
>>> v.append(foo)
>>>
```

```
>>> u
['a', 3, 7, 'b']
>>> 'a' in u
True
>>> 'a' and 'b' in u
True
>>>
```

List indexing and slicing

u[i], i goes from 0 to len(u)-1

Also u[len(u)-k] = u[-k], k in (0,len(k)]

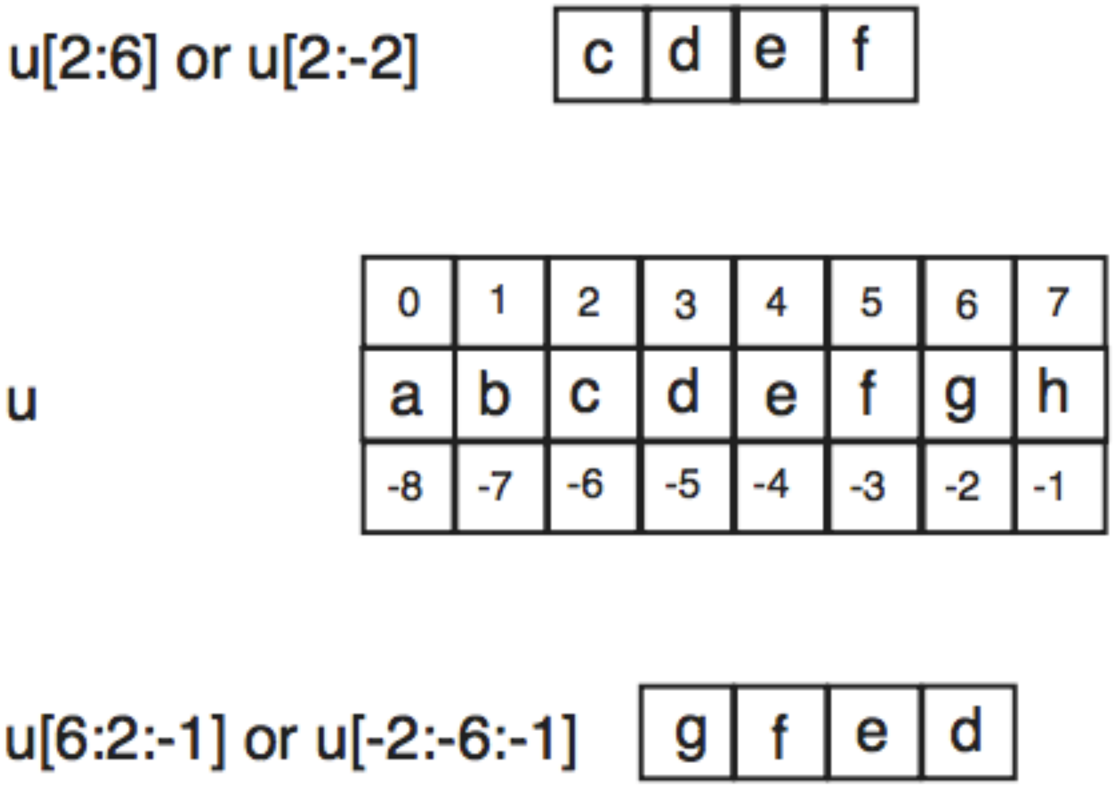
Try out:

```
>>> u=['a', 3,7,'b']
>>> u[0]
>>> u[-1]
>>> u[-2]
>>> u[3]
>>> u[1]
>>> u[-3]
>>> u[-4]
>>> u[-5]
>>>
```

List slicing

Slicing can create a NEW list

u[start:end] a list of length end-start



u[2:6] or u[2:-2]

c	d	e	f

u

0	1	2	3	4	5	6	7
a	b	c	d	e	f	g	h
-8	-7	-6	-5	-4	-3	-2	-1

u[6:2:-1] or u[-2:-6:-1]

g	f	e	d

TRY out:

```
>>> u=[0,1,2,3,4,5,6,7]
>>> su=u[2:4]
>>> su
>>> su[0]=17
>>> su
>>> u
>>> u[2:4]=['3.14','a']
>>> u
```

From J. Stewart: Python for Scientists

List slicing

TRY out:

```
>>> u=[0,1,2,3,4,5,6,7]
>>> su=u[2:4]
>>> su
>>> su[0]=17
>>> su
>>> u
>>> u[2:4]=['3.14','a']
>>> u
```

What is u[: -1] ?

u[1:]?

u[:]

u[0:8]?

u[8] ?

u[0:10]?

u[6:2:-2] ?
u[6:1:-1]?

List mutability

```
>>> x=2
>>> y=x
>>> y=3
>>> x
>>> x=2.0
>>> y=x
>>> y=3.0
>>> x
```

```
>>> x=[1,2,3]
>>> y=x
>>> y=['foo', 4,0]
>>> x
>>> y
>>> z=x
>>> z
[1, 2, 3]
>>> x
[1, 2, 3]
>>> z[0]=1
>>> z[0]='foo'
>>> z
['foo', 2, 3]
>>> x
['foo', 2, 3]
>>>
```

Changing the contents of a list changes the object, leaving its identity unchanged

```
TRY
>>> x=['a',23,90]
>>> z=x
>>> id(x)
>>> id(z)
>>> z=x[:]
>>> id(z)
>>> z[0]='foo'
>>> z
>>> x
>>>
```

```
## SHALLOW copy

>>> l1=[['foo',2],5,6]
>>> l2=l1[:]
>>> l2[1]=50
>>> l2
[['foo', 2], 50, 6]
>>> l1
[['foo', 2], 5, 6]
>>> l2[0][0]='foo1'
>>> l2
[['foo1', 2], 50, 6]
>>> l1
[['foo1', 2], 5, 6]
>>>
```

```
## DEEP copy

>>> import copy
>>> l1=[['foo',2],5,6]
>>> l2=copy.deepcopy(l1)
>>> l2[0][0]='foo1'
>>> l2
[['foo1', 2], 5, 6]
>>> l1
[['foo', 2], 5, 6]
>>>
```

NOTE: list.copy() seems to be still a shallow copy in ver.3.6

Tuples

IMMutable objects that are like lists, but whose elements cannot be changed

```
>>> (a,b,c)=(3,2,4+5j)

>>> (a,b,c)=3,2,4+5j

>>> a,b,c=3,4,’a'
```

```
>>> z=(4,5,6)
>>> type(z)
<class 'tuple'>
>>> x=z
>>> x
(4, 5, 6)
>>> x[0]
4
>>> x[0]=40
Traceback (most recent call last):
  File "<pyshell#277>", line 1, in <module>
    x[0]=40
TypeError: 'tuple' object does not support item assignment
>>> x+z
(4, 5, 6, 4, 5, 6)
>>> 2*z
(4, 5, 6, 4, 5, 6)
>>> id(x)
4331745928
>>> id(z)
4331745928
```

Strings: Immutable containers of alphanumeric characters

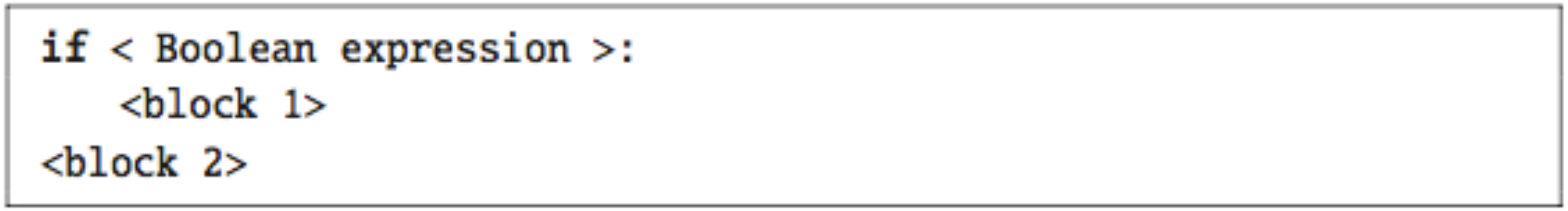
```
>>> s1="It's a bird"
>>> s1[0]
'I'
>>> s1[0]='i'
Traceback (most recent call last):
  File "<pyshell#296>", line 1, in <module>
    s1[0]='i'
TypeError: 'str' object does not support item assignment
>>> s2='"Bird?" she asked'
>>> s2[0]
'"'
>>>
```

Dictionaries— {key1:object1,key2:object2, …}

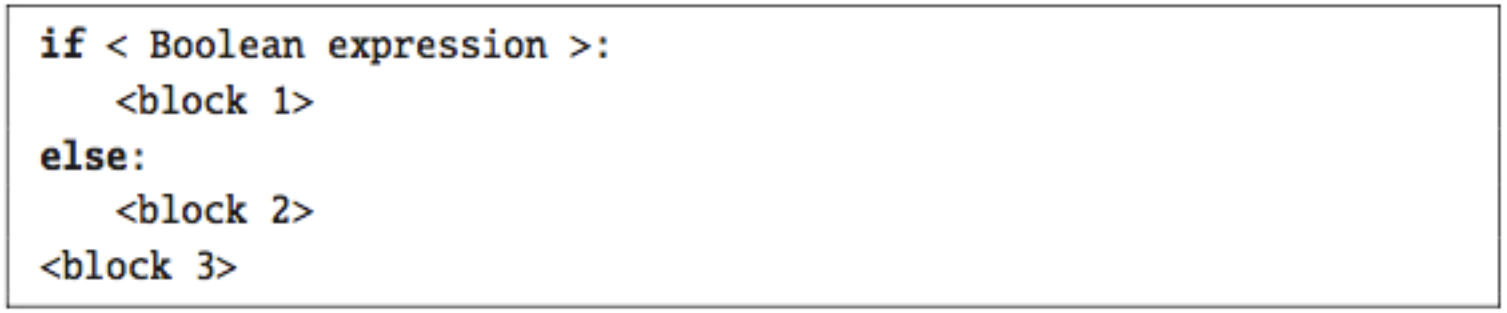
```
>>> param1={'hello':'world','pi':3.14,9:10}
>>> param1['hello']
>>> param1[9]
>>> param1['another']='entry'
>>> param1
>>>
```

Exercise: Find if dictionaries are mutable

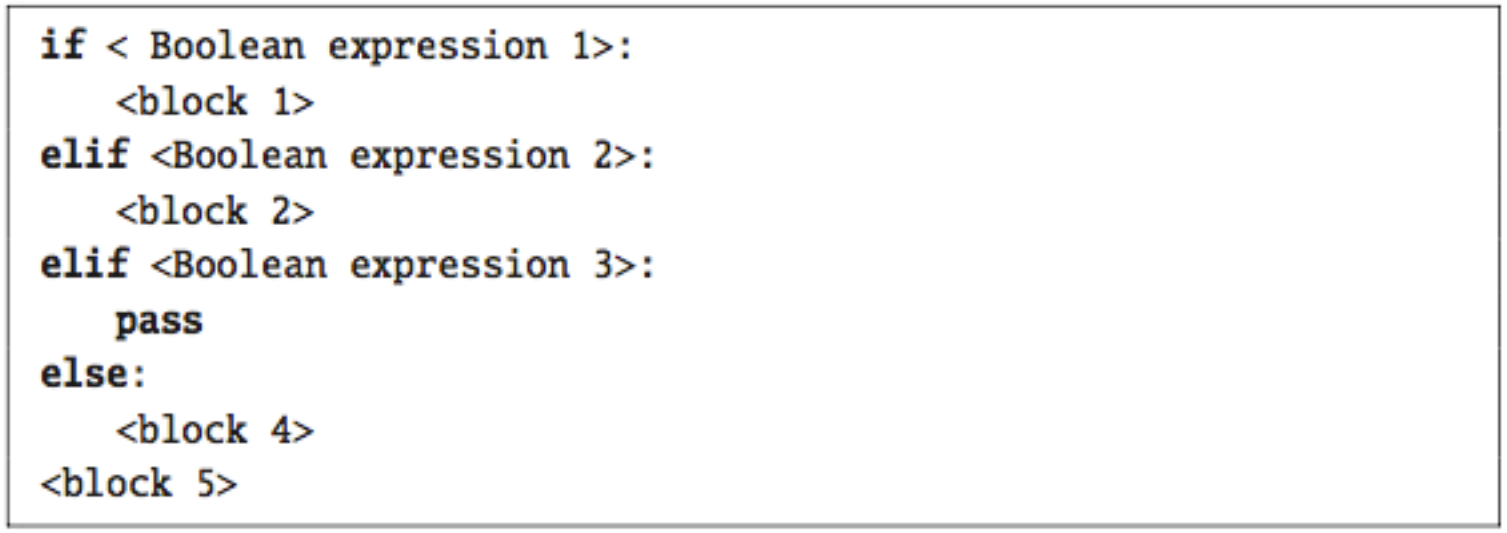
IF, ELIF



```
if < Boolean expression >:
    <block 1>
<block 2>
```



```
if < Boolean expression >:
    <block 1>
else:
    <block 2>
<block 3>
```



```
if < Boolean expression 1>:
    <block 1>
elif <Boolean expression 2>:
    <block 2>
elif <Boolean expression 3>:
    pass
else:
    <block 4>
<block 5>
```

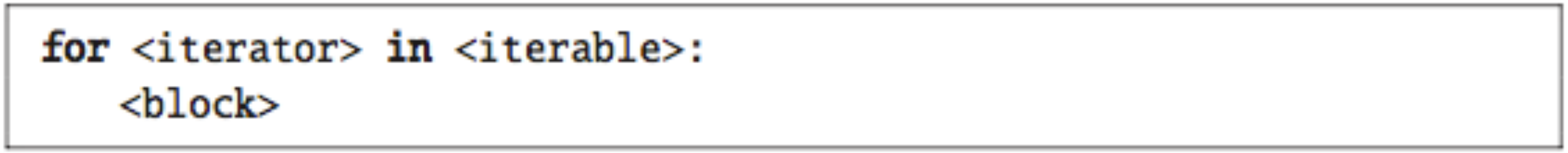
```
x=.4
if(0<x<1):
    print("x lies between 0 and 1")
```

```
x=.4
if(0<x<1):
    print("x lies between 0 and 1")
else:
    print("x lies outside (0,1)")
```

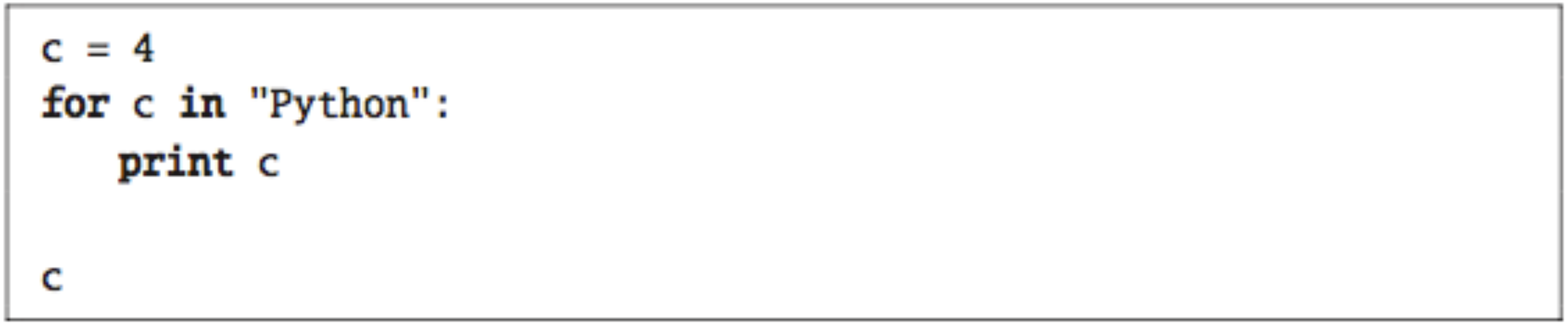
```
print(" in (0,1)") if 0<x<1  else print("outside (0,1)")
```

Loop constructs

- **For** and **While**.
- **For** iterates in iterables such as containers
- **While** checks Boolean values/satisfiability

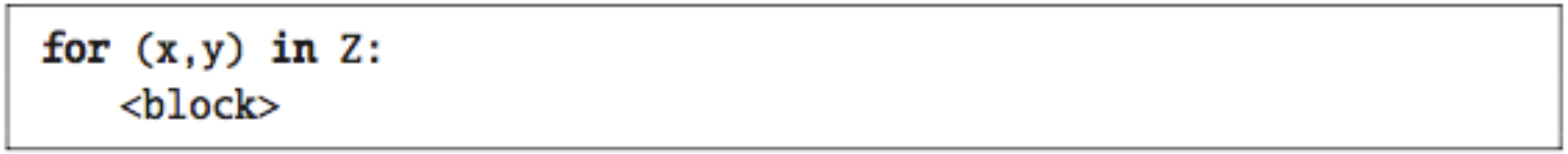


```
for <iterator> in <iterable>:
    <block>
```

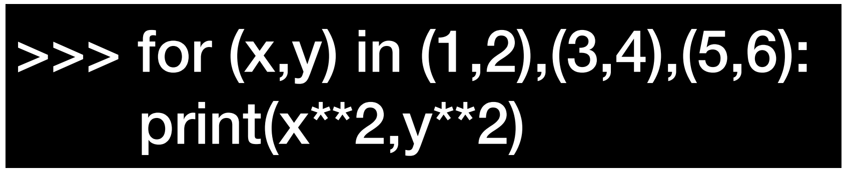


```
c = 4
for c in "Python":
    print c

c
```



```
for (x,y) in Z:
    <block>
```



```
>>> for (x,y) in (1,2),(3,4),(5,6):
        print(x**2,y**2)
```

range function: Warning: slightly different from ver. 2.x (xrange in ver. 2.x)

range(start,end,step) “creates a list” start,start+step,start+2*step, …
till it is less than end.
Not accessible as a list object, till
list(range(start,end,step))
default start=0, default step=1

```
>>> x=range(2,10,3)
>>> x[0]
2
>>> x[1]
5
>>> x[2]
8
>>> x
range(2, 10, 3)
>>> type(x)
<class ‘range'>
```

```
>>> y=list(range(2,10,3))
>>> y
[2, 5, 8]
>>> y=list(range(2,10,2))
>>> y
[2, 4, 6, 8]
>>> type(y)
<class 'list'>
>>>
```

```
TRY
>>> l=[]
>>> for y in range(10):
        l.append(y)
```

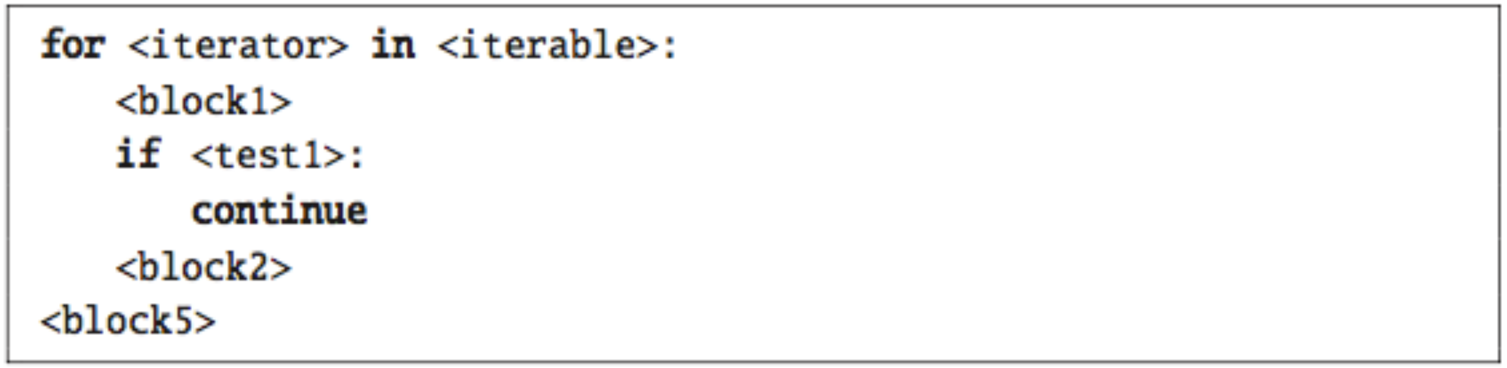
Example: 1-d maps

Logistic map: x -> f(x)=r x(1-x)

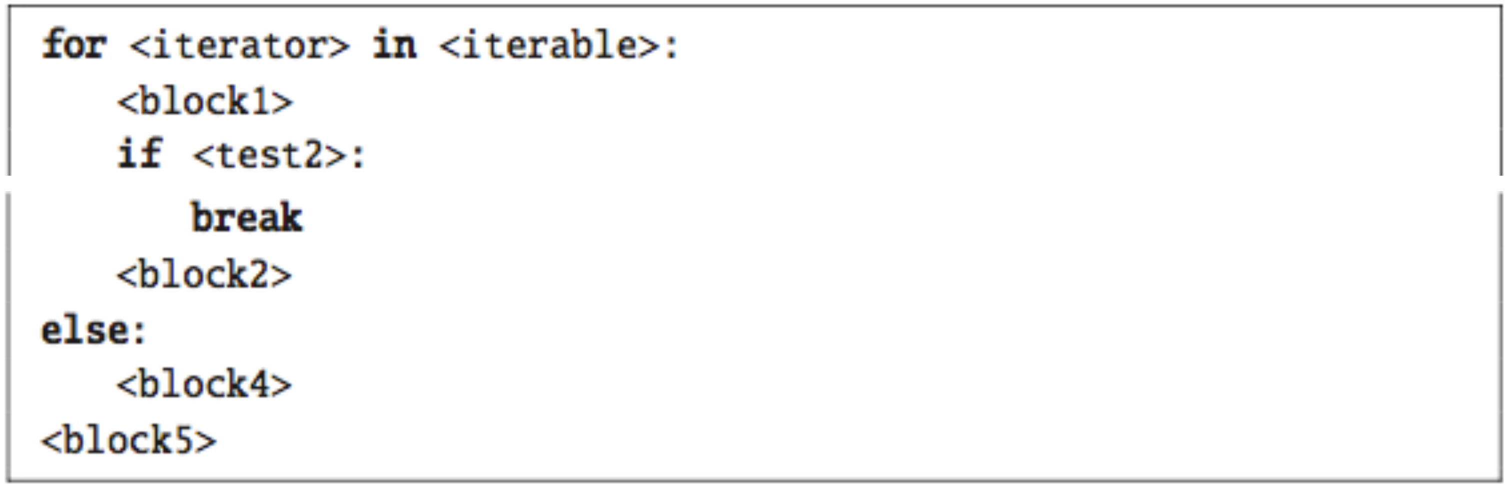
```
def logistic_map(r,x,n):
    for i in range(n+1):
        print(x,end=', ')
        x=r*x*(1-x)
```

Exercise: code a function for the ``doubling map” x -> f(x)=2 x modulo 1

see iterates for 100 times and notice that for arbitrary initial conditions (in (0,1)) they go to 0. When do they do that and why?



```
for <iterator> in <iterable>:
    <block1>
    if <test1>:
        continue
    <block2>
<block5>
```



```
for <iterator> in <iterable>:
    <block1>
    if <test2>:
        break
    <block2>
else:
    <block4>
<block5>
```

If test2 is False, block2 is iterated till last iteration step.
then control passes to else: and block4 is performed before block5.
If test2 is True, iterator is escaped and block5 is evaluated.

Example: Primality

```
def primeq(x):
    for i in range(2,int(x**.5)+1):
        if x%i==0:
            print('False')
            break
    else:
            print('True')
```

Note the indent in the “else” statement.
Test what happens if it is aligned with “if”.

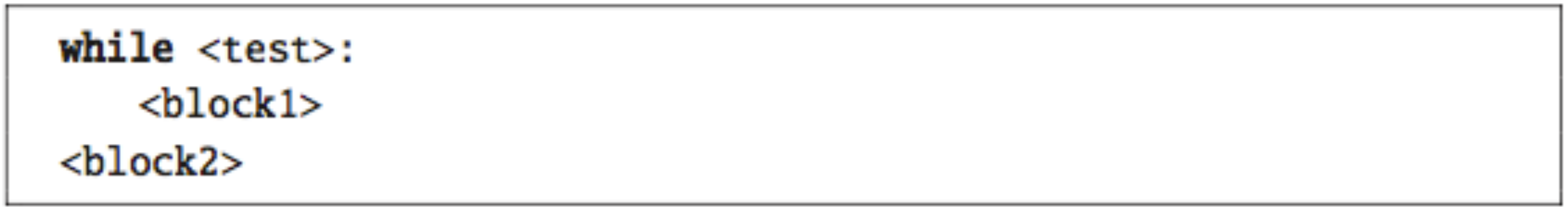
List comprehensions

```
>>> L1=list(range(10))
>>> L1
>>> L2=[x*x for x in L1]
>>> L2
>> L3=[x*x for x in L1 if x%2==0]
>>> L3
```

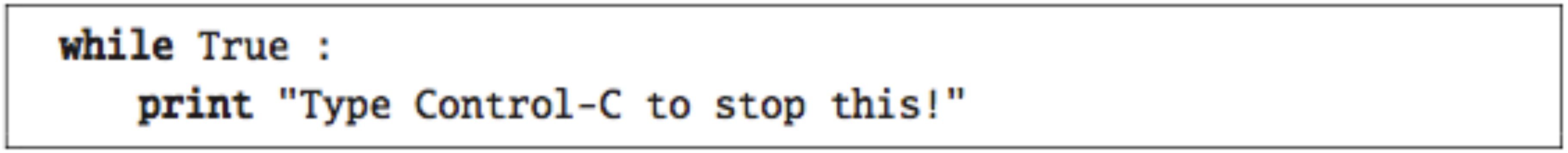
```
>>> lpoints=[(x,x/2) for x in L1]
>>> lpoints
>>> ldist=[(x*x+y*x)**.5 for (x,y) in lpoints]
>>> ldist
>>>
```

TRY:
lpoints1=[(x,y) for x in L1 for y in L2]

While



```
while <test>:
    <block1>
<block2>
```



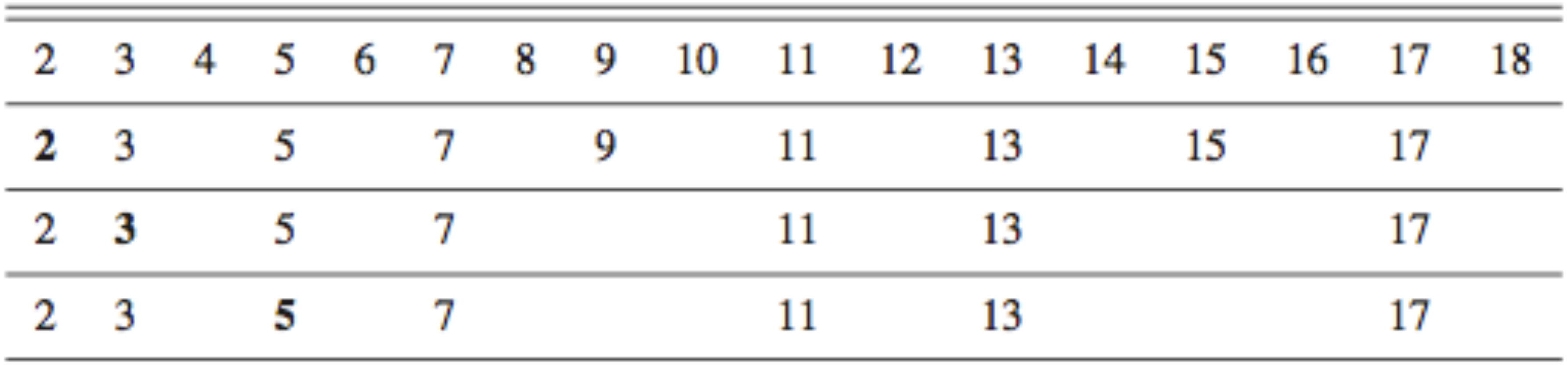
```
while True :
    print "Type Control-C to stop this!"
```

As for “for”, “while” can be interrupted by continue, break, if …

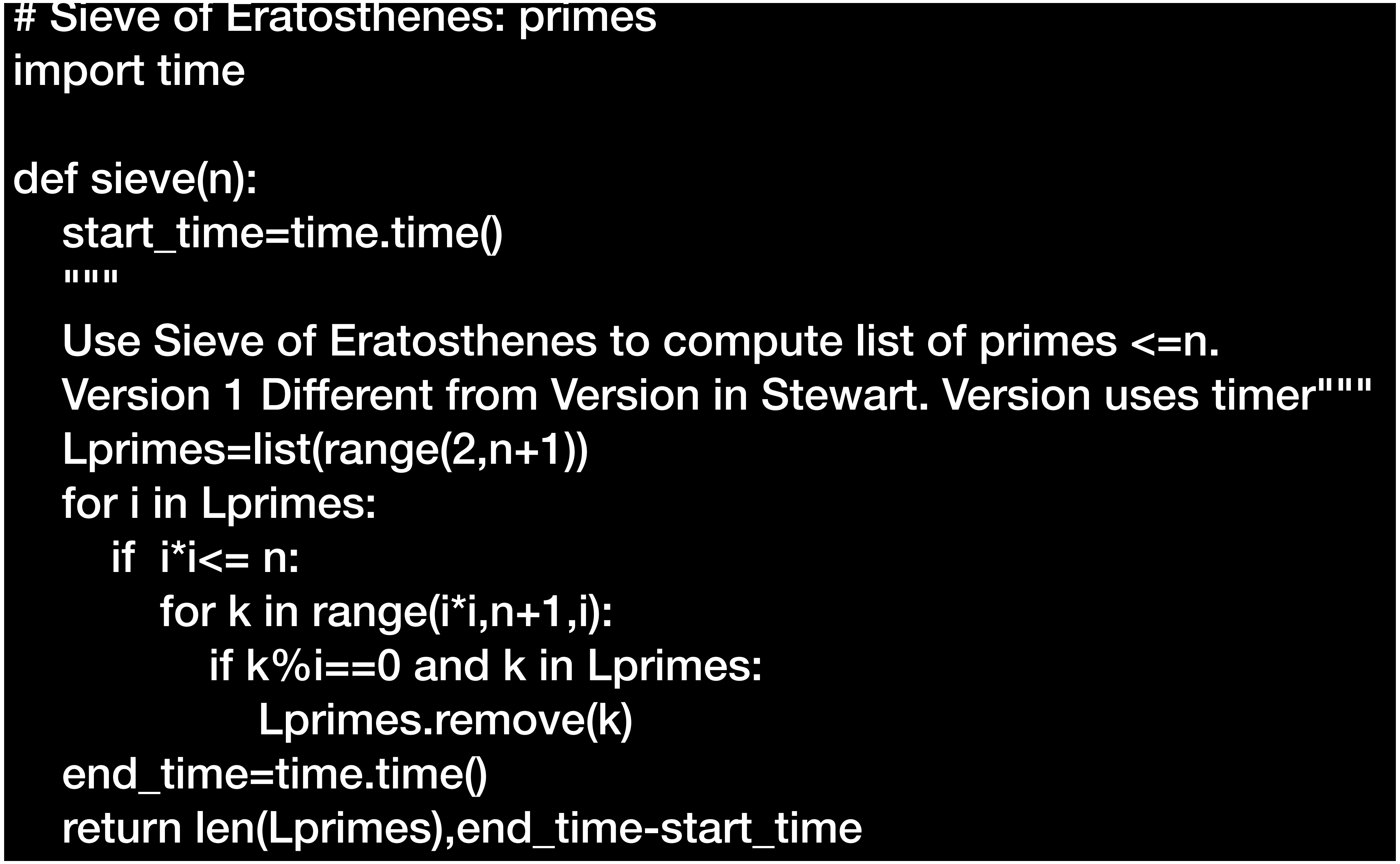
Understand the output of

```
for i in range(10):
    while i in range(5):
        print(i,i**2)
        i+=1
    else:
        print(i,i)
```

Sieve of Eratosthenes:
List of prime numbers



2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
2	3		5		7		9		11		13		15		17	
2	**3**		5		7				11		13				17	
2	3		**5**		7				11		13				17	



```
# Sieve of Eratosthenes: primes
import time

def sieve(n):
    start_time=time.time()
    """
    Use Sieve of Eratosthenes to compute list of primes <=n.
    Version 1 Different from Version in Stewart. Version uses timer"""
    Lprimes=list(range(2,n+1))
    for i in Lprimes:
        if  i*i<= n:
            for k in range(i*i,n+1,i):
                if k%i==0 and k in Lprimes:
                    Lprimes.remove(k)
    end_time=time.time()
    return len(Lprimes),end_time-start_time
```