PYTHON tutorial

LASCON VII 2018

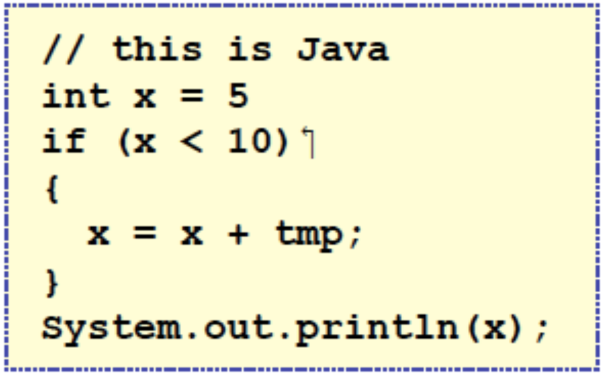
NILTON KAMIJI AND RENAN SHIMOURA

Why python?

- Python is an interpreted object oriented programming language
  - Extensive documentation and huge community
  - Modularity with nice modules for scientific computing/data analysis/visualization
  - Large number of modules for neuroscience
- Goal:
  - Lean enough python so you can start using python for neuroscience simulation and data analysis
- Suggested reading:
  - Muller E, Bednar JA, Diesmann M,Gewaltig M-O, Hines M and Davison AP (2015) Python in neuroscience. Front. Neuroinform. 9:11.doi: 10.3389/fninf.2015.00011

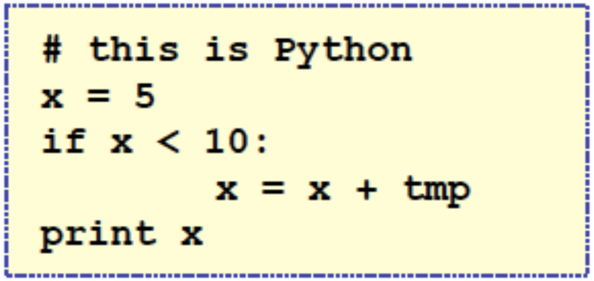
Syntax

- ❑ Interactive interpreter
- ❑ No variable declaration
- ❑ Flexible syntax
  - ➢ No { } for blocks, just indentation
  - ➢ No ( ) for if/while conditions
- ❑ # for comments



```
// this is Java
int x = 5
if (x < 10)
{
  x = x + tmp;
}
System.out.println(x);
```

Java



```
# this is Python
x = 5
if x < 10:
       x = x + tmp
print x
```

Python

”Hello, World”

- C

```
#include <stdio.h>

int main(int argc, char **argv)
{
    print(”Hello, World!\n”);
}
```

- Java

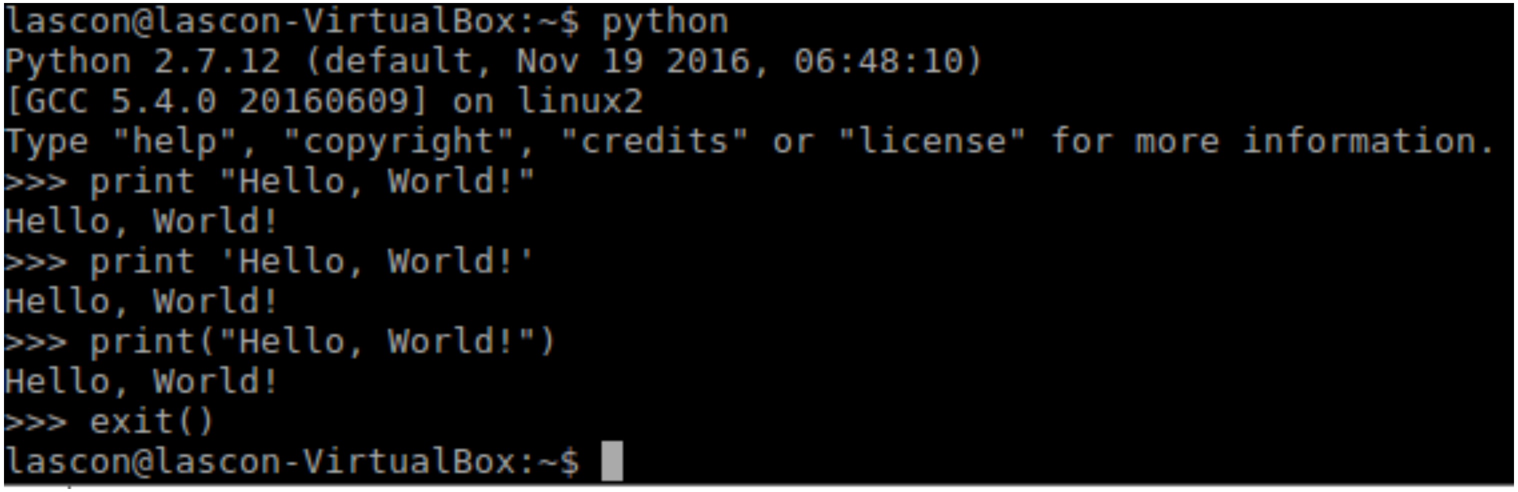
```
public class Hello
{
    public static void main(String argv[])
    {
        System.out.println(”Hello, World!”);
    }
}
```

- Python

```
print ”Hello, World!”
```

”Hello, World”

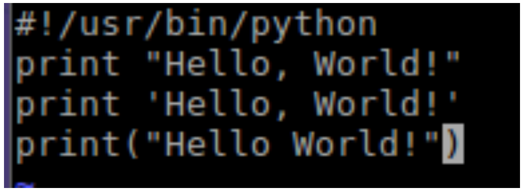
1. In a terminal:



```
lascon@lascon-VirtualBox:~$ python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print "Hello, World!"
Hello, World!
>>> print 'Hello, World!'
Hello, World!
>>> print("Hello, World!")
Hello, World!
>>> exit()
lascon@lascon-VirtualBox:~$ 
```

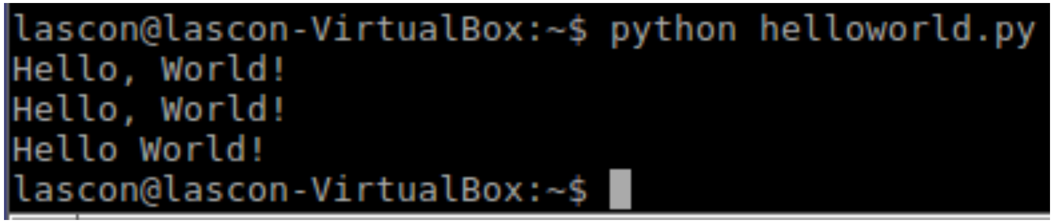
”Hello, World”

2. Create a file *filename.py* that contains



```
#!/usr/bin/python
print "Hello, World!"
print 'Hello, World!'
print("Hello World!")
```

3. Execute as:



```
lascon@lascon-VirtualBox:~$ python helloworld.py
Hello, World!
Hello, World!
Hello World!
lascon@lascon-VirtualBox:~$
```

Variables and Data Types

- No need to declare / specify type
- Just need to assign (initialize)

```
x = 1!
x = ‘hello world’!
print a!
```

- Assignment makes reference between variable and object
  - y = x **does not make a copy** of x
  - y = x makes a **reference** the object x references

Variables and Data Types

- ❑int → 45
- ❑long → 4872987323L
- ❑float → 32.679
- ❑Complex → 3+2j
- ❑str → ‘hello’
- ❑boolean → 0 or 1, True or False
- ❑Can use type(object) to check:
  - ➢ eg. type(3), type(3.0), type(3+2j)
  - ➢ eg. x=4.5, type(x)
- ❑Convert to different types
  - ➢ str(0.5) → '0.5’

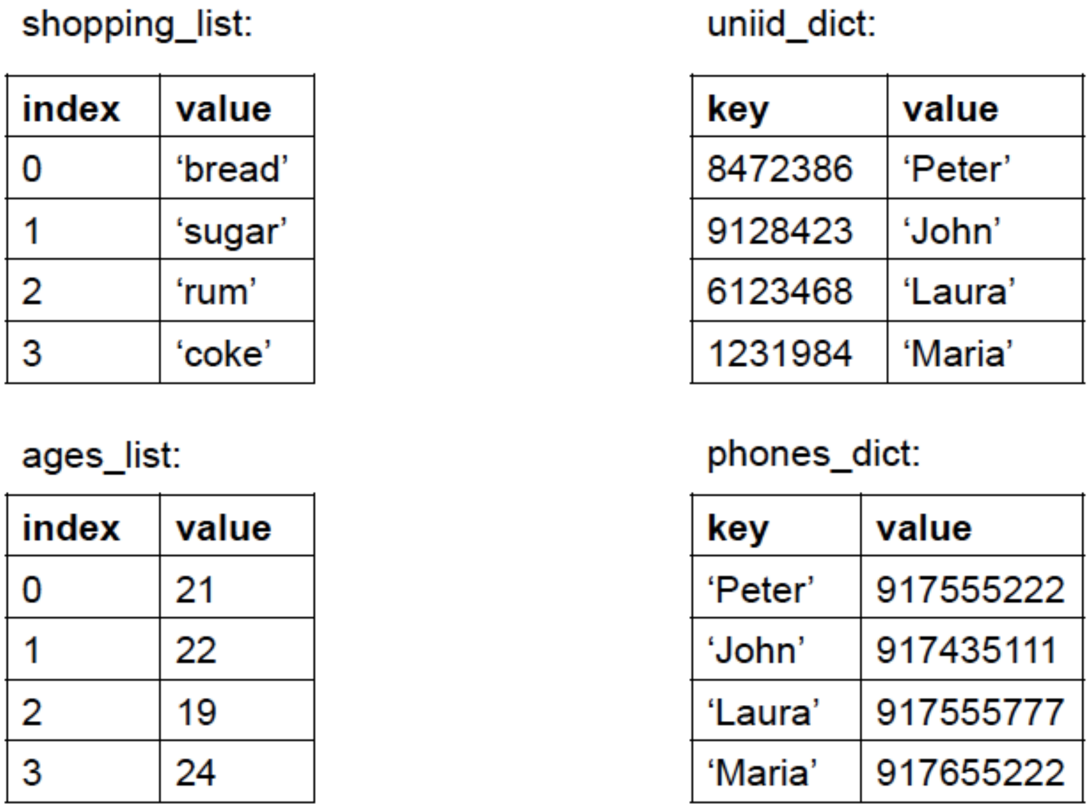
Training suggestion: https://www.codecademy.com/learn/learn-python

Containers

- Can contain variables or other containers
- 3 main types:

List	`l = [ 2, 3, 5, 8 ]`
Tuple (read-only list)	`t = ( 2, 3, 5, 8 )`
Dictionary (key-value map)	`d = {"two": 2, "three": 3}`

Containers



shopping_list:

index	value
0	‘bread’
1	‘sugar’
2	‘rum’
3	‘coke’

uniid_dict:

key	value
8472386	‘Peter’
9128423	‘John’
6123468	‘Laura’
1231984	‘Maria’

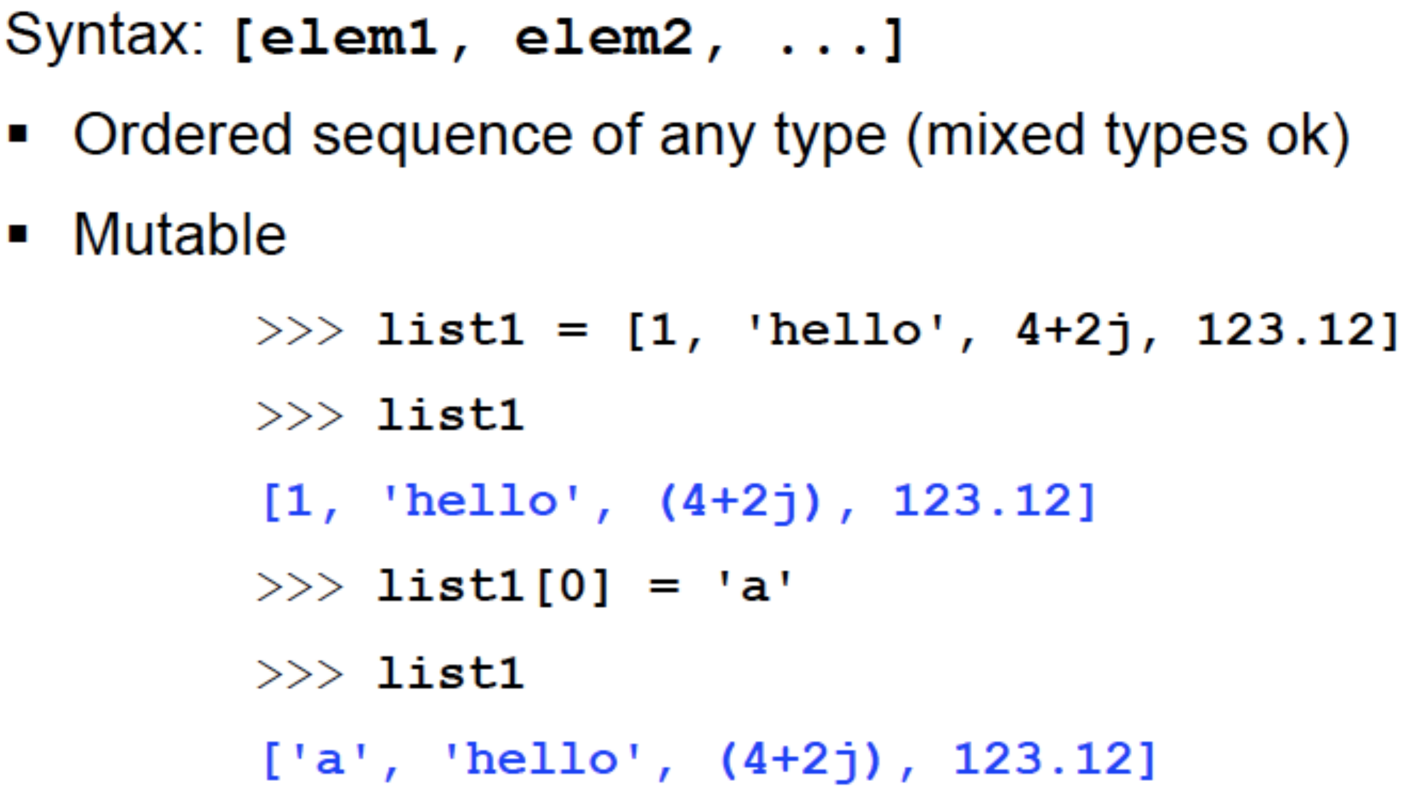
ages_list:

index	value
0	21
1	22
2	19
3	24

phones_dict:

key	value
‘Peter’	917555222
‘John’	917435111
‘Laura’	917555777
‘Maria’	917655222

Containers: List

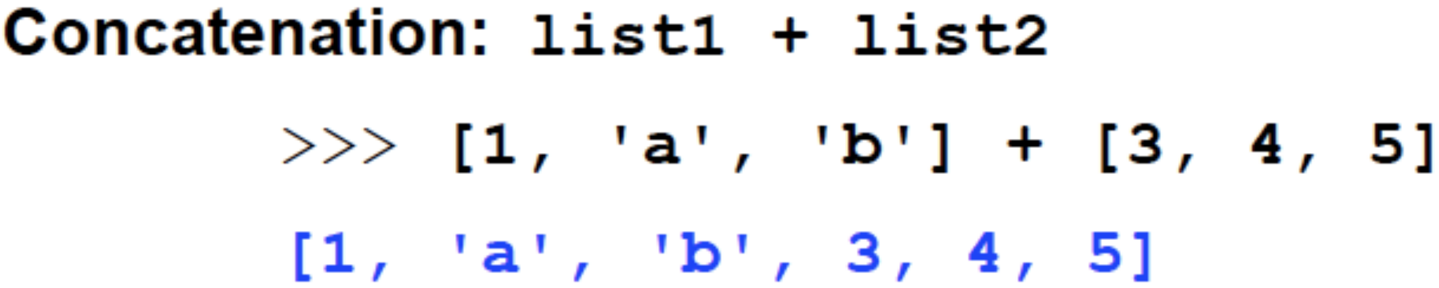


Syntax: `[elem1, elem2, ...]`

- Ordered sequence of any type (mixed types ok)
- Mutable

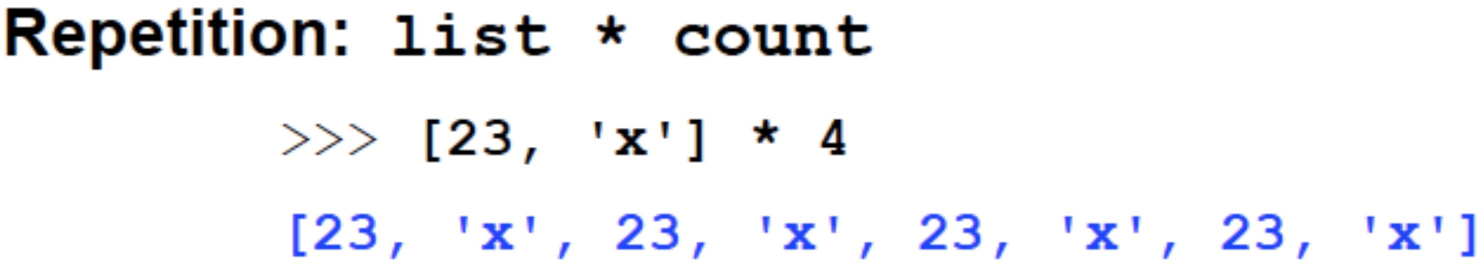
```
>>> list1 = [1, 'hello', 4+2j, 123.12]
>>> list1
[1, 'hello', (4+2j), 123.12]
>>> list1[0] = 'a'
>>> list1
['a', 'hello', (4+2j), 123.12]
```

Containers: List



Concatenation: `list1 + list2`

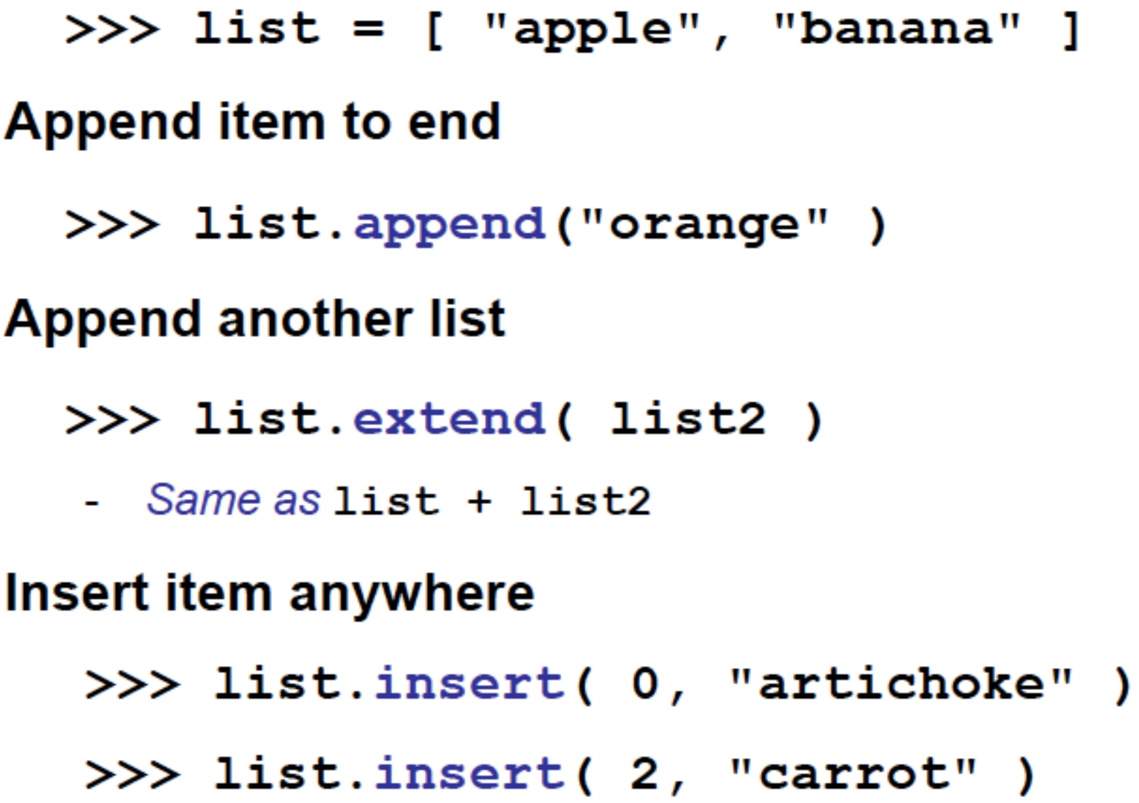
```
>>> [1, 'a', 'b'] + [3, 4, 5]
[1, 'a', 'b', 3, 4, 5]
```



Repetition: `list * count`

```
>>> [23, 'x'] * 4
[23, 'x', 23, 'x', 23, 'x', 23, 'x']
```

Containers: List



```
>>> list = [ "apple", "banana" ]
```

Append item to end

```
>>> list.append("orange" )
```

Append another list

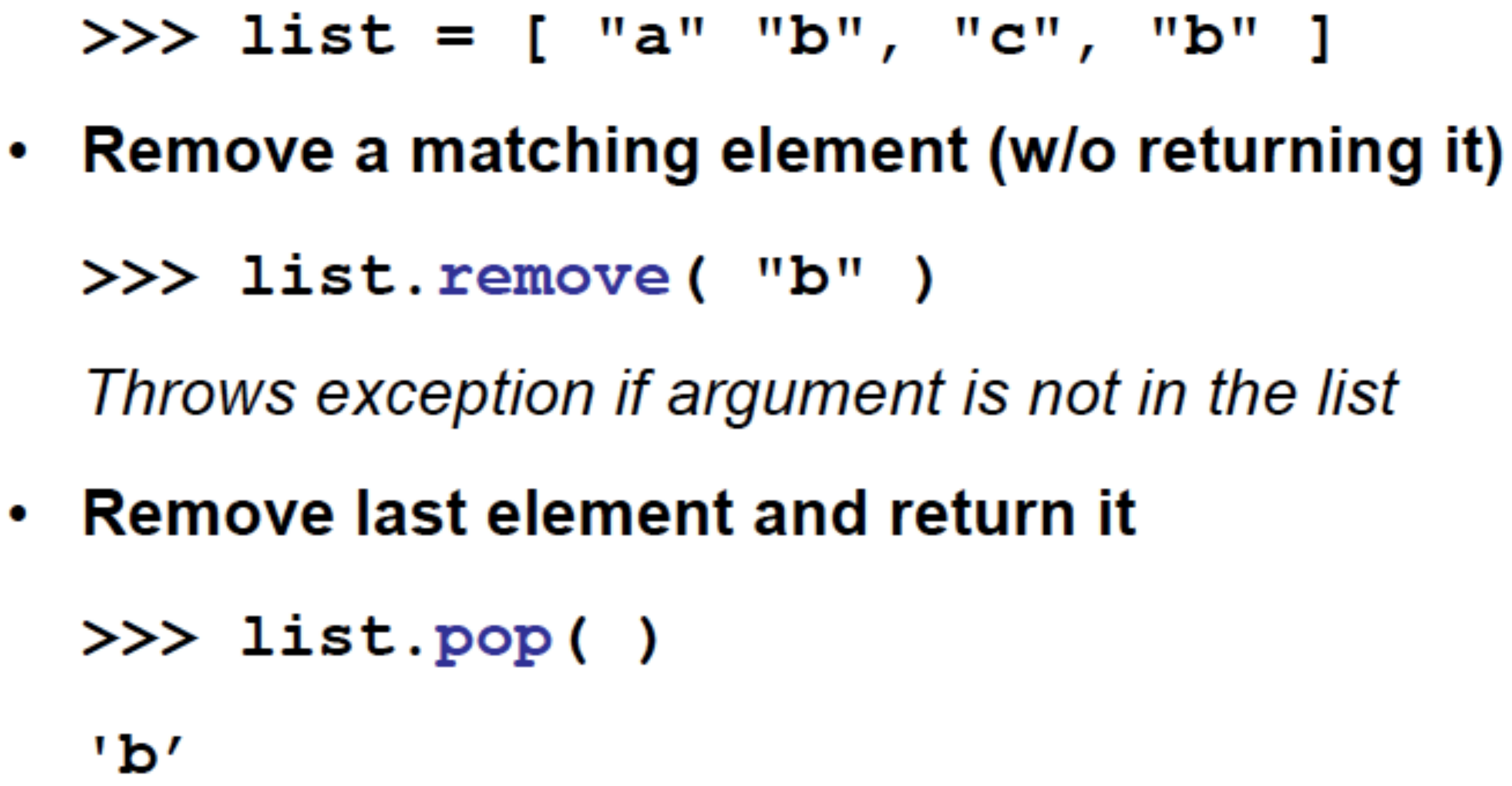
```
>>> list.extend( list2 )
```

- *Same as* `list + list2`

Insert item anywhere

```
>>> list.insert( 0, "artichoke" )

>>> list.insert( 2, "carrot" )
```

Containers: List



```
>>> list = [ "a" "b", "c", "b" ]
```

- **Remove a matching element (w/o returning it)**

```
>>> list.remove( "b" )
```

Throws exception if argument is not in the list

- **Remove last element and return it**

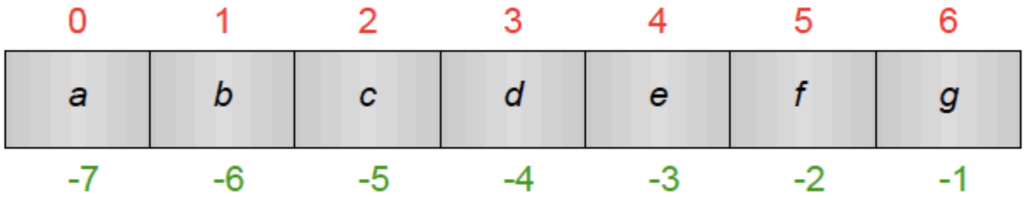
```
>>> list.pop( )
'b’
```

Containers: List

❑ Indexing

Syntax: **`list[n]`**

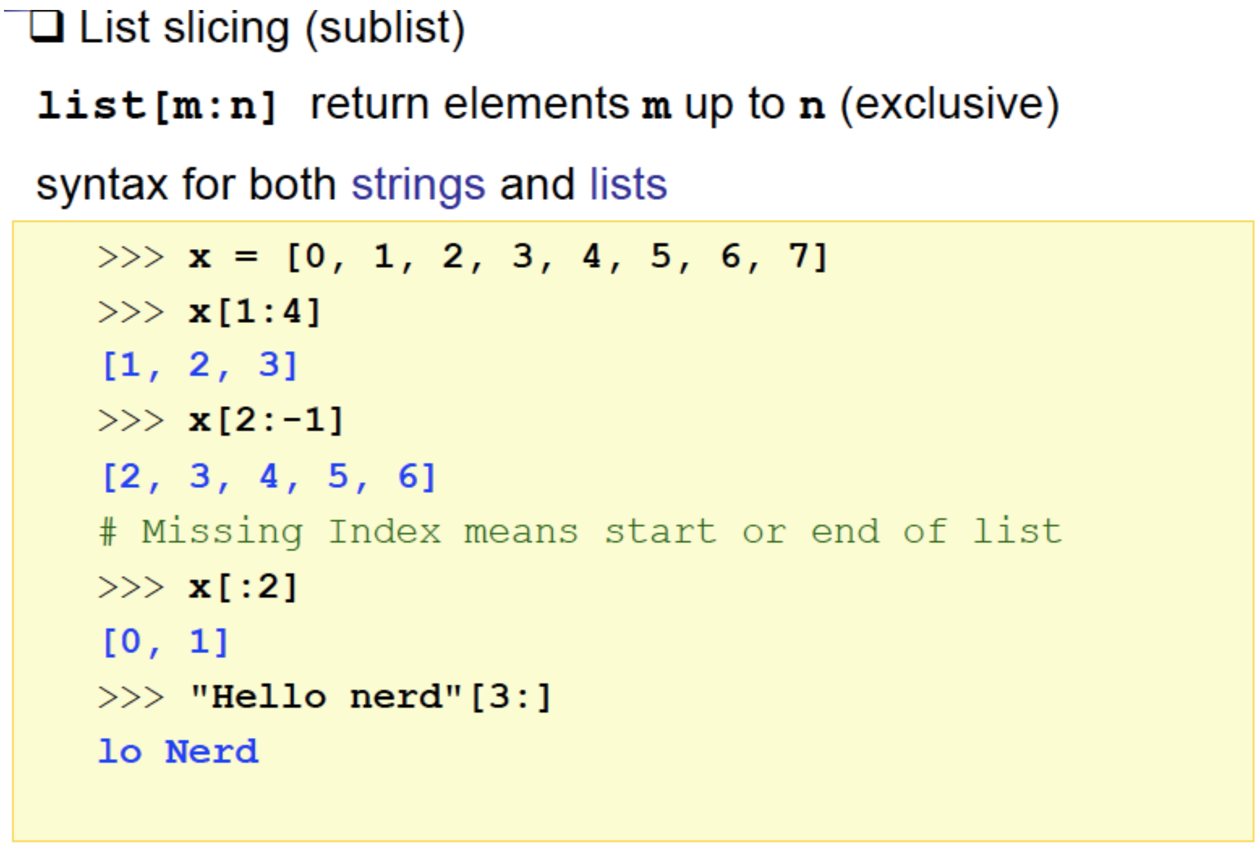
- Positive indices count from the left: `list[0]`
- Negative indices count from the right: `list[-1]`



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

```
list[0] == a        list[-1] == g
list[2] == c        list[-2] == f
list[6] == g        list[-7] == a
```

Containers: List



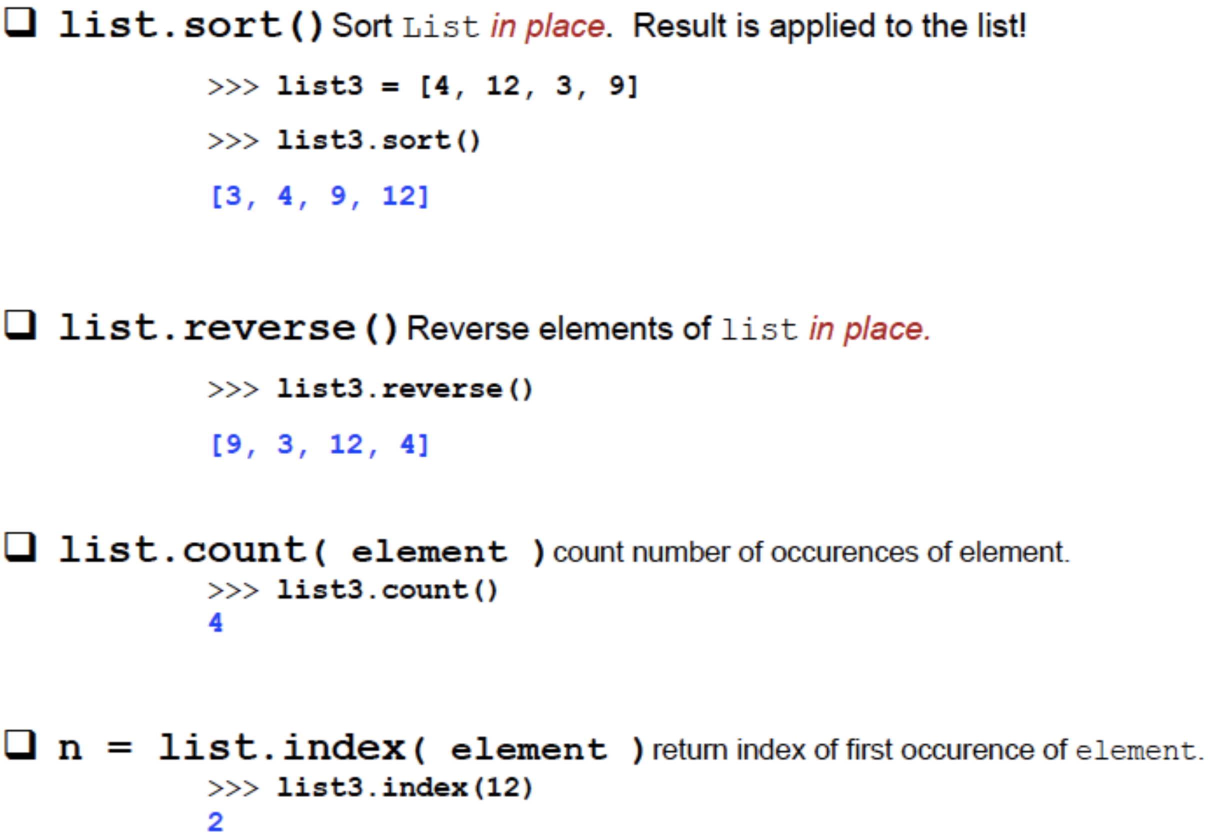
❑ List slicing (sublist)

list[m:n] return elements **m** up to **n** (exclusive)

syntax for both strings and lists

```
>>> x = [0, 1, 2, 3, 4, 5, 6, 7]
>>> x[1:4]
[1, 2, 3]
>>> x[2:-1]
[2, 3, 4, 5, 6]
# Missing Index means start or end of list
>>> x[:2]
[0, 1]
>>> "Hello nerd"[3:]
lo Nerd
```

Containers: List



❑ `list.sort()` Sort `List` *in place*. Result is applied to the list!

```
>>> list3 = [4, 12, 3, 9]
>>> list3.sort()
[3, 4, 9, 12]
```

❑ `list.reverse()` Reverse elements of `list` *in place.*

```
>>> list3.reverse()
[9, 3, 12, 4]
```

❑ `list.count( element )` count number of occurences of element.

```
>>> list3.count()
4
```

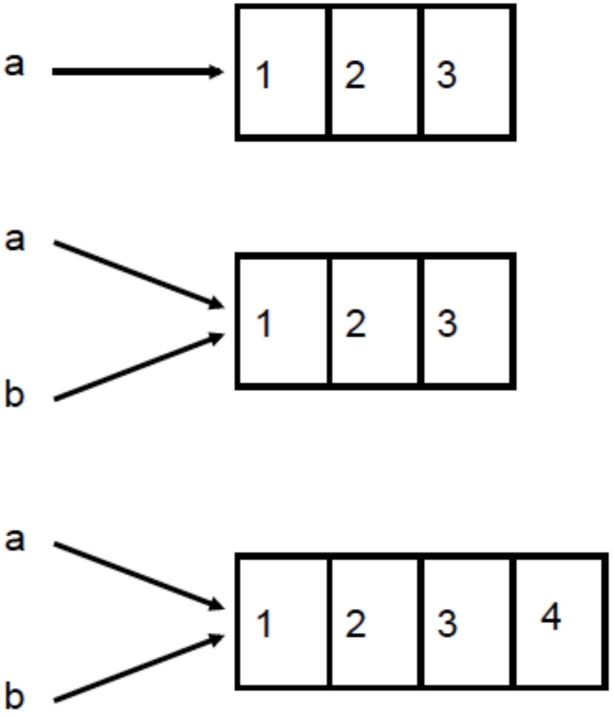
❑ `n = list.index( element )` return index of first occurence of `element`.

```
>>> list3.index(12)
2
```

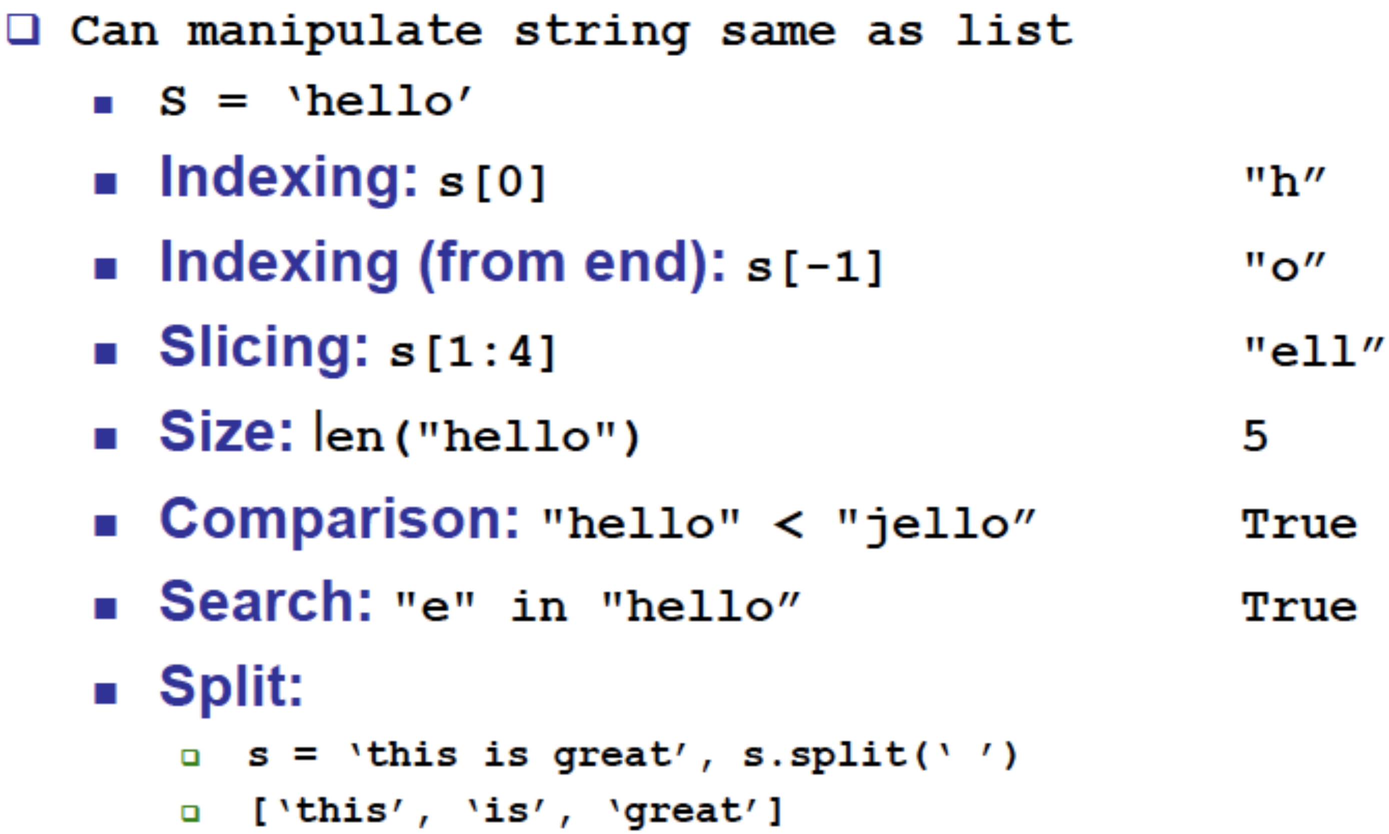
Containers: List

- Modifying shared lists

```
a = [1, 2, 3]

b = a

a.append(4)
```

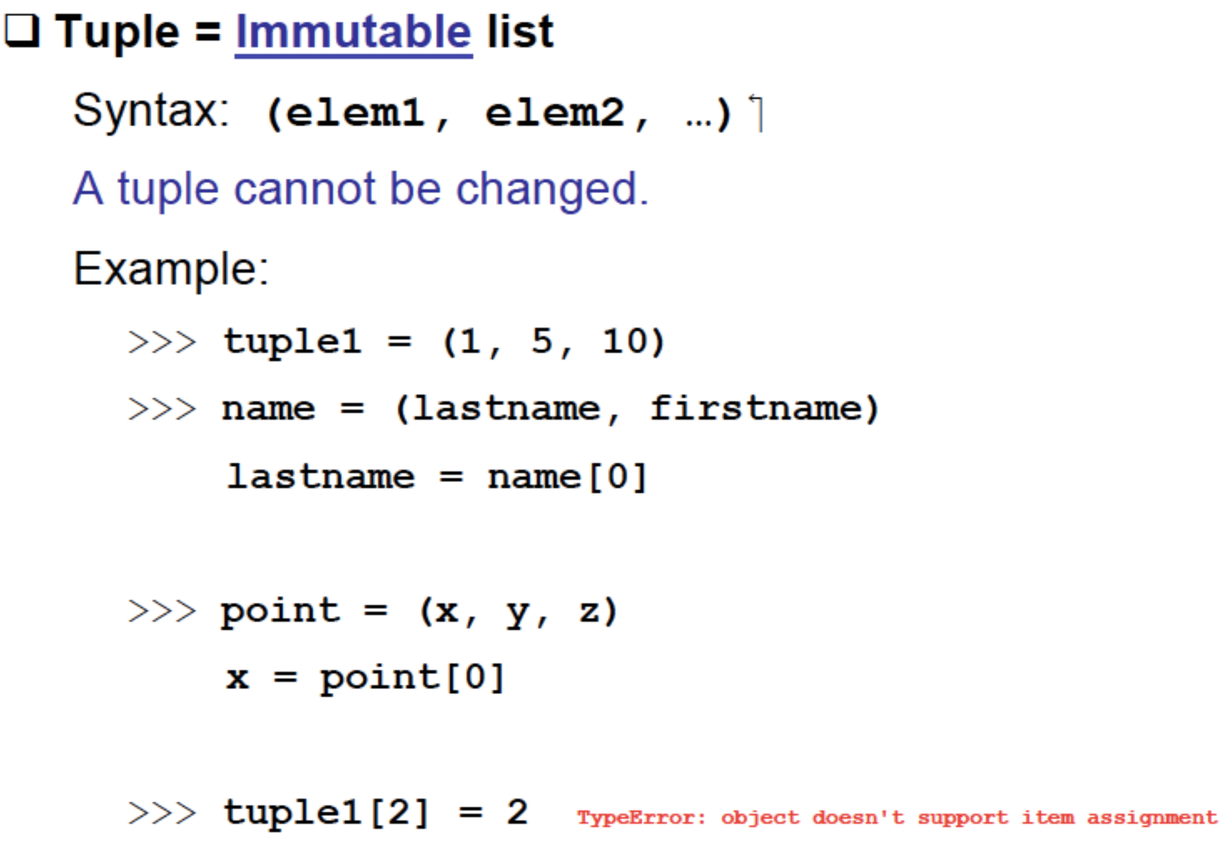


Containers: List



- ❑ Can manipulate string same as list
  - S = ‘hello’
  - Indexing: s[0] "h″
  - Indexing (from end): s[-1] "o″
  - Slicing: s[1:4] "ell″
  - Size: len("hello") 5
  - Comparison: "hello" < "jello″ True
  - Search: "e" in "hello″ True
  - Split:
    - s = ‘this is great’, s.split(‘ ’)
    - [‘this’, ‘is’, ‘great’]

Containers: Tupple



❑ **Tuple = Immutable list**

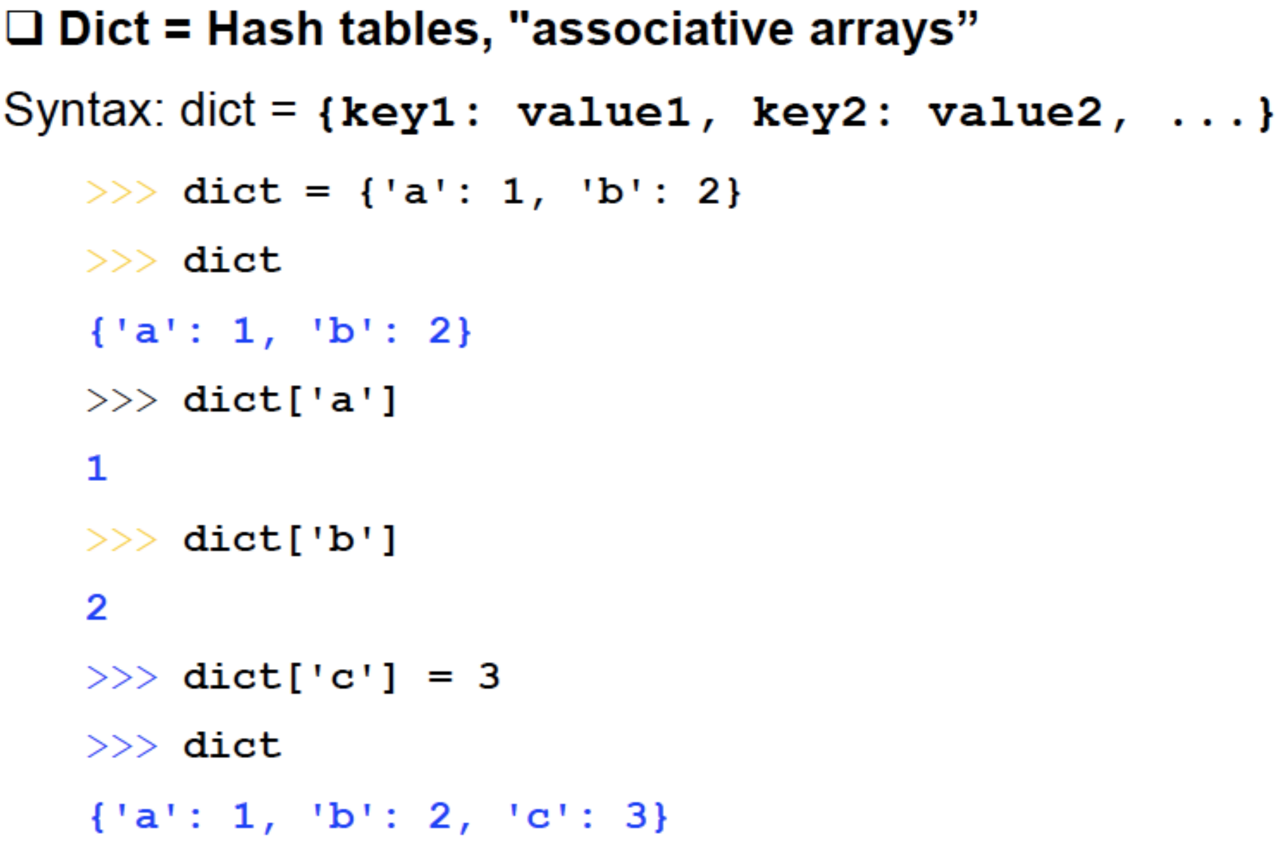
Syntax: `(elem1, elem2, …)`

A tuple cannot be changed.

Example:

```
>>> tuple1 = (1, 5, 10)
>>> name = (lastname, firstname)
    lastname = name[0]


>>> point = (x, y, z)
    x = point[0]


>>> tuple1[2] = 2    TypeError: object doesn't support item assignment
```

Containers: Dict

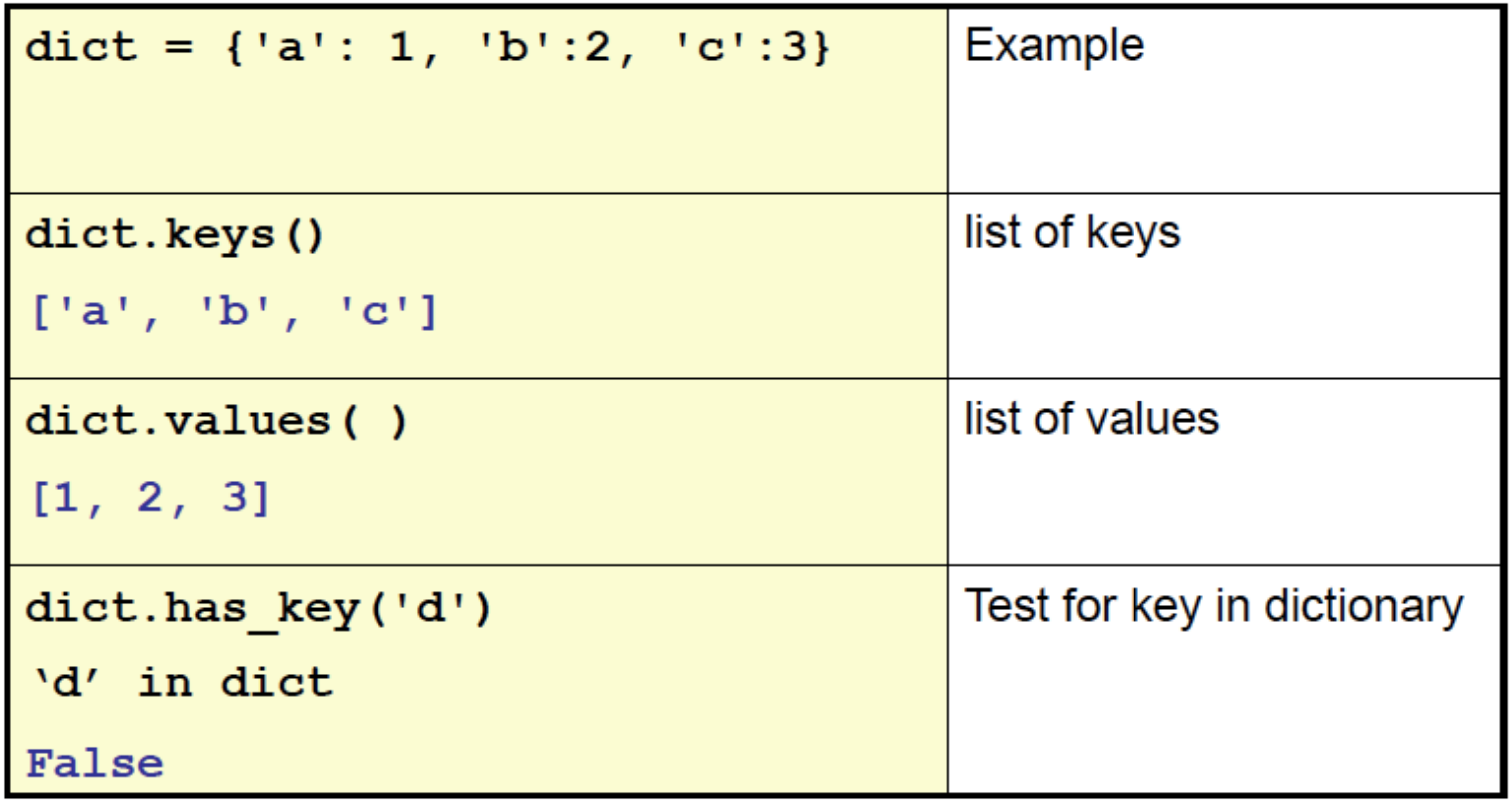


❑ **Dict = Hash tables, "associative arrays"**

Syntax: dict = **{key1: value1, key2: value2, ...}**

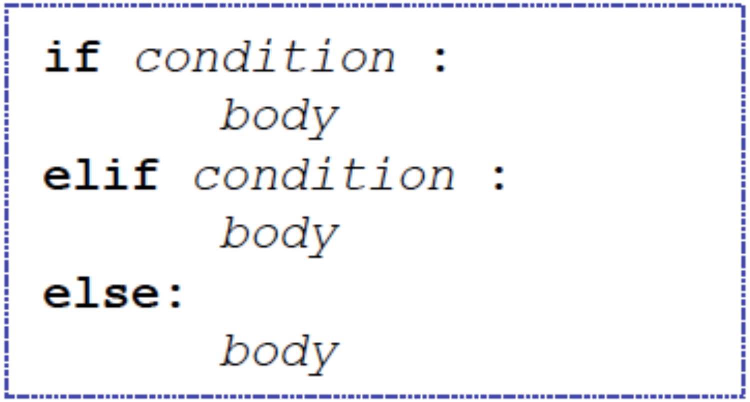
```
>>> dict = {'a': 1, 'b': 2}
>>> dict
{'a': 1, 'b': 2}
>>> dict['a']
1
>>> dict['b']
2
>>> dict['c'] = 3
>>> dict
{'a': 1, 'b': 2, 'c': 3}
```

Containers: Dict

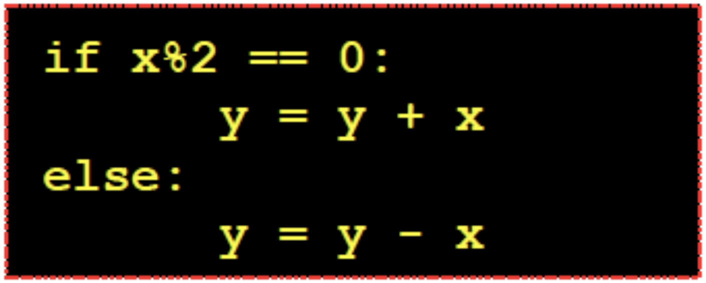


dict = {'a': 1, 'b':2, 'c':3}	Example
dict.keys() ['a', 'b', 'c']	list of keys
dict.values( ) [1, 2, 3]	list of values
dict.has_key('d') ‘d’ in dict False	Test for key in dictionary

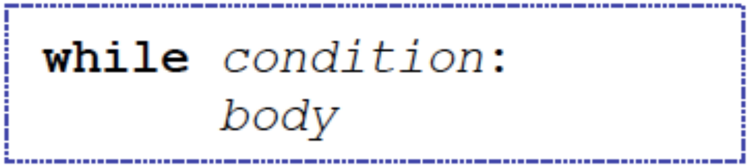
Flow control



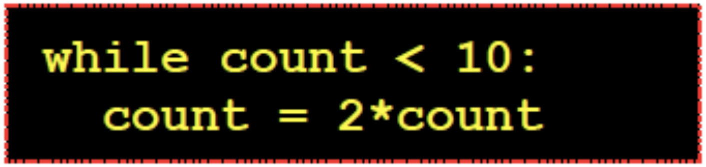
```
if condition :
      body
elif condition :
      body
else:
      body
```



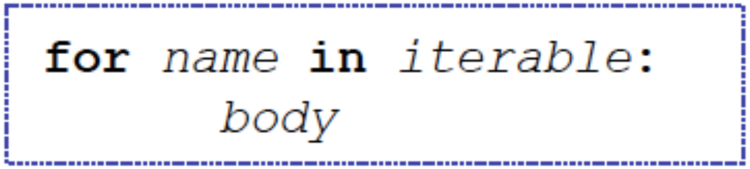
```
if x%2 == 0:
      y = y + x
else:
      y = y - x
```



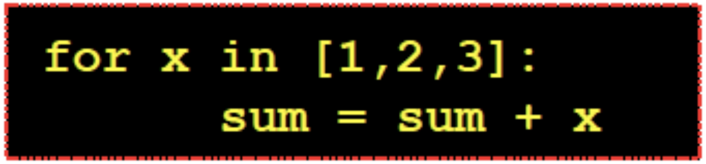
```
while condition:
      body
```



```
while count < 10:
  count = 2*count
```

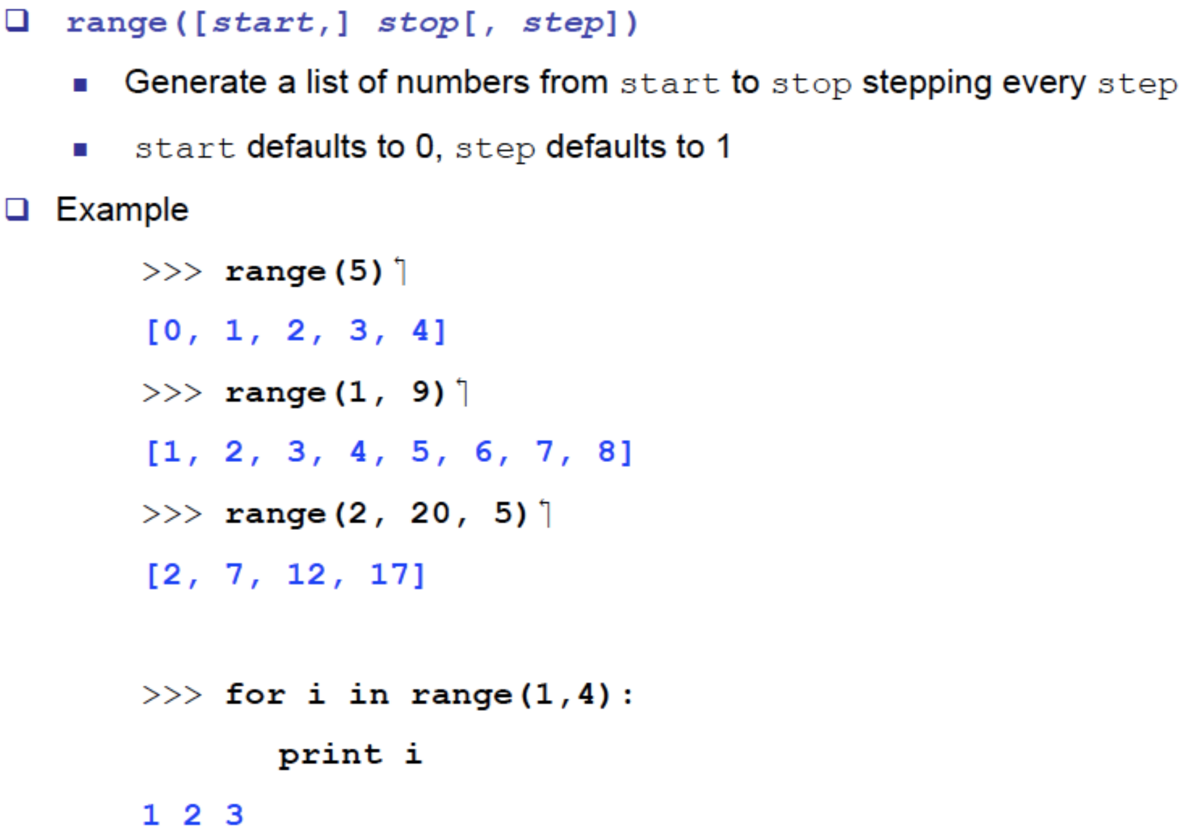


```
for name in iterable:
      body
```



```
for x in [1,2,3]:
      sum = sum + x
```

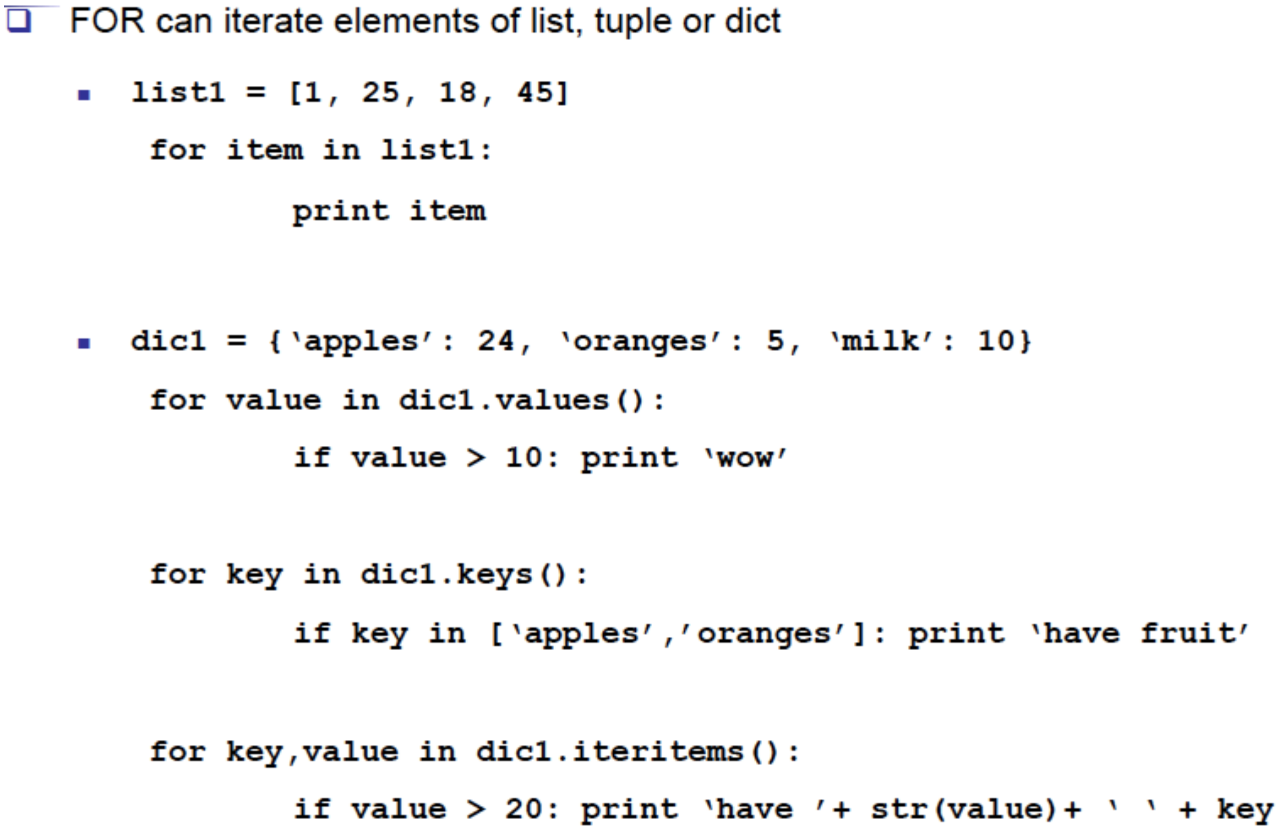
Flow control



- ❑ `range([start,] stop[, step])`
  - ■ Generate a list of numbers from `start` to `stop` stepping every `step`
  - ■ `start` defaults to 0, `step` defaults to 1
- ❑ Example

```
>>> range(5)↰
[0, 1, 2, 3, 4]
>>> range(1, 9)↰
[1, 2, 3, 4, 5, 6, 7, 8]
>>> range(2, 20, 5)↰
[2, 7, 12, 17]

>>> for i in range(1,4):
        print i
1 2 3
```

Flow control



❑ FOR can iterate elements of list, tuple or dict

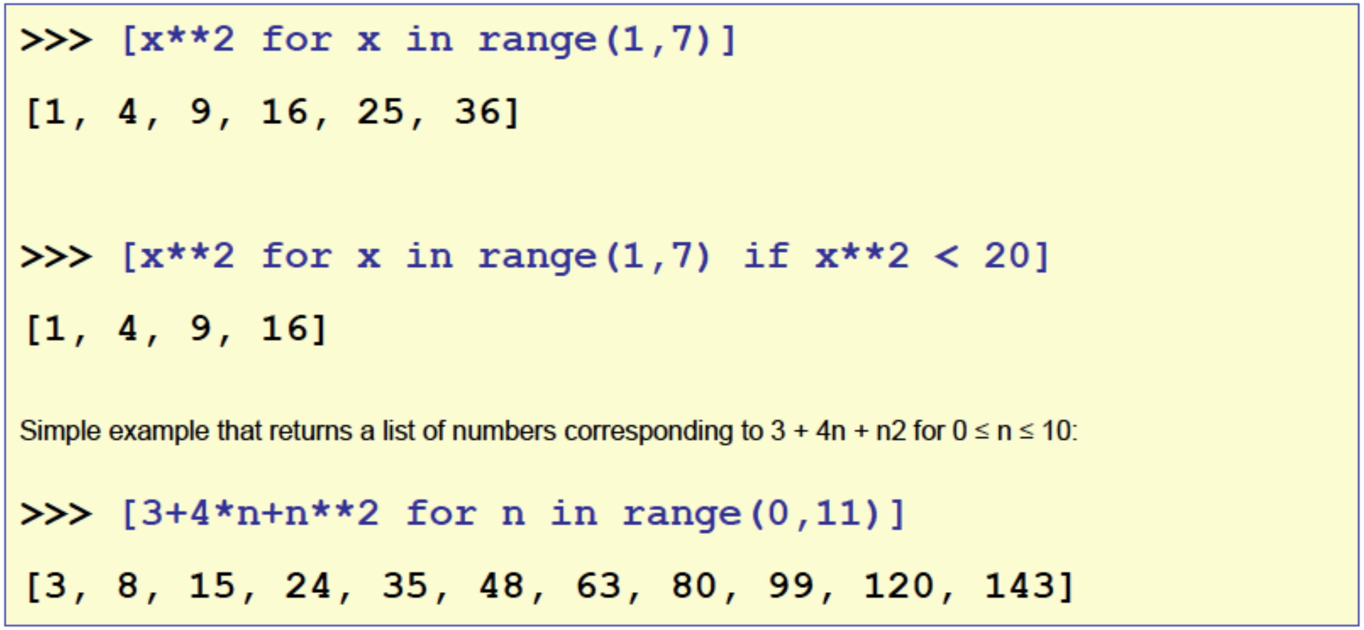
-
  ```
  list1 = [1, 25, 18, 45]
   for item in list1:
          print item
  ```

-
  ```
  dic1 = {‘apples’: 24, ‘oranges’: 5, ‘milk’: 10}
   for value in dic1.values():
          if value > 10: print ‘wow’

   for key in dic1.keys():
          if key in [‘apples’,’oranges’]: print ‘have fruit’

   for key,value in dic1.iteritems():
          if value > 20: print ‘have ’+ str(value)+ ‘ ‘ + key
  ```

List using flow control

```
[ expression for var in list if cond]
```

Generate a list by applying an expression to every element of an iterable

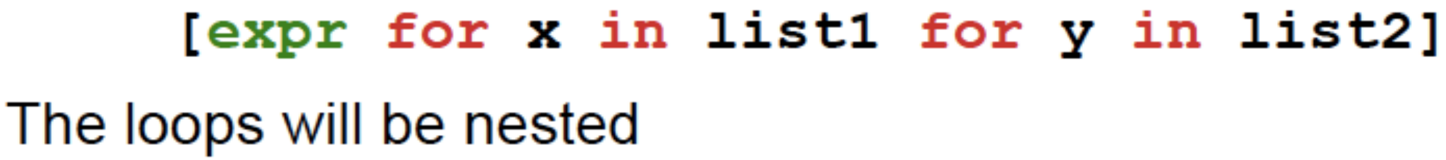


```
>>> [x**2 for x in range(1,7)]
[1, 4, 9, 16, 25, 36]


>>> [x**2 for x in range(1,7) if x**2 < 20]
[1, 4, 9, 16]
```

Simple example that returns a list of numbers corresponding to 3 + 4n + n2 for 0 ≤ n ≤ 10:

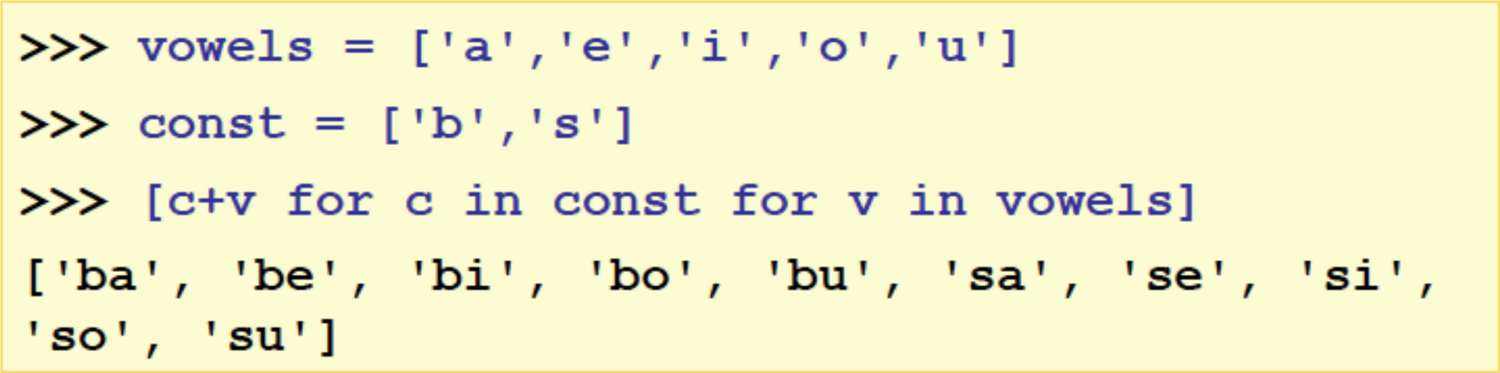
```
>>> [3+4*n+n**2 for n in range(0,11)]
[3, 8, 15, 24, 35, 48, 63, 80, 99, 120, 143]
```

List using flow control



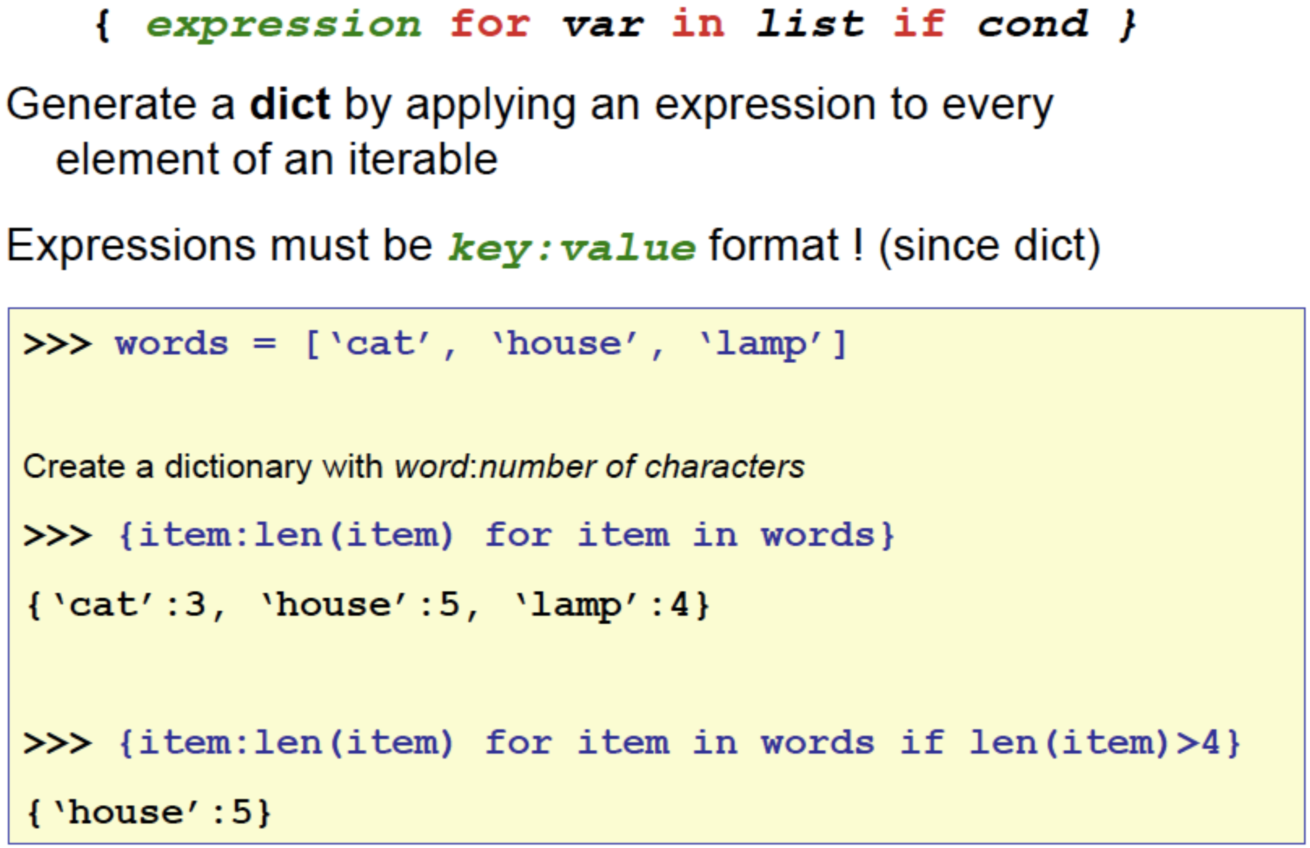
```
[expr for x in list1 for y in list2]
```

The loops will be nested



```
>>> vowels = ['a','e','i','o','u']
>>> const = ['b','s']
>>> [c+v for c in const for v in vowels]
['ba', 'be', 'bi', 'bo', 'bu', 'sa', 'se', 'si',
'so', 'su']
```

Dict using flow control



```
{ expression for var in list if cond }
```

Generate a **dict** by applying an expression to every element of an iterable

Expressions must be `key:value` format ! (since dict)

```
>>> words = [‘cat’, ‘house’, ‘lamp’]
```

Create a dictionary with *word:number of characters*

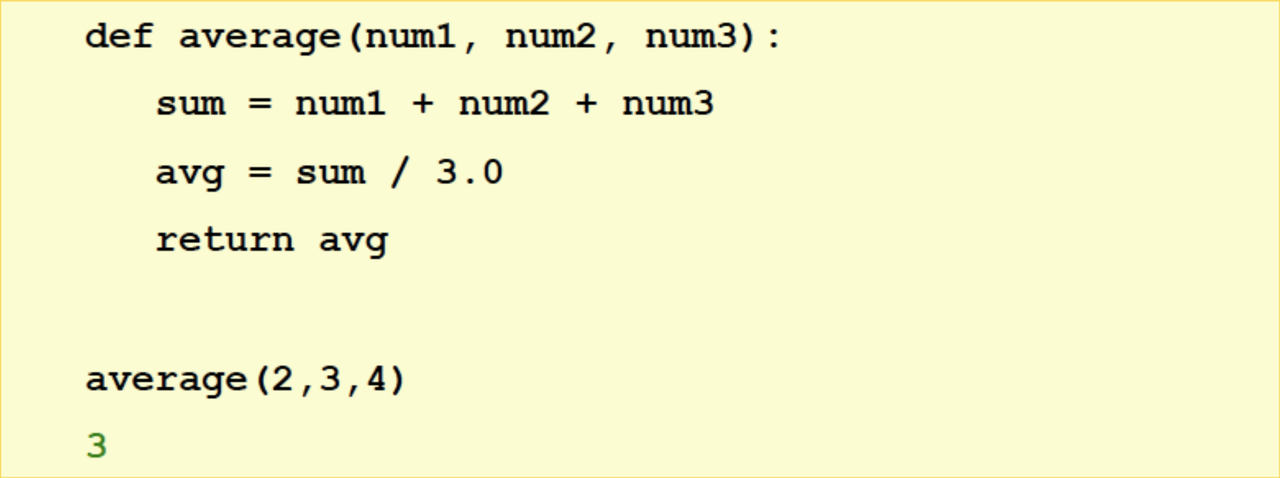
```
>>> {item:len(item) for item in words}
{‘cat’:3, ‘house’:5, ‘lamp’:4}


>>> {item:len(item) for item in words if len(item)>4}
{‘house’:5}
```

Functions

Syntax: `def func(arg1, arg2, …):`

```
                    body
                    return x
```

- Body of function must be indented



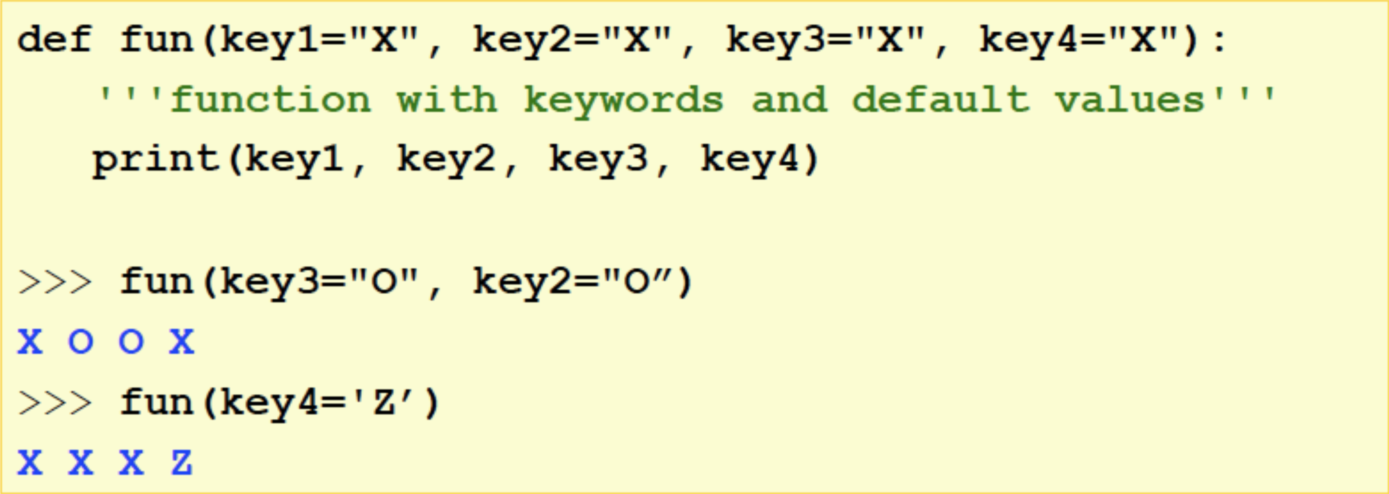
```
def average(num1, num2, num3):
    sum = num1 + num2 + num3
    avg = sum / 3.0
    return avg


average(2,3,4)
3
```

Functions

Functions can be invoked using the name of the argument and a value

```
func(argument=value, ...)
```

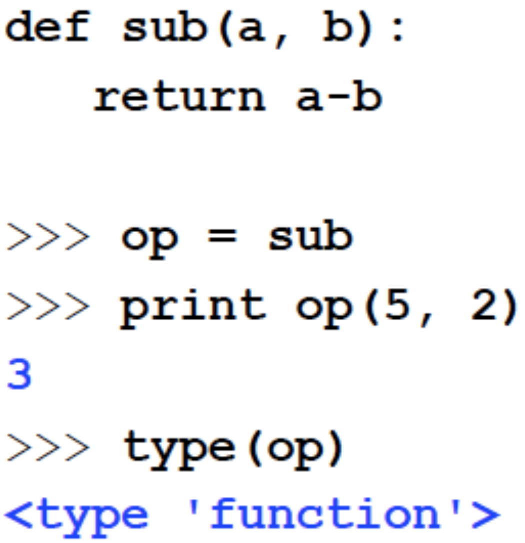
- The order of values passed by keyword does not matter



```
def fun(key1="X", key2="X", key3="X", key4="X"):
    '''function with keywords and default values'''
    print(key1, key2, key3, key4)

>>> fun(key3="O", key2="O”)
X O O X
>>> fun(key4='Z’)
X X X Z
```

Functions

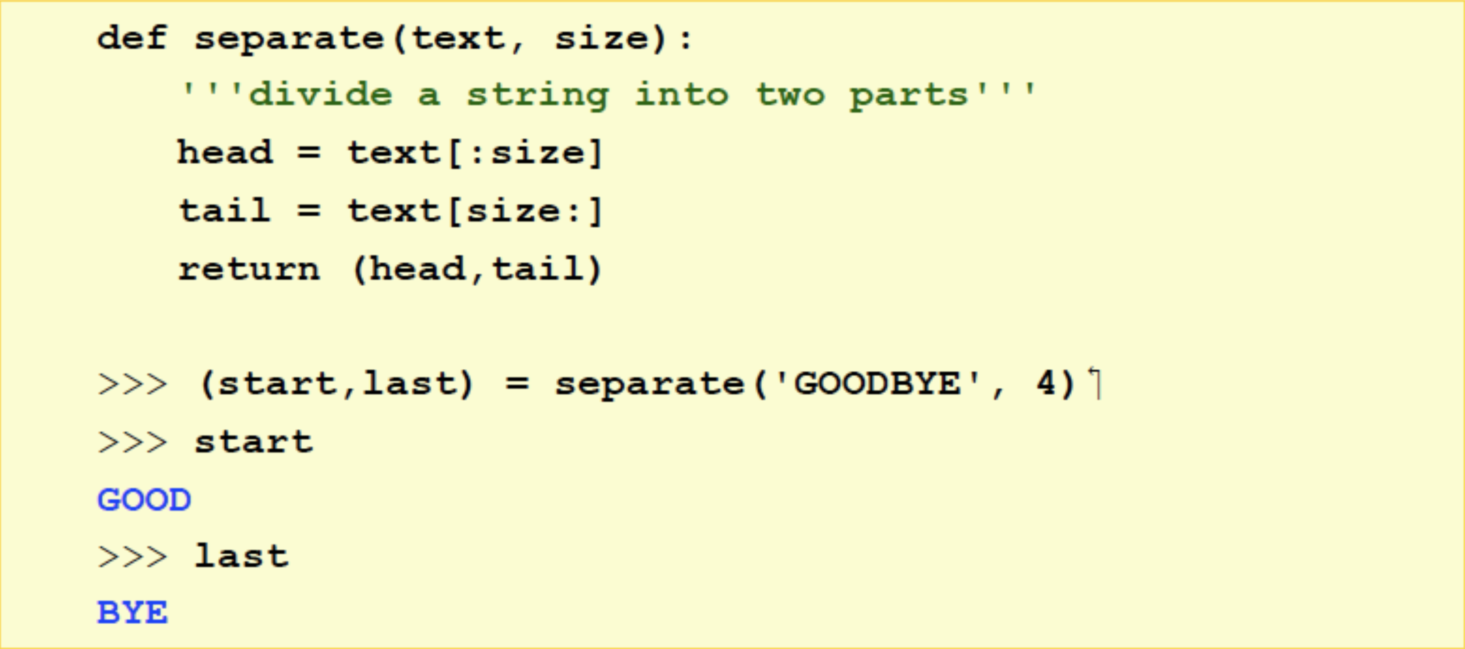
- Functions can be used just like any other data type
- Functions can be assigned to variables



```
def sub(a, b):
    return a-b

>>> op = sub
>>> print op(5, 2)
3
>>> type(op)
<type 'function'>
```

Functions

Functions can return multiple values (as a tuple)

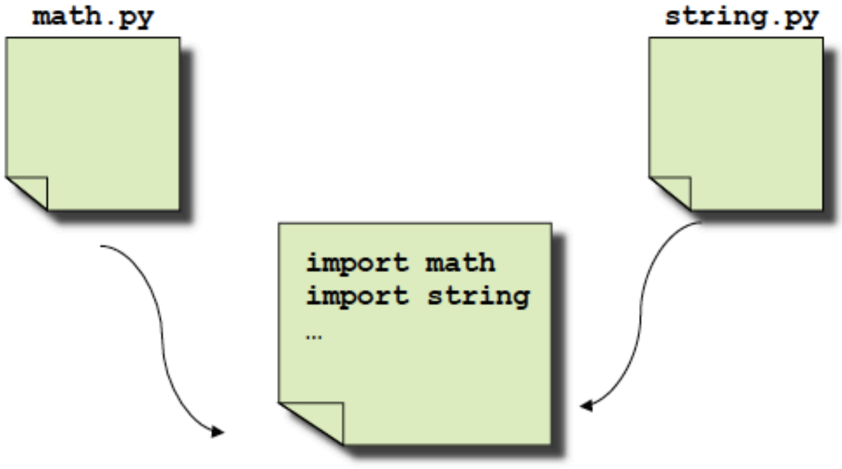


```
def separate(text, size):
    '''divide a string into two parts'''
    head = text[:size]
    tail = text[size:]
    return (head,tail)

>>> (start,last) = separate('GOODBYE', 4)
>>> start
GOOD
>>> last
BYE
```

Modules

A file containing Python definitions and statements

- Modules can be “imported”
- Module file name must end in .py
- Used to divide code between files

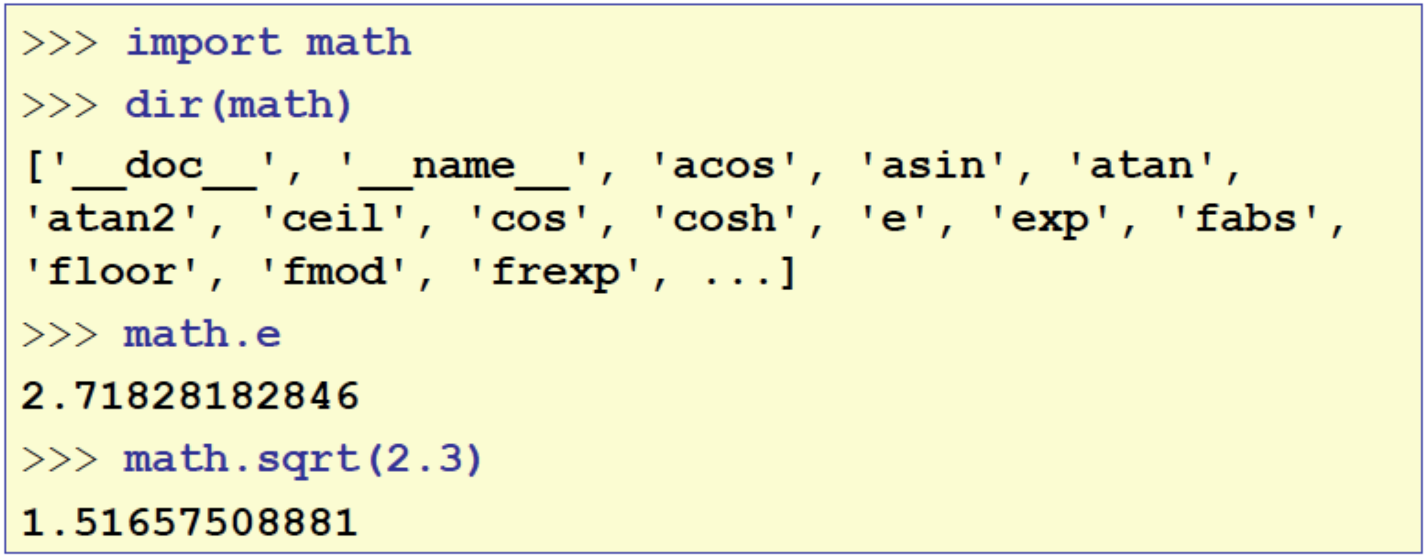


```
import math
import string
…
```

Modules

```
import <module name>
```

- **module name** is the file name without **.py** extension
- You must use the module name to call functions

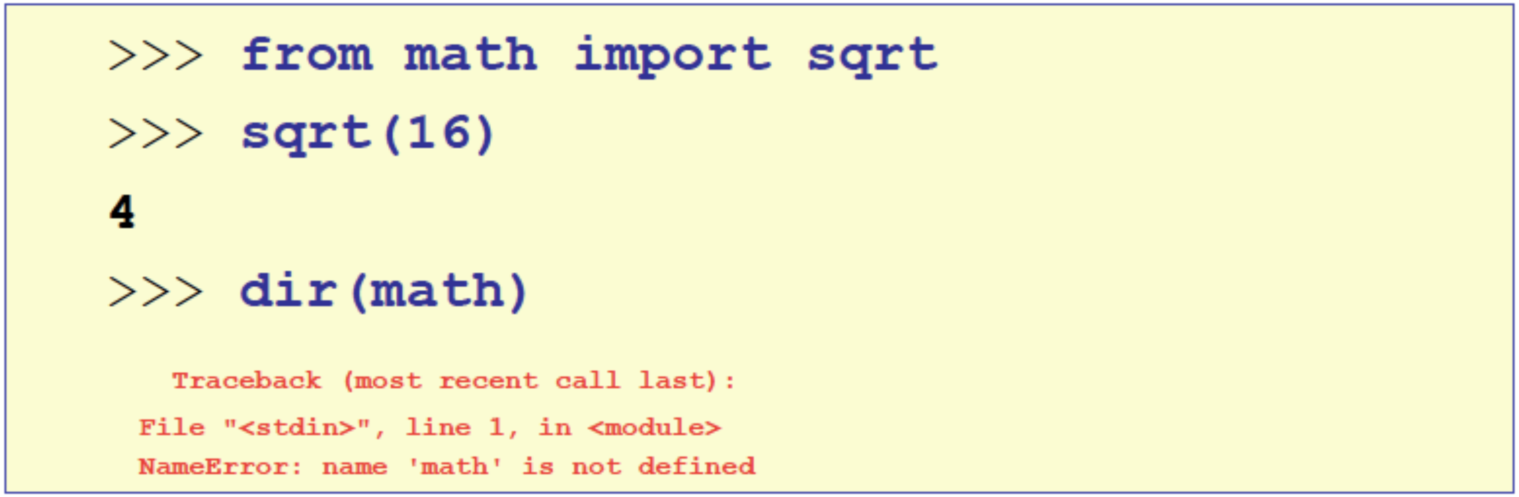


```
>>> import math
>>> dir(math)
['__doc__', '__name__', 'acos', 'asin', 'atan',
'atan2', 'ceil', 'cos', 'cosh', 'e', 'exp', 'fabs',
'floor', 'fmod', 'frexp', ...]
>>> math.e
2.71828182846
>>> math.sqrt(2.3)
1.51657508881
```

Modules

```
from <module> import <name>
```

- Import a specific name from a module into global namespace
- Module name is not required to access imported name(s)

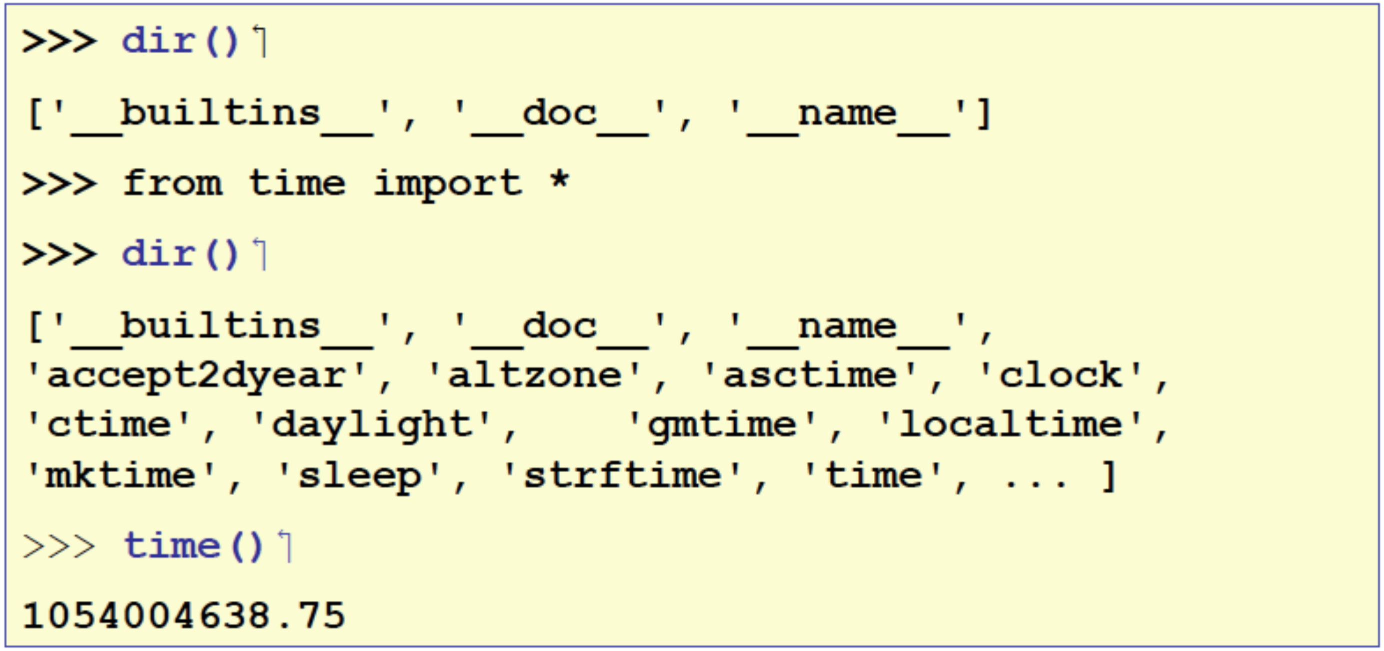


```
>>> from math import sqrt
>>> sqrt(16)
4
>>> dir(math)

    Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  NameError: name 'math' is not defined
```

Modules

```
from <module> import *
```

- Import everything into global namespace

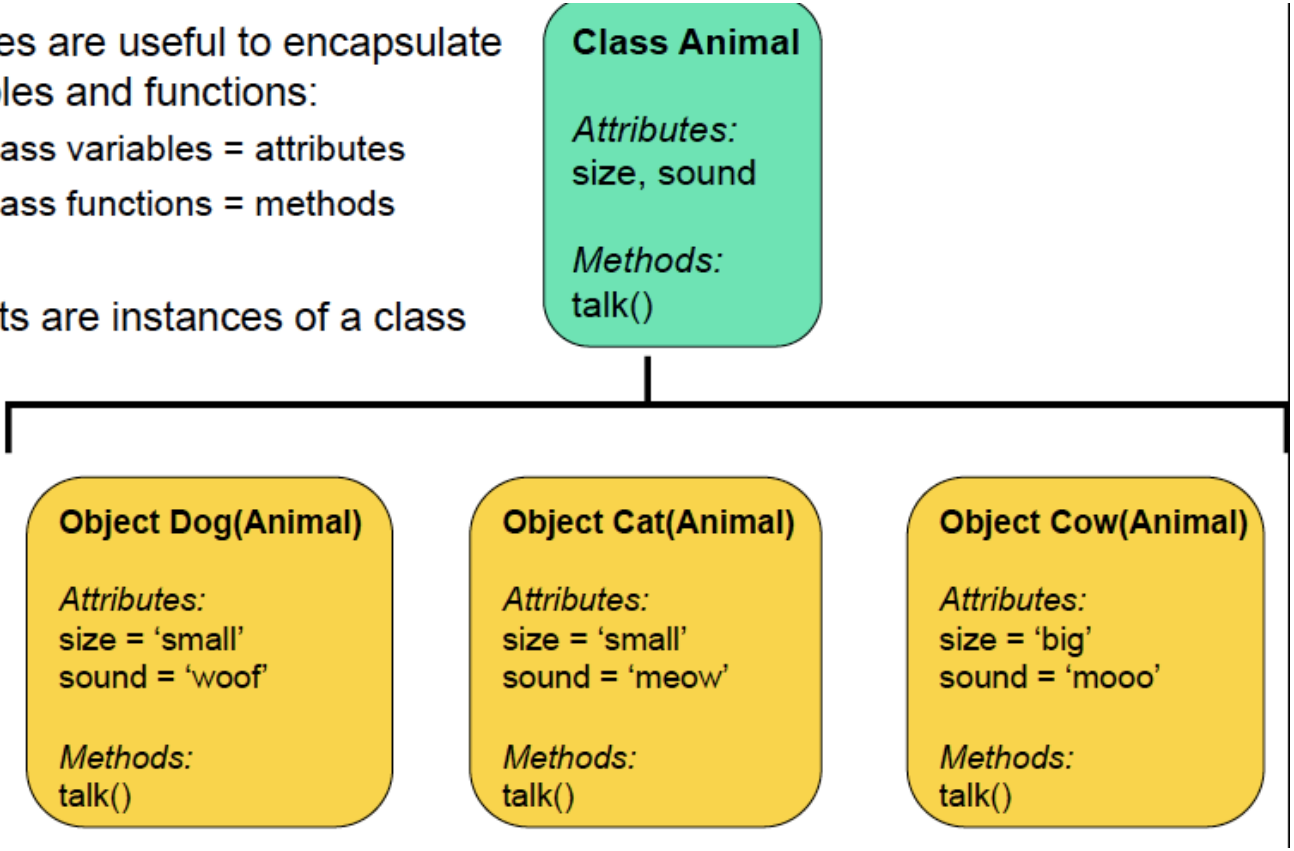


```
>>> dir()↰
['__builtins__', '__doc__', '__name__']
>>> from time import *
>>> dir()↰
['__builtins__', '__doc__', '__name__',
'accept2dyear', 'altzone', 'asctime', 'clock',
'ctime', 'daylight',    'gmtime', 'localtime',
'mktime', 'sleep', 'strftime', 'time', ... ]
>>> time()↰
1054004638.75
```

Classes and objects

- ❑ Classes are useful to encapsulate variables and functions:
  - ❑ Class variables = attributes
  - ❑ Class functions = methods

- ❑ Objects are instances of a class



Class Animal

Attributes:
size, sound

Methods:
talk()

Object Dog(Animal)

Attributes:
size = ‘small’
sound = ‘woof’

Methods:
talk()

Object Cat(Animal)

Attributes:
size = ‘small’
sound = ‘meow’

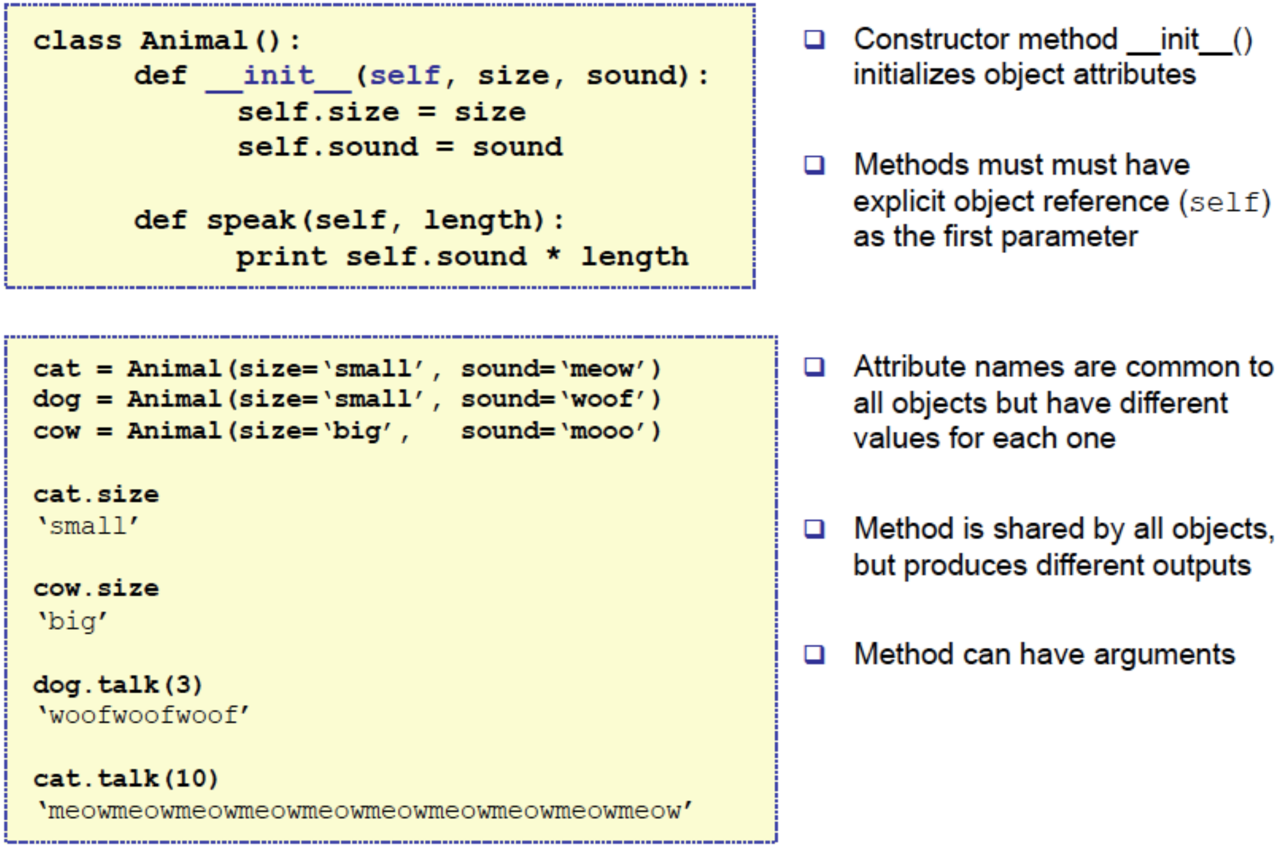
Methods:
talk()

Object Cow(Animal)

Attributes:
size = ‘big’
sound = ‘mooo’

Methods:
talk()

Classes and objects



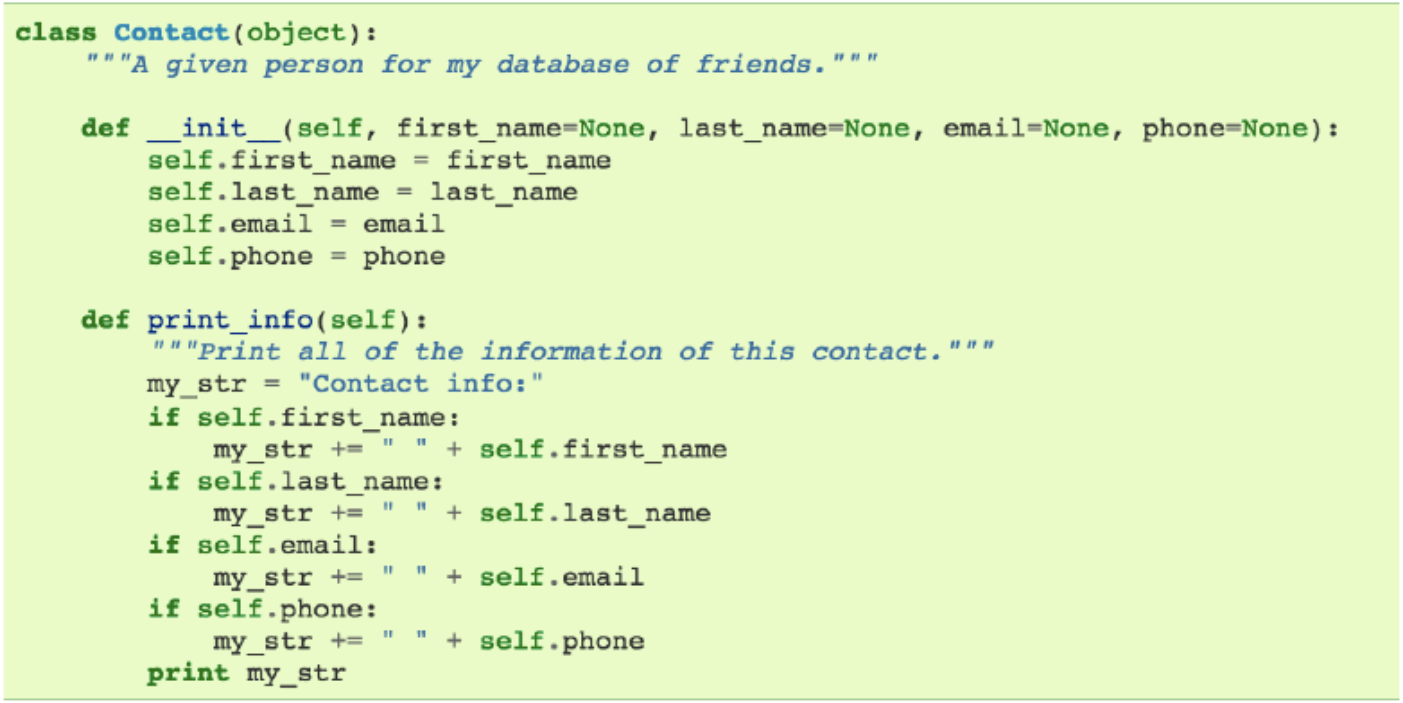
```
class Animal():
    def __init__(self, size, sound):
        self.size = size
        self.sound = sound

    def speak(self, length):
        print self.sound * length
```

- ❑ Constructor method __init__() initializes object attributes
- ❑ Methods must must have explicit object reference (`self`) as the first parameter

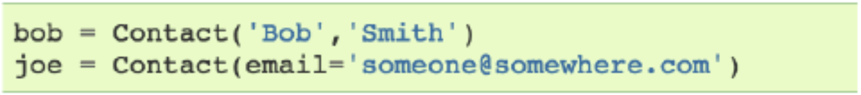
```
cat = Animal(size=‘small’, sound=‘meow’)
dog = Animal(size=‘small’, sound=‘woof’)
cow = Animal(size=‘big’,   sound=‘mooo’)

cat.size
‘small’

cow.size
‘big’

dog.talk(3)
‘woofwoofwoof’

cat.talk(10)
‘meowmeowmeowmeowmeowmeowmeowmeowmeowmeow’
```

- ❑ Attribute names are common to all objects but have different values for each one
- ❑ Method is shared by all objects, but produces different outputs
- ❑ Method can have arguments

Classes and objects



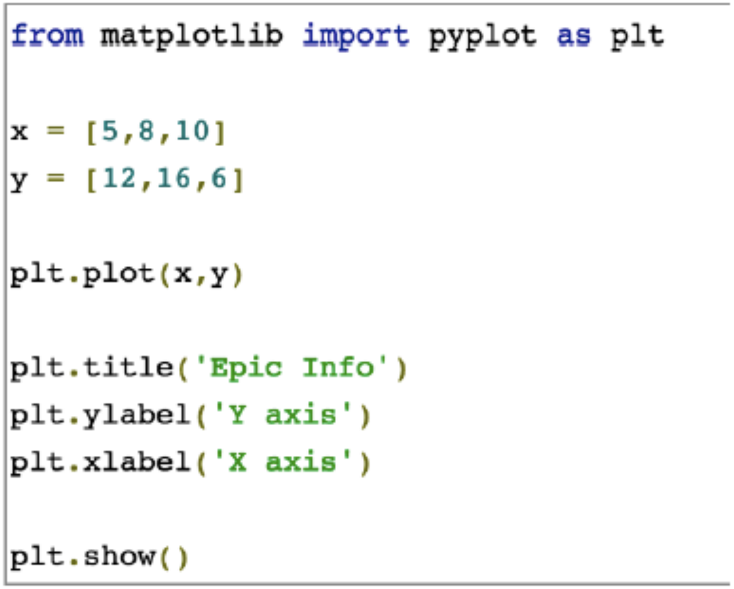
```python
class Contact(object):
    """A given person for my database of friends."""

    def __init__(self, first_name=None, last_name=None, email=None, phone=None):
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.phone = phone

    def print_info(self):
        """Print all of the information of this contact."""
        my_str = "Contact info:"
        if self.first_name:
            my_str += " " + self.first_name
        if self.last_name:
            my_str += " " + self.last_name
        if self.email:
            my_str += " " + self.email
        if self.phone:
            my_str += " " + self.phone
        print my_str
```



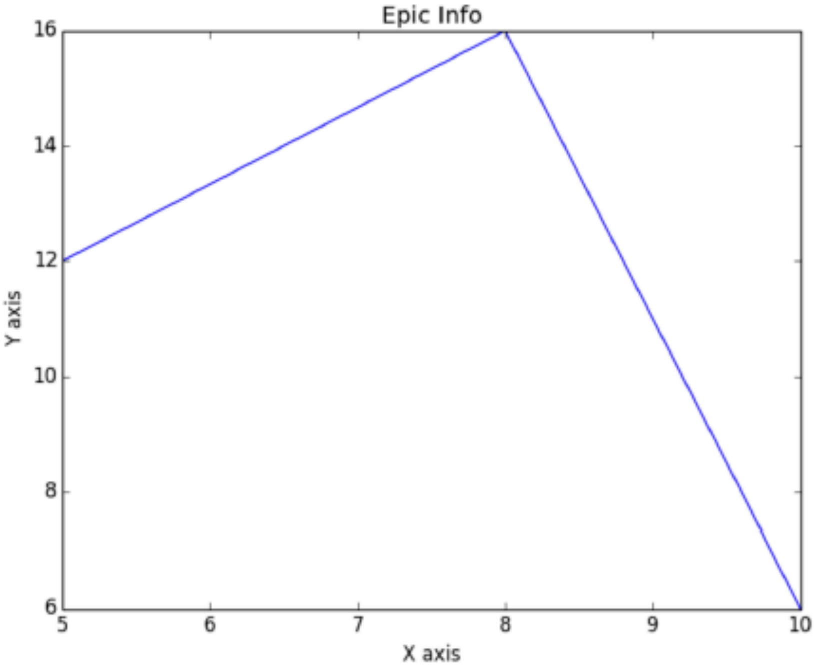
```python
bob = Contact('Bob','Smith')
joe = Contact(email='someone@somewhere.com')
```

Plotting

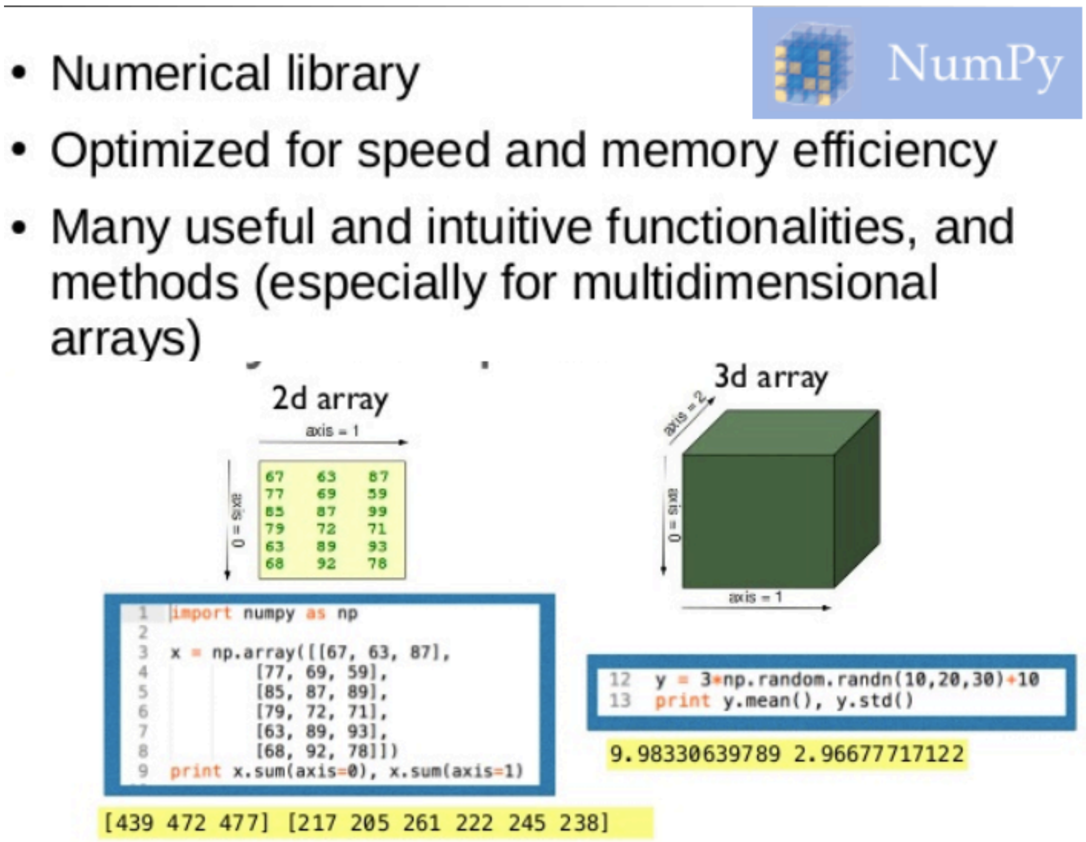
- ❑ Matplotlib (very similar to Matlab)



```
from matplotlib import pyplot as plt

x = [5,8,10]
y = [12,16,6]

plt.plot(x,y)

plt.title('Epic Info')
plt.ylabel('Y axis')
plt.xlabel('X axis')

plt.show()
```



Plotting



- Numerical library
- Optimized for speed and memory efficiency
- Many useful and intuitive functionalities, and methods (especially for multidimensional arrays)

67	63	87
77	69	59
85	87	99
79	72	71
63	89	93
68	92	78

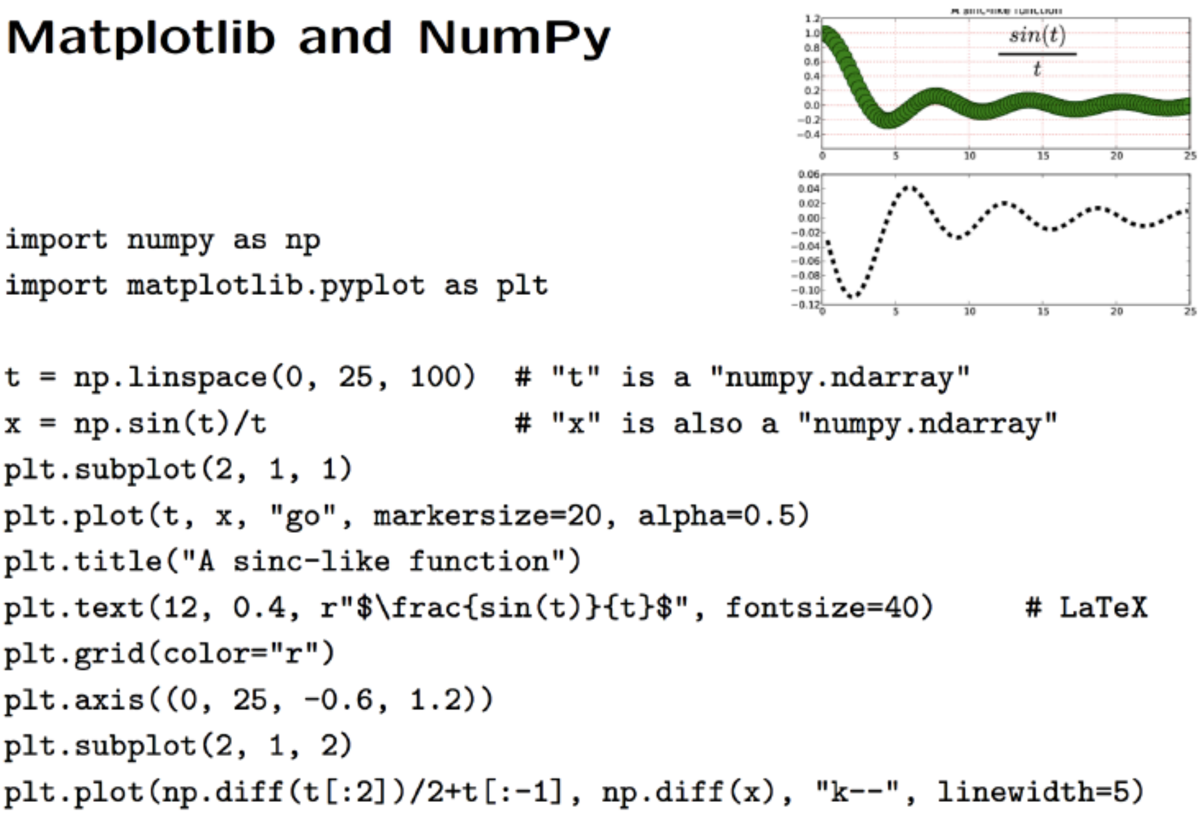
```
import numpy as np

x = np.array([[67, 63, 87],
              [77, 69, 59],
              [85, 87, 89],
              [79, 72, 71],
              [63, 89, 93],
              [68, 92, 78]])
print x.sum(axis=0), x.sum(axis=1)
```

```
[439 472 477] [217 205 261 222 245 238]
```

```
y = 3*np.random.randn(10,20,30)+10
print y.mean(), y.std()
```

```
9.98330639789 2.96677717122
```

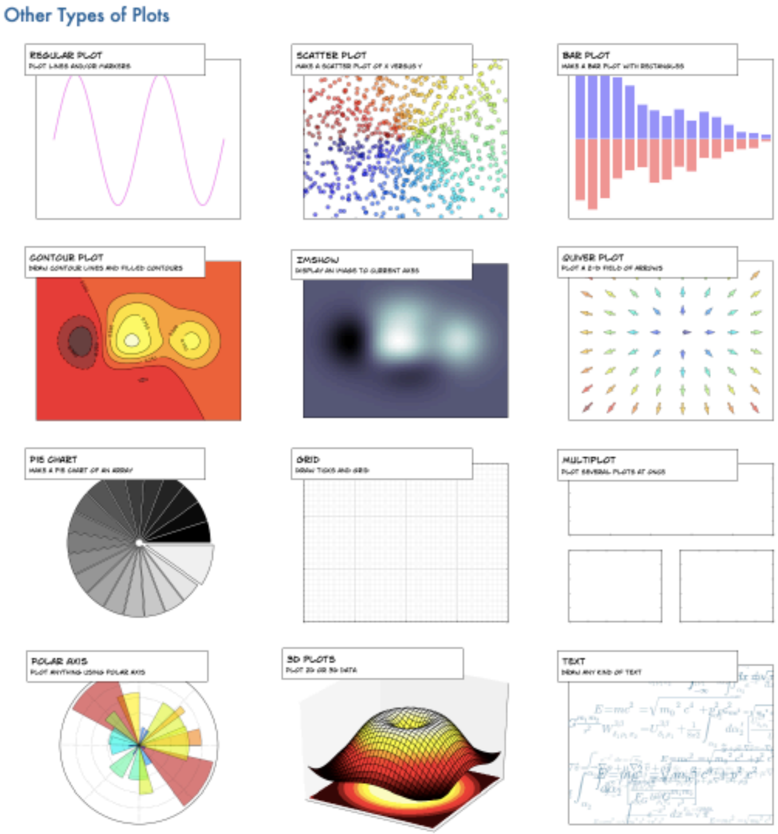
Plotting



Matplotlib and NumPy

```
import numpy as np
import matplotlib.pyplot as plt

t = np.linspace(0, 25, 100)  # "t" is a "numpy.ndarray"
x = np.sin(t)/t              # "x" is also a "numpy.ndarray"
plt.subplot(2, 1, 1)
plt.plot(t, x, "go", markersize=20, alpha=0.5)
plt.title("A sinc-like function")
plt.text(12, 0.4, r"$\frac{sin(t)}{t}$", fontsize=40)     # LaTeX
plt.grid(color="r")
plt.axis((0, 25, -0.6, 1.2))
plt.subplot(2, 1, 2)
plt.plot(np.diff(t[:2])/2+t[:-1], np.diff(x), "k--", linewidth=5)
```

Plotting



Other Types of Plots