Python

Very short 12-slides-introduction

– EFREI –
– ESTIA –
Guillaume Rivière
Last update: March 2018

Python language

- A few words about Python
  - 1991, Guido van Rossum (NL)
  - Object-oriented, Multi-paradigm, Imperative
  - Interpreted
  - Garbage collector
  - Versions 2.7.x and 3.3.x
  - Interpreter for Windows, Mac, Linux, Unix, …
  - Blocks are identified with whitespaces indentation

Input, Output, Declare a function

test.py

```
def max(a, b):
    """ Return the maximum of the two parameters """
    if a > b:
        return a
    else:
        return b

x = input('Give the number X: ')
y = input('Give the number Y: ')
print 'The maximum is ' + str(max(x,y))
```

```
[user@debian]$ python test.py
Give the number X: 2
Give the number Y: 8
The maximum is 8
```

```
[user@debian]$ python test.py
Give the number X: 2
Give the number Y: 3.5
The maximum is 3.5
```

Comments

- Comment until end of line: hash symbol

```
print 'hello' # This is a one-line comment
```

- Comment several lines: triple simple-quotes

```
''' This is a
multi-line
Comment
'''
```

- Docstring: triple double-quotes

```
def double (x):
       """ This function computes the double of a value """
       return 2*x
```

Declare a class

rectangle.py

```
class Rectangle:
    """ A simple rectangle class """
    width = 0
    height = 0
    def area(self):
        return self.width * self.height
    def perimeter(self):
        return 2*self.width + 2*self.height

r1 = Rectangle()
r1.width = 2.3
r1.height = 3.4
print 'The perimeter is', r1.perimeter()
print 'The area is', r1.area()
```

```
[user@debian]$ python main.py
The perimeter is 11.4
The area is 7.82
```

Import

rectangle.py

```
class Rectangle:
    """ A simple rectangle class """
    def __init__(self, w=0, h=0):
        self.width = w
        self.height = h
    def area(self):
        return self.width * self.height
    def perimeter(self):
        return 2*self.width + 2*self.height
```

main.py

```
import rectangle

r1 = rectangle.Rectangle()
print 'Perimeter =', r1.perimeter()
print 'Area =', r1.area()

r2 = rectangle.Rectangle(2, 3)
print 'Perimeter =', r2.perimeter()
print 'Area =', r2.area()
```

```
[user@debian]$ python main.py
The perimeter is  0
The area is  0
The perimeter is  10
The area is  6
```

Heritage

rectangle.py

```
class Rectangle:
    """ A simple rectangle class """
    def __init__(self, w=0, h=0):
        self.width = w
        self.height = h
    def area(self):
        return self.width * self.height
    def perimeter(self):
        return 2*self.width + 2*self.height

class GraphicRectangle(Rectangle):
    """ A graphic rectangle class """
    def __init__(self, w, h, x=0, y=0, color='white'):
        Rectangle.__init__(self, w, h)
        self.x = x
        self.y = y
        self.color = color
    def move(self, dx, dy):
        self.x += dx
        self.y += dy
```

Data structures

- All data structures can contain mixed types

Type	Description	Example
List	A mutable list	`[4.0, 'string', True]`
Tuple	An **immutable** list	`(4.0, 'string', True)`
Set	**Unordered** set, contains **no duplicates**	`{4.0, 'string', True}`
Dict	A mutable **associative array**	`{'key1': 1.0, 3: False}`

Operations on a Dictonnary

- Dict data structures are mutable **associative array**

test.py
```
d = {'key1': 1.0, 3: False}
print d
print d['key1']
print d[3]

d[3] = 'User'
print d

d[4] = 'Name'
print d

del d['key1']
print d

d.clear()
print d
```

```
[user@debian]$ python test.py
{'key1': 1.0, 3: False}
1.0
False
{'key1': 1.0, 3: 'User'}
{'key1': 1.0, 3: 'User', 4: 'Name'}
{3: 'User', 4: 'Name'}
{}
```

Operations on a List

- List data structures are **mutable** list

test.py
```
li = [4.0, 'string', True]
print li

li.append('new')
print li
li.insert(2, 'new')
print li

print li.index(4.0)
print li.index('new')
print 'toto' in li

li.remove(4.0)
print li
li.remove('new')
print li
print li.pop()
print li
```

```
[user@debian]$ python test.py
[4.0, 'string', True]
[4.0, 'string', True, 'new']
[4.0, 'string', 'new', True, 'new']
0
2
False
['string', 'new', True, 'new']
['string', True, 'new']
new
['string', True]
```

Operations on a List

test.py
```
li = ['a', 'b', 'c']
print li

li.extend(['d', 'e'])
print li

print li[0]
print li[2]
print li[-1]
print li[-3]

print li[1:3]
print li[1:-1]
print li[0:3]
```

```
[user@debian]$ python test.py
['a', 'b', 'c']
['a', 'b', 'c', 'd', 'e']
a
c
e
c
['b', 'c']
['b', 'c', 'd']
['a', 'b', 'c']
```

Operations on a List

test.py
```
li = [1, 2] * 3
print li

li = li + ['a', 'b']
print li

li += ['c', 'd']
print li
```

```
[user@debian]$ python test.py
[1, 2, 1, 2, 1, 2]
[1, 2, 1, 2, 1, 2, 'a', 'b']
[1, 2, 1, 2, 1, 2, 'a', 'b', 'c', 'd']
```

Operations on a Tuple

- A tupple is an **immutable** list
  - Once it is created, it can never be changed !

test.py

```
t = ('a', 'b', 'c', 'd', 'e')
print t

print t[0]
print t[-1]
print t[1:3]
```

```
[user@debian]$ python test.py
('a', 'b', 'c', 'd', 'e')
a
e
('b', 'c')
```

Documentation

- Experienced programmers can directly start learning Python with the free book:

  “Dive Into Python”

  The last version is available at

  `http://diveintopython.org/`