PYTHON TUTORIAL

Copied from Guido van Rossum's tutorial
on www.python.org

OUTLINE

- interactive "shell"
- basic types: numbers, strings
- container types: lists, dictionaries, tuples
- variables
- control structures
- functions & procedures
- classes & instances
- scripts and modules
- numpy module

Interactive “shell”

- Great for learning the language
- Great for experimenting with the library
- Great for testing your own modules

- Type statements or expressions at prompt:

```
>>> print "Hello, world"

>>> x = 12**2
>>> x/2

>>> # this is a comment
```

Interactive “shell”

- Great for learning the language
- Great for experimenting with the library
- Great for testing your own modules

- Type statements or expressions at prompt:

```
>>> print "Hello, world"
Hello, world
>>> x = 12**2
>>> x/2
72
>>> # this is a comment
```

Basic types: Numbers

- Integer

```
>>> a = 12
>>> type(a)

>>> a / 5
```

- Real

```
>>> x = 12.
>>> print type(x)

>>> x / 5
```

- Logical

```
>>> a == x
```

- Arithmetic operations

```
>>> a + 12

>>> b = (5.+x) * a - 13.**2
>>> print b
```

- Logic operations

```
>>> cond = a == x and b < 40.
>>> print cond

>>> not cond or False
```

Basic types: Numbers

- Integer

```
>>> a = 12
>>> type(a)
<type 'int'>
>>> a / 5
2
```

- Real

```
>>> x = 12.
>>> print type(x)

>>> x / 5
```

- Logical

```
>>> a == x
```

- Arithmetic operations

```
>>> a + 12

>>> b = (5.+x) * a - 13.**2
>>> print b
```

- Logic operations

```
>>> cond = a == x and b < 40.
>>> print cond

>>> not cond or False
```

Basic types: Numbers

- Integer

```
>>> a = 12
>>> type(a)
<type 'int'>
>>> a / 5
2
```

- Real

```
>>> x = 12.
>>> print type(x)
<type 'float'>
>>> x / 5
2.4
```

- Logical

```
>>> a == x
```

- Arithmetic operations

```
>>> a + 12

>>> b = (5.+x) * a - 13.**2
>>> print b
```

- Logic operations

```
>>> cond = a == x and b < 40.
>>> print cond

>>> not cond or False
```

Basic types: Numbers

- Integer

```
>>> a = 12
>>> type(a)
<type 'int'>
>>> a / 5
2
```

- Real

```
>>> x = 12.
>>> print type(x)
<type 'float'>
>>> x / 5
2.4
```

- Logical

```
>>> a == x
True
```

- Arithmetic operations

```
>>> a + 12

>>> b = (5.+x) * a - 13.**2
>>> print b
```

- Logic operations

```
>>> cond = a == x and b < 40.
>>> print cond

>>> not cond or False
```

Basic types: Numbers

- Integer

```
>>> a = 12
>>> type(a)
<type 'int'>
>>> a / 5
2
```

- Real

```
>>> x = 12.
>>> print type(x)
<type 'float'>
>>> x / 5
2.4
```

- Logical

```
>>> a == x
True
```

- Arithmetic operations

```
>>> a + 12
14
>>> b = (5.+x) * a - 13.**2
>>> print b
35.
```

- Logic operations

```
>>> cond = a == x and b < 40.
>>> print cond

>>> not cond or False
```

Basic types: Numbers

- Integer

```
>>> a = 12
>>> type(a)
<type 'int'>
>>> a / 5
2
```

- Real

```
>>> x = 12.
>>> print type(x)
<type 'float'>
>>> x / 5
2.4
```

- Logical

```
>>> a == x
True
```

- Arithmetic operations

```
>>> a + 12
14
>>> b = (5.+x) * a - 13.**2
>>> print b
35.
```

- Logic operations

```
>>> cond = a == x and b < 40.
>>> print cond
True
>>> not cond or False
True
```

Basic types: Strings

```
>>> "hello" + "world"

>>> "hello"*3

>>> "hello"[0]

>>> "hello"[-1]

>>> "hello"[1:4]

>>> len("hello")

>>> "hello" < "jello"

>>> "e" in "hello"
```

Basic types: Strings

```
>>> "hello" + "world"     # concatenation
"helloworld"
>>> "hello"*3

>>> "hello"[0]

>>> "hello"[-1]

>>> "hello"[1:4]

>>> len("hello")

>>> "hello" < "jello"

>>> "e" in "hello"
```

Basic types: Strings

```
>>> "hello" + "world"     # concatenation
"helloworld"
>>> "hello"*3             # repetition
"hellohellohello"
>>> "hello"[0]

>>> "hello"[-1]

>>> "hello"[1:4]

>>> len("hello")

>>> "hello" < "jello"

>>> "e" in "hello"
```

Basic types: Strings

```
>>> "hello" + "world"     # concatenation
"helloworld"
>>> "hello"*3             # repetition
"hellohellohello"
>>> "hello"[0]            # indexing
"h"
>>> "hello"[-1]

>>> "hello"[1:4]

>>> len("hello")

>>> "hello" < "jello"

>>> "e" in "hello"
```

Basic types: Strings

```
>>> "hello" + "world"     # concatenation
"helloworld"
>>> "hello"*3             # repetition
"hellohellohello"
>>> "hello"[0]            # indexing
"h"
>>> "hello"[-1]           # (from end)
"o"
>>> "hello"[1:4]

>>> len("hello")

>>> "hello" < "jello"

>>> "e" in "hello"
```

Basic types: Strings

```
>>> "hello" + "world"     # concatenation
"helloworld"
>>> "hello"*3             # repetition
"hellohellohello"
>>> "hello"[0]            # indexing
"h"
>>> "hello"[-1]           # (from end)
"o"
>>> "hello"[1:4]          # slicing
"ell"
>>> len("hello")

>>> "hello" < "jello"

>>> "e" in "hello"
```

Basic types: Strings

```
>>> "hello" + "world"     # concatenation
"helloworld"
>>> "hello"*3             # repetition
"hellohellohello"
>>> "hello"[0]            # indexing
"h"
>>> "hello"[-1]           # (from end)
"o"
>>> "hello"[1:4]          # slicing
"ell"
>>> len("hello")          # size
5
>>> "hello" < "jello"

>>> "e" in "hello"
```

Basic types: Strings

```
>>> "hello" + "world"     # concatenation
"helloworld"
>>> "hello"*3             # repetition
"hellohellohello"
>>> "hello"[0]            # indexing
"h"
>>> "hello"[-1]           # (from end)
"o"
>>> "hello"[1:4]          # slicing
"ell"
>>> len("hello")          # size
5
>>> "hello" < "jello"     # comparison
True
>>> "e" in "hello"
```

Basic types: Strings

```
>>> "hello" + "world"     # concatenation
"helloworld"
>>> "hello"*3             # repetition
"hellohellohello"
>>> "hello"[0]            # indexing
"h"
>>> "hello"[-1]           # (from end)
"o"
>>> "hello"[1:4]          # slicing
"ell"
>>> len("hello")          # size
5
>>> "hello" < "jello"     # comparison
True
>>> "e" in "hello"        # search
True
```

Container types: Lists

- Flexible arrays

```
>>> a = [99, "bottles of beer", ["on", "the", "wall"]]
>>> type(a)
```

- Same operators as for strings

```
>>> a+b, a*3, a[0], a[-1], a[1:], len(a)
```

- Item and slice assignment

```
>>> a[0] = 98; print a

>>> a[1:2] = ["bottles", "of", "beer"]; print a

>>> del a[-1]
```

Container types: Lists

- Flexible arrays

```
>>> a = [99, "bottles of beer", ["on", "the", "wall"]]
>>> type(a)
<type, 'list'>
```

- Same operators as for strings

```
>>> a+b, a*3, a[0], a[-1], a[1:], len(a)
```

- Item and slice assignment

```
>>> a[0] = 98; print a

>>> a[1:2] = ["bottles", "of", "beer"]; print a

>>> del a[-1]
```

Container types: Lists

- Flexible arrays

```
>>> a = [99, "bottles of beer", ["on", "the", "wall"]]
>>> type(a)
<type, 'list'>
```

- Same operators as for strings

```
>>> a+b, a*3, a[0], a[-1], a[1:], len(a)
```

- Item and slice assignment

```
>>> a[0] = 98; print a
[98, "bottles of beer", ["on", "the", "wall"]]
>>> a[1:2] = ["bottles", "of", "beer"]; print a
[98, "bottles", "of", "beer", ["on", "the", "wall"]]
>>> del a[-1]
[98, "bottles", "of", "beer"]
```

Container types: Lists

```
>>> a = range(5); print a

>>> a.append(5); print a

>>> b = a.pop(); print b, a

>>> a.insert(0, 5.5); print a

>>> v = a.pop(0); print v

>>> a.reverse(); print a

>>> a.sort(); print a
```

Container types: Lists

```
>>> a = range(5); print a
[0,1,2,3,4]
>>> a.append(5); print a

>>> b = a.pop(); print b, a

>>> a.insert(0, 5.5); print a

>>> v = a.pop(0); print v

>>> a.reverse(); print a

>>> a.sort(); print a
```

Container types: Lists

```
>>> a = range(5); print a
[0,1,2,3,4]
>>> a.append(5); print a
[0,1,2,3,4,5]
>>> b = a.pop(); print b, a

>>> a.insert(0, 5.5); print a

>>> v = a.pop(0); print v

>>> a.reverse(); print a

>>> a.sort(); print a
```

Container types: Lists

```
>>> a = range(5); print a
[0,1,2,3,4]
>>> a.append(5); print a
[0,1,2,3,4,5]
>>> b = a.pop(); print b, a
5 [0,1,2,3,4]
>>> a.insert(0, 5.5); print a

>>> v = a.pop(0); print v

>>> a.reverse(); print a

>>> a.sort(); print a
```

Container types: Lists

```
>>> a = range(5); print a
[0,1,2,3,4]
>>> a.append(5); print a
[0,1,2,3,4,5]
>>> b = a.pop(); print b, a
5 [0,1,2,3,4]
>>> a.insert(0, 5.5); print a
[5.5,0,1,2,3,4]
>>> v = a.pop(0); print v


>>> a.reverse(); print a


>>> a.sort(); print a
```

Container types: Lists

```
>>> a = range(5); print a
[0,1,2,3,4]
>>> a.append(5); print a
[0,1,2,3,4,5]
>>> b = a.pop(); print b, a
5 [0,1,2,3,4]
>>> a.insert(0, 5.5); print a
[5.5,0,1,2,3,4]
>>> v = a.pop(0); print v
5.5
>>> a.reverse(); print a


>>> a.sort(); print a
```

Container types: Lists

```
>>> a = range(5); print a
[0,1,2,3,4]
>>> a.append(5); print a
[0,1,2,3,4,5]
>>> b = a.pop(); print b, a
5 [0,1,2,3,4]
>>> a.insert(0, 5.5); print a
[5.5,0,1,2,3,4]
>>> v = a.pop(0); print v
5.5
>>> a.reverse(); print a
[4,3,2,1,0]
>>> a.sort(); print a
```

Container types: Lists

```
>>> a = range(5); print a
[0,1,2,3,4]
>>> a.append(5); print a
[0,1,2,3,4,5]
>>> b = a.pop(); print b, a
5 [0,1,2,3,4]
>>> a.insert(0, 5.5); print a
[5.5,0,1,2,3,4]
>>> v = a.pop(0); print v
5.5
>>> a.reverse(); print a
[4,3,2,1,0]
>>> a.sort(); print a
[0,1,2,3,4]
```

Container types: Other

- Tuple : an immutable list

```
>>> a = (99, "bottles of beer", ["on", "the", "wall"],)
>>> a[2][0] = "in"; print a

>>> a[0] = 0
```

- Dictionaries : list indexed by any immutable object

```
>>> a = { 1 : "un", 12. : "douze", "glob" : 13 }
>>> print a[1], a[12.], a["glob"]

>>> new_key = 27; a[new_key] = ["2", "7"]; print a[27]

>>> a[ ["2", "7"] ] = 0

>>> print a.keys(), a.values(), a.has_key(12)
```

Container types: Other

- Tuple : an immutable list

```
>>> a = (99, "bottles of beer", ["on", "the", "wall"],)
>>> a[2][0] = "in"; print a
(99, 'bottles of beer', ['in', 'the', 'wall'])
>>> a[0] = 0
```

- Dictionaries : list indexed by any immutable object

```
>>> a = { 1 : "un", 12. : "douze", "glob" : 13 }
>>> print a[1], a[12.], a["glob"]

>>> new_key = 27; a[new_key] = ["2", "7"]; print a[27]

>>> a[ ["2", "7"] ] = 0

>>> print a.keys(), a.values(), a.has_key(12)
```

Container types: Other

- Tuple : an immutable list

```
>>> a = (99, "bottles of beer", ["on", "the", "wall"],)
>>> a[2][0] = "in"; print a
(99, 'bottles of beer', ['in', 'the', 'wall'])
>>> a[0] = 0
TypeError: 'tuple' object does not support assignment
```

- Dictionaries : list indexed by any immutable object

```
>>> a = { 1 : "un", 12. : "douze", "glob" : 13 }
>>> print a[1], a[12.], a["glob"]

>>> new_key = 27; a[new_key] = ["2", "7"]; print a[27]

>>> a[ ["2", "7"] ] = 0

>>> print a.keys(), a.values(), a.has_key(12)
```

Container types: Other

- Tuple : an immutable list

```
>>> a = (99, "bottles of beer", ["on", "the", "wall"],)
>>> a[2][0] = "in"; print a
(99, 'bottles of beer', ['in', 'the', 'wall'])
>>> a[0] = 0
TypeError: 'tuple' object does not support assignment
```

- Dictionaries : list indexed by any immutable object

```
>>> a = { 1 : "un", 12. : "douze", "glob" : 13 }
>>> print a[1], a[12.], a["glob"]
un douze 13
>>> new_key = 27; a[new_key] = ["2", "7"]; print a[27]

>>> a[ ["2", "7"] ] = 0

>>> print a.keys(), a.values(), a.has_key(12)
```

Container types: Other

- Tuple : an immutable list

```
>>> a = (99, "bottles of beer", ["on", "the", "wall"],)
>>> a[2][0] = "in"; print a
(99, 'bottles of beer', ['in', 'the', 'wall'])
>>> a[0] = 0
TypeError: 'tuple' object does not support assignment
```

- Dictionaries : list indexed by any immutable object

```
>>> a = { 1 : "un", 12. : "douze", "glob" : 13 }
>>> print a[1], a[12.], a["glob"]
un douze 13
>>> new_key = 27; a[new_key] = ["2", "7"]; print a[27]
['2, '7']
>>> a[ ["2", "7"] ] = 0


>>> print a.keys(), a.values(), a.has_key(12)
```

Container types: Other

- Tuple : an immutable list

```
>>> a = (99, "bottles of beer", ["on", "the", "wall"],)
>>> a[2][0] = "in"; print a
(99, 'bottles of beer', ['in', 'the', 'wall'])
>>> a[0] = 0
TypeError: 'tuple' object does not support assignment
```

- Dictionaries : list indexed by any immutable object

```
>>> a = { 1 : "un", 12. : "douze", "glob" : 13 }
>>> print a[1], a[12.], a["glob"]
un douze 13
>>> new_key = 27; a[new_key] = ["2", "7"]; print a[27]
['2, '7']
>>> a[ ["2", "7"] ] = 0
TypeError: unhashable type: 'list'
>>> print a.keys(), a.values(), a.has_key(12)
```

Container types: Other

- Tuple : an immutable list

```
>>> a = (99, "bottles of beer", ["on", "the", "wall"],)
>>> a[2][0] = "in"; print a
(99, 'bottles of beer', ['in', 'the', 'wall'])
>>> a[0] = 0
TypeError: 'tuple' object does not support assignment
```

- Dictionaries : list indexed by any immutable object

```
>>> a = { 1 : "un", 12. : "douze", "glob" : 13 }
>>> print a[1], a[12.], a["glob"]
un douze 13
>>> new_key = 27; a[new_key] = ["2", "7"]; print a[27]
['2, '7']
>>> a[ ["2", "7"] ] = 0
TypeError: unhashable type: 'list'
>>> print a.keys(), a.values(), a.has_key(12)
[1, 'glob', 27, 12.] ['un', 13, ['2', '7'], 'douze'] True
```

Variables

- No need to declare
- Need to assign (initialize)

  use of uninitialized variable raises exception
- Not typed

  ```
  >>> var = 12**2
  >>> var = "hello world"
  ```
- ***Everything*** is a variable:
  - functions
  - modules
  - classes

Reference semantics

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

- Very useful; but beware :

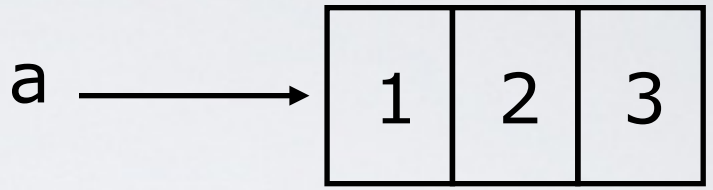
```
>>> a = [1, 2, 3]
>>> b = a
>>> a.append(4)
>>> print b
[1, 2, 3, 4]
```

- Use of deep copy to avoid this:

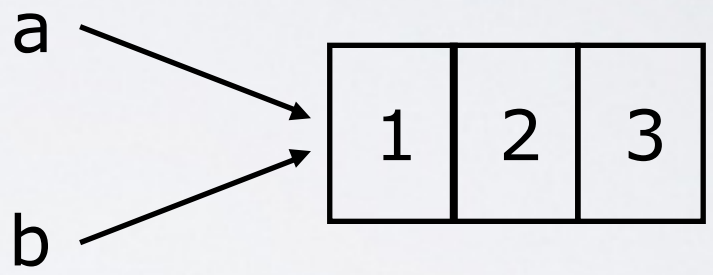
```
>>> import copy
>>> c = copy.deepcopy(a)
```

Changing a shared list

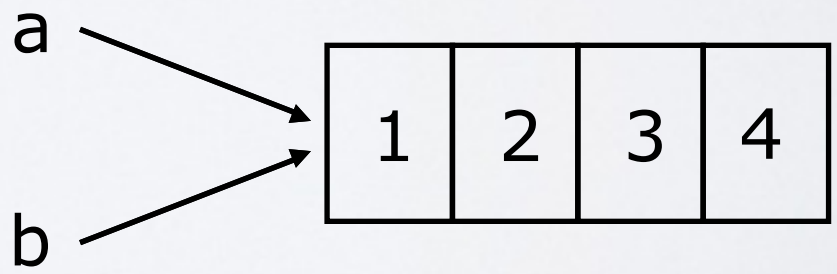
a = [1, 2, 3]



b = a



a.append(4)

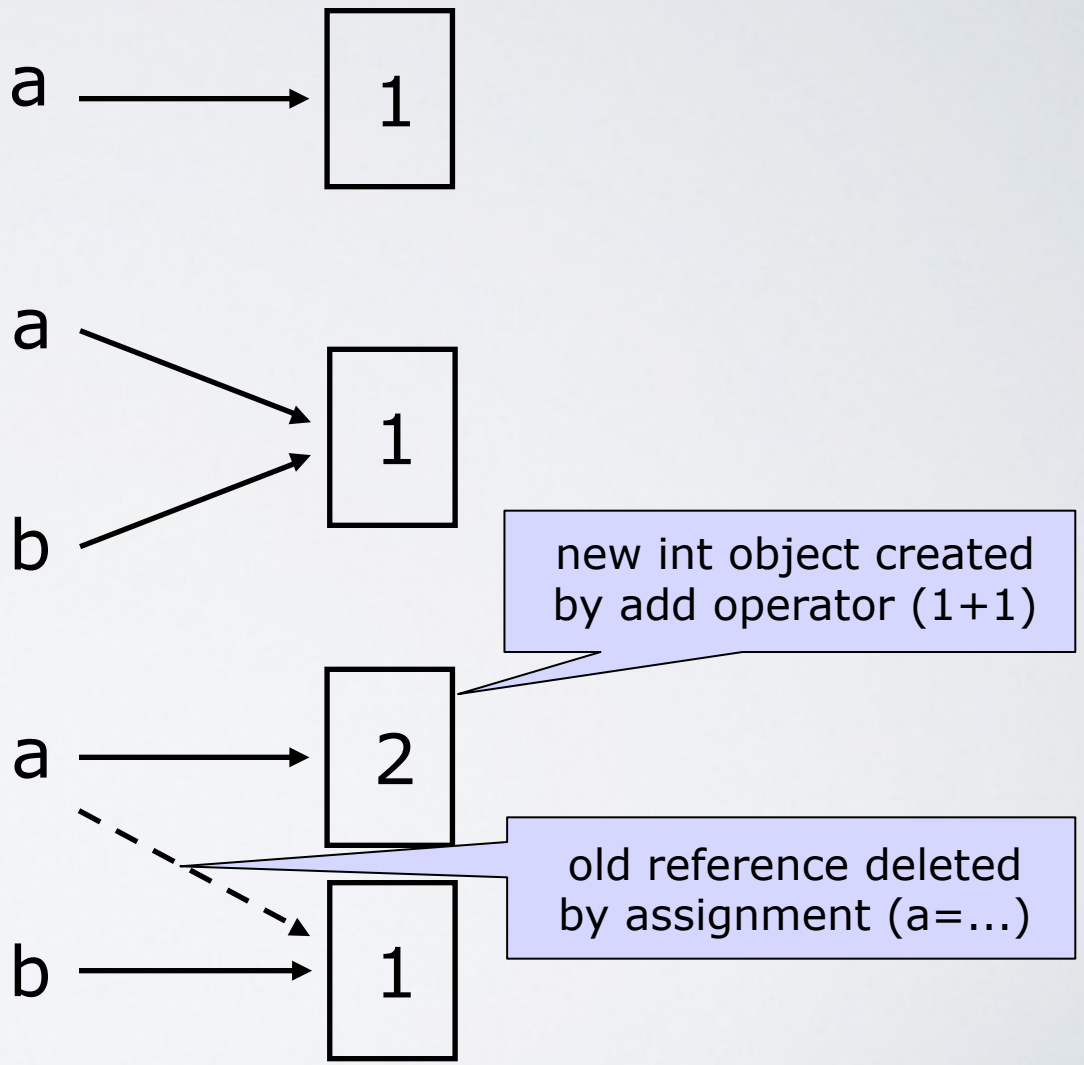


Changing an integer

a = 1

b = a

a = a+1



Control structures

- Condition:

```
>>> a = 12.
>>> if a <= 6.:
...         print "a is inferior to 6"
...     elif a%2 == 0:
...         print "a is even "
...     else:
...         print "a is ", a
```

- For loop:

```
>>> a = range(5)
>>> b = []
>>> print a, b
[0, 1, 2, 3, 4] []
>>> for i in a:
...         b.append(3*i)
>>> print b
```

- While loop:

```
>>> x = 1.
>>> while x < 10:
...         x += 2*x
>>> print x
```

- Break, continue (and yield)

Control structures

- Condition:

```
>>> a = 12.
>>> if a <= 6.:
...         print "a is inferior to 6"
...     elif a%2 == 0:
...         print "a is even "
...     else:
...         print "a is ", a
a is 12
```

- While loop:

```
>>> x = 1.
>>> while x < 10:
...         x += 2*x
>>> print x
```

- For loop:

```
>>> a = range(5)
>>> b = []
>>> print a, b
[0, 1, 2, 3, 4] []
>>> for i in a:
...         b.append(3*i)
>>> print b
```

- Break, continue (and yield)

Control structures

- Condition:

```
>>> a = 12.
>>> if a <= 6.:
...         print "a is inferior to 6"
...     elif a%2 == 0:
...         print "a is even "
...     else:
...         print "a is ", a
a is 12
```

- While loop:

```
>>> x = 1.
>>> while x < 10:
...         x += 2*x
>>> print x
27
```

- For loop:

```
>>> a = range(5)
>>> b = []
>>> print a, b
[0, 1, 2, 3, 4] []
>>> for i in a:
...         b.append(3*i)
>>> print b
```

- Break, continue (and yield)

Control structures

- Condition:

```
>>> a = 12.
>>> if a <= 6.:
...         print "a is inferior to 6"
...     elif a%2 == 0:
...         print "a is even "
...     else:
...         print "a is ", a
a is 12
```

- While loop:

```
>>> x = 1.
>>> while x < 10:
...         x += 2*x
>>> print x
27
```

- For loop:

```
>>> a = range(5)
>>> b = []
>>> print a, b
[0, 1, 2, 3, 4] []
>>> for i in a:
...         b.append(3*i)
>>> print b
[0, 3, 6, 9, 12]
```

- Break, continue (and yield)

Functions, procedures

- Procedure:

```
>>> def printing_procedure(something, val):
...         """ procedure print val times something (if val inferior to 5)"""
...         if val >=5 : return
...         print somehting*val
...         something = ""
>>> help(printing_procedure)


>>> s = "hello"
>>> printing_procedure(s, 3)


>>> print s


>>> printing_procedure(s,6)
```

Functions, procedures

- Procedure:

```
>>> def printing_procedure(something, val):
...         """ procedure print val times something (if val inferior to 5)"""
...         if val >=5 : return
...         print somehting*val
...         something = ""
>>> help(printing_procedure)
--- display help... type ‘q’ to quit help ---
>>> s = "hello"
>>> printing_procedure(s, 3)

>>> print s

>>> printing_procedure(s,6)
```

Functions, procedures

- Procedure:

```
>>> def printing_procedure(something, val):
...         """ procedure print val times something (if val inferior to 5)"""
...         if val >=5 : return
...         print somehting*val
...         something = ""
>>> help(printing_procedure)
--- display help... type ‘q’ to quit help ---
>>> s = "hello"
>>> printing_procedure(s, 3)
hellohellohello
>>> print s

>>> printing_procedure(s,6)
```

Functions, procedures

- Procedure:

```
>>> def printing_procedure(something, val):
...         """ procedure print val times something (if val inferior to 5)"""
...         if val >=5 : return
...         print somehting*val
...         something = ""
>>> help(printing_procedure)
--- display help... type ‘q’ to quit help ---
>>> s = "hello"
>>> printing_procedure(s, 3)
hellohellohello
>>> print s
hello
>>> printing_procedure(s,6)
```

Functions, procedures

- Procedure:

```
>>> def printing_procedure(something, val):
...         """ procedure print val times something (if val inferior to 5)"""
...         if val >=5 : return
...         print somehting*val
...         something = ""
>>> help(printing_procedure)
--- display help... type ‘q’ to quit help ---
>>> s = "hello"
>>> printing_procedure(s, 3)
hellohellohello
>>> print s
hello
>>> printing_procedure(s,6)
>>>
```

Functions, procedures

- Function:

```
>>> def printing_function(something):
...         """ print something as many times as number of characters
...         in something string and return it"""
...         val = len(somehting)
...         print something*val
...         return val
>>> s = "hell"
>>> v = printing_function(s)


>>> print v
```

Functions, procedures

- Function:

```
>>> def printing_function(something):
...         """ print something as many times as number of characters
...         in something string and return it"""
...         val = len(somehting)
...         print something*val
...         return val
>>> s = "hell"
>>> v = printing_function(s)
hellhellhellhell
>>> print v
```

Functions, procedures

- Function:

```
>>> def printing_function(something):
...         """ print something as many times as number of characters
...         in something string and return it"""
...         val = len(somehting)
...         print something*val
...         return val
>>> s = "hell"
>>> v = printing_function(s)
hellhellhellhell
>>> print v
4
```

Using classes

- A class is a type definition which specifies attributs and methods working on the attributs
- In interactive mode, remember the ‘help’ function
- To create an instance, simply call the class object:

```
>>> x = list()
>>> print type(x), x
```

- To use methods of the instance, call using dot notation:

```
>>> x.append(1.)
>>> print x
```

- To inspect instance variables, use dot notation:

```
>>> x.__hash__
```

Using classes

- A class is a type definition which specifies attributs and methods working on the attributs
- In interactive mode, remember the ‘help’ function
- To create an instance, simply call the class object:

```
>>> x = list()
>>> print type(x), x
<type ‘list’> []
```

- To use methods of the instance, call using dot notation:

```
>>> x.append(1.)
>>> print x
```

- To inspect instance variables, use dot notation:

```
>>> x.__hash__
```

Using classes

- A class is a type definition which specifies attributs and methods working on the attributs
- In interactive mode, remember the ‘help’ function
- To create an instance, simply call the class object:

```
>>> x = list()
>>> print type(x), x
<type ‘list’> []
```

- To use methods of the instance, call using dot notation:

```
>>> x.append(1.)
>>> print x
[1.]
```

- To inspect instance variables, use dot notation:

```
>>> x.__hash__
```

Using classes

- A class is a type definition which specifies attributs and methods working on the attributs
- In interactive mode, remember the ‘help’ function
- To create an instance, simply call the class object:

```
>>> x = list()
>>> print type(x), x
<type ‘list’> []
```

- To use methods of the instance, call using dot notation:

```
>>> x.append(1.)
>>> print x
[1.]
```

- To inspect instance variables, use dot notation:

```
>>> x.__hash__
None
```

Scripts

- Series of commands in *bar*.py file:

  ```
  print "bar"
  x = 12
  ```

- Execute the commands from shell:

  ```
  $ python bar.py
  bar
  $
  ```

- Execute the commands from shell but stay in interpretor:

  ```
  $ python -i bar.py
  bar
  >>> x + 2
  14
  ```

- Execute within python:

  ```
  >>> import bar
  ```

Modules

- Collection of stuff in *foo*.py file
  - functions, classes, variables

- Importing modules:

```
>>> import math
>>> math.pi
3.141592653589793
>>> from math import cos
>>> cos(math.pi)
-1.0
>>> from math import sin as sinus
>>> sinus(0.)
0.0
```

- Some common modules: copy, string, math, random, numpy, matplotlib, scipy

Numpy

- Defines the ‘array’ type: list with all element of the same type

```
>>> import numpy
>>> a = numpy.array([1,2,3,4])
>>> type(a); a.dtype

>>> a[0] = 0.5; print a
```

- Cannot change size or type afterward
- But can change shape of the array:

```
>>> b = a.reshape([2,2]); print b


>>> a.shape = [2,2]; print a
```

Numpy

- Defines the ‘array’ type: list with all element of the same type

```
>>> import numpy
>>> a = numpy.array([1,2,3,4])
>>> type(a); a.dtype
<type ‘numpy.ndarray’> dtype(‘int64’)
>>> a[0] = 0.5; print a
```

- Cannot change size or type afterward
- But can change shape of the array:

```
>>> b = a.reshape([2,2]); print b
```

```
>>> a.shape = [2,2]; print a
```

Numpy

- Defines the ‘array’ type: list with all element of the same type

```
>>> import numpy
>>> a = numpy.array([1,2,3,4])
>>> type(a); a.dtype
<type ‘numpy.ndarray’> dtype(‘int64’)
>>> a[0] = 0.5; print a
[0 2 3 4]
```

- Cannot change size or type afterward
- But can change shape of the array:

```
>>> b = a.reshape([2,2]); print b
```

```
>>> a.shape = [2,2]; print a
```

Numpy

- Defines the ‘array’ type: list with all element of the same type

```
>>> import numpy
>>> a = numpy.array([1,2,3,4])
>>> type(a); a.dtype
<type ‘numpy.ndarray’> dtype(‘int64’)
>>> a[0] = 0.5; print a
[0 2 3 4]
```

- Cannot change size or type afterward
- But can change shape of the array:

```
>>> b = a.reshape([2,2]); print b
[ [0 2]
  [3 4] ]
>>> a.shape = [2,2]; print a
```

Numpy

- Defines the ‘array’ type: list with all element of the same type

```
>>> import numpy
>>> a = numpy.array([1,2,3,4])
>>> type(a); a.dtype
<type ‘numpy.ndarray’> dtype(‘int64’)
>>> a[0] = 0.5; print a
[0 2 3 4]
```

- Cannot change size or type afterward
- But can change shape of the array:

```
>>> b = a.reshape([2,2]); print b
[ [0 2]
  [3 4] ]
>>> a.shape = [2,2]; print a
[ [0 2]
  [3 4] ]
```

Numpy

- Create an array from a list:

```
>>> numpy.array([1,2,3,4,5], dtype=numpy.float)
```

- Create an array from a function:

```
>>> numpy.arange(3)

>>> numpy.zeros(3)

>>> numpy.eye(2)

>>> numpy.ones(3)

>>> numpy.diag([3,7])

>>> numpy.random.rand(2)
```

Numpy

- Create an array from a list:

```
>>> numpy.array([1,2,3,4,5], dtype=numpy.float)
[1. 2. 3. 4. 5.]
```

- Create an array from a function:

```
>>> numpy.arange(3)

>>> numpy.zeros(3)

>>> numpy.eye(2)

>>> numpy.ones(3)

>>> numpy.diag([3,7])

>>> numpy.random.rand(2)
```

Numpy

- Create an array from a list:

```
>>> numpy.array([1,2,3,4,5], dtype=numpy.float)
[1. 2. 3. 4. 5.]
```

- Create an array from a function:

```
>>> numpy.arange(3)
[0 1 2]
>>> numpy.zeros(3)

>>> numpy.eye(2)

>>> numpy.ones(3)

>>> numpy.diag([3,7])

>>> numpy.random.rand(2)
```

Numpy

- Create an array from a list:

```
>>> numpy.array([1,2,3,4,5], dtype=numpy.float)
[1. 2. 3. 4. 5.]
```

- Create an array from a function:

```
>>> numpy.arange(3)
[0 1 2]
>>> numpy.zeros(3)
[0 0 0]
>>> numpy.eye(2)

>>> numpy.ones(3)

>>> numpy.diag([3,7])

>>> numpy.random.rand(2)
```

Numpy

- Create an array from a list:

```
>>> numpy.array([1,2,3,4,5], dtype=numpy.float)
[1. 2. 3. 4. 5.]
```

- Create an array from a function:

```
>>> numpy.arange(3)
[0 1 2]
>>> numpy.zeros(3)
[0 0 0]
>>> numpy.eye(2)
[ [1 0] [0 1] ]
>>> numpy.ones(3)

>>> numpy.diag([3,7])

>>> numpy.random.rand(2)
```

Numpy

- Create an array from a list:

```
>>> numpy.array([1,2,3,4,5], dtype=numpy.float)
[1. 2. 3. 4. 5.]
```

- Create an array from a function:

```
>>> numpy.arange(3)
[0 1 2]
>>> numpy.zeros(3)
[0 0 0]
>>> numpy.eye(2)
[ [1 0] [0 1] ]
>>> numpy.ones(3)
[1 1 1]
>>> numpy.diag([3,7])

>>> numpy.random.rand(2)
```

Numpy

- Create an array from a list:

```
>>> numpy.array([1,2,3,4,5], dtype=numpy.float)
[1. 2. 3. 4. 5.]
```

- Create an array from a function:

```
>>> numpy.arange(3)
[0 1 2]
>>> numpy.zeros(3)
[0 0 0]
>>> numpy.eye(2)
[ [1 0] [0 1] ]
>>> numpy.ones(3)
[1 1 1]
>>> numpy.diag([3,7])
[ [3 0] [0 7] ]
>>> numpy.random.rand(2)
```

Numpy

- Create an array from a list:

```
>>> numpy.array([1,2,3,4,5], dtype=numpy.float)
[1. 2. 3. 4. 5.]
```

- Create an array from a function:

```
>>> numpy.arange(3)
[0 1 2]
>>> numpy.zeros(3)
[0 0 0]
>>> numpy.eye(2)
[ [1 0] [0 1] ]
>>> numpy.ones(3)
[1 1 1]
>>> numpy.diag([3,7])
[ [3 0] [0 7] ]
>>> numpy.random.rand(2)
[0.26230435 0.97418097]
```

Numpy

- Additions:

```
>>> a = numpy.array([1,2]); b = numpy.array([3,4]); a + b
```

```
>>> a + b + 6
```

- Multiplication:

```
>>> a * 2
```

```
>>> 1. / a
```

```
>>> a ** 2
```

- Matrix vector product:

```
>>> numpy.dot( numpy.array([[1,2],[3,4]]), a)
```

Numpy

- Additions:

```
>>> a = numpy.array([1,2]); b = numpy.array([3,4]); a + b
[4 6]
>>> a + b + 6
```

- Multiplication:

```
>>> a * 2

>>> 1. / a

>>> a ** 2
```

- Matrix vector product:

```
>>> numpy.dot( numpy.array([[1,2],[3,4]]), a)
```

Numpy

- Additions:

```
>>> a = numpy.array([1,2]); b = numpy.array([3,4]); a + b
[4 6]
>>> a + b + 6
[10 12]
```

- Multiplication:

```
>>> a * 2

>>> 1. / a

>>> a ** 2
```

- Matrix vector product:

```
>>> numpy.dot( numpy.array([[1,2],[3,4]]), a)
```

Numpy

- Additions:

  ```
  >>> a = numpy.array([1,2]); b = numpy.array([3,4]); a + b
  [4 6]
  >>> a + b + 6
  [10 12]
  ```

- Multiplication:

  ```
  >>> a * 2
  [2 4]
  >>> 1. / a

  >>> a ** 2
  ```

- Matrix vector product:

  ```
  >>> numpy.dot( numpy.array([[1,2],[3,4]]), a)
  ```

Numpy

- Additions:

```
>>> a = numpy.array([1,2]); b = numpy.array([3,4]); a + b
[4 6]
>>> a + b + 6
[10 12]
```

- Multiplication:

```
>>> a * 2
[2 4]
>>> 1. / a
[1. 0.5]
>>> a ** 2
```

- Matrix vector product:

```
>>> numpy.dot( numpy.array([[1,2],[3,4]]), a)
```

Numpy

- Additions:

```
>>> a = numpy.array([1,2]); b = numpy.array([3,4]); a + b
[4 6]
>>> a + b + 6
[10 12]
```

- Multiplication:

```
>>> a * 2
[2 4]
>>> 1. / a
[1. 0.5]
>>> a ** 2
[1 4]
```

- Matrix vector product:

```
>>> numpy.dot( numpy.array([[1,2],[3,4]]), a)
```

Numpy

- Additions:

```
>>> a = numpy.array([1,2]); b = numpy.array([3,4]); a + b
[4 6]
>>> a + b + 6
[10 12]
```

- Multiplication:

```
>>> a * 2
[2 4]
>>> 1. / a
[1. 0.5]
>>> a ** 2
[1 4]
```

- Matrix vector product:

```
>>> numpy.dot( numpy.array([[1,2],[3,4]]), a)
[5 11]
```