Projet1 – TP1 : Introduction to Python

Marin Bougeret, Bogdan Pasca, Kevin Perrot
{ Bogdan.Pasca } @ens-lyon.fr

September 13, 2010

Python in a flash !

What ?

History – Invented in 1990 by Guido Van Rossum
- The name "Python" stems from "Monty Python's Flying Circus"
- Intended to be a scripting language on Amoeba OS
- First public release was in 1991

Goals – Designed to be simple yet powerful
- Allow modular programming
- Great emphasis on readability
- Rapid application development
- Easy to embed in and extend with other languages

How ?

SHELL interactive – perfect for : learning a language, testing a library, testing *your* modules
- to start Python simply type `python` in your terminal
- to execute a Python file (*.py*) feed the file as a parameter : `python yourfile.py`

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

(built in)Data structures : Decimal, Octal, Hexa, Complex(.5+4j),Long
- **Strings** :

```
>>> # Concatenation
>>> "Hello"+"World"
'HelloWorld'
>>> # Repetition
>>> "Arenaire"*3
'ArenaireArenaireArenaire'
>>> # Indexing
>>> "Arenaire"[0]
```

```
'A'
>>> # Slicing
>>> "Arenaire"[1:3]
'ren'
>>> # Size
>>> len("Arenaire")
8
>>> # Comparison
>>> "Arenaire" < "arenaire"
True
>>> # Search
>>> "A" in "Arenaire"
True
```

- **Lists** :

```
>>> a=[99,"bottles of beer",["on","the","wall"]]
>>> b=[98,"bottles of water"]
>>> a+b
[99, 'bottles of beer', ['on', 'the', 'wall'], 98, 'bottles of water']
```

  - Same operators as for strings : a+b, a*3, a[0], a[-1], a[1 :], len(a)
  - Item and slice assignment

```
>>> a[0]
99
>>> a[1:2]
['bottles of beer']
>>> del a[-1]
>>> print a
[99, 'bottles of beer']
```

  - more list operations :

```
>>> a = range(5)
>>> print a
[0, 1, 2, 3, 4]
>>> a.append(5)
>>> print a
[0, 1, 2, 3, 4, 5]
>>> a.pop()
5
>>> print a
[0, 1, 2, 3, 4]
>>> a.insert(0,42)
>>> print a
[42, 0, 1, 2, 3, 4]
>>> a.pop(0)
42
>>> print a
[0, 1, 2, 3, 4]
>>> a.reverse()
```

```
>>> print a
[4, 3, 2, 1, 0]
>>> a.sort()
>>> print a
[0, 1, 2, 3, 4]
```

- **Tuples** :
  - `key = (lastname, firstname)`
  - `point = x, y, z # parentheses optional`
  - `x, y, z = point # unpack`
  - `lastname = key[0]`
  - `singleton = (1,) # trailing comma!!!`
  - `empty = () # parentheses!`
  - tuples vs. lists ; tuples immutable

Variables – No need to declare
- Need to assign (initialize)
- use of uninitialized variable raises exception
- Not typed

```
>>> friendly=0
>>> if friendly: greeting="Hello"
else: greeting = 12**2

>>> print greeting
144
```

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 !
- Example :

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

Control structures – `if condition: statements`
`[elif condition: statements]*`
`[else: statements]`
- `while condition: statements`
- `for var in sequence: statements`
- `break`
- `continue`

```
>>> # Fibonacci series
>>> a=0
>>> b=1
>>> while b<100:
print b
```

```
a,b=b,a+b


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

Procedures and functions – General form :

```
def name(arg1, arg2, âĂę)
  Statements
  return  # from procedure OR
  return expression  # from function
```

- Procedures can omit any *return*
- example :

```
>>> def gcd(a,b):
"greatest common divisor"
while a!=0:
a,b= b%a,a # parallel assignement
return b

>>> gcd.__doc__
'greatest common divisor'
>>> gcd(12,20)
4
```

Classes and objects – : Classes

```
class ClassName:
statements

class ClassName(BaseClass1, BaseClass2)
statements
```

- Objects : `x = ClassName()` creates a new instance of class ClassName and assigns it to the variable x
- Example :

```
>>> class Stack:
    "A well-known data structure..."
    def __init__(self): # constructor
        self.items=[]
    def push(self,x):
        self.items.append(x) # the sky is the limit
```

```
    def pop(self):
        x=self.items[-1] # what happens if it's empty?
        del self.items[-1]
        return x
    def empty(self):
        return len(self.items)==0 # Boolean result

>>> # To create an instance
>>> x = Stack()  # no 'new' operator!
>>> x.empty()
True
>>> x.push(1)
>>> x.items
[1]
>>> x.empty()
False
>>> x.push("hello")
>>> x.items
[1, 'hello']
>>> x.pop()
'hello'
```

Getting the grips

1. Write a function that computes the area of a circle given the radius **R** (circle.py)
2. try importing your function using the `import` statement : import circle and try importing it like this : from circle import area. What is the difference ?
3. Try adding 3.1 and 5.6. What is the result ? Why ?
4. Print values with a specified precision : "%f" % 8.7 vs. "%1f" % 8.7
5. Outputting : a="larger", "the result is %s than 4" % a. Try this for different formats : %d, %f,
6. Write a new program which fetches the radius from the user and then computes the circle's area. Use the `radius=int(raw_input("Please enter a radius"))`. What happens if you give the input as a floating point ? Modify the program so that it accepts floating point values for the radius. What happens if you feed a character string instead of int or float ? Modify the program so that it exits gently. Think about using the :

   ```
   try
       statements
   except(ValueError):
       print "you entered invalid input"
   ```

7. Files. Create by hand a file containing numbers on each line. Write a python program that reads the file line by line, converts the values read into floating-point values and writes them in another file such that line k in the second file contains the sum of the elements from 1 :k from the first file. For reading the data from the first file you can either use the

readline function or iterate on the file object itself (for line in f :). For writing data in the file you can use : `print » f, "my data %f" % acc`

References

The course by Eddy Caron 2009 an the references therein.