A short introduction to Python 3

Arnaud Degroote, Pierrick Koch
LAAS-CNRS
April 2013

Why Python ?

- Readability and expressiveness,
  - high level features (lists, sets, mappings,...),
- user friendly
  - automatic allocation and garbage collecting,
  - huge standard library (regexp, numpy,...),
  - dynamic typing,
- glue for many applications
  - Python/C API,
  - Bindings for many tools (CORBA, OpenCV,...)
- Object oriented
- Efficiency
  - Compilation on the fly

Syntax

- One instruction per line, except in case of opening symbol """,(,f,[: line ends up at closing symbol.
- Blocs are defined by 4 space indentations,
- semicolon allows to put several instructions on the same line,
- backslash allows to extend an instruction on the following line.
- # for comments,
- instruction `pass` does nothing

String - literal

- delimited by simple or double quotes
- examples

```
print("string between double quotes.")

print('string between double quotes.')
```

- """allows to define a multiline string.

Numeric types - literal

- integer
  - OS size or long integer without bound,
  - automatic conversion.
- floating point numbers
  - OS size for double
- complex numbers
  - 1.+2.5j
- boolean: True or False
- None : non-typed value meaning no value.

Numeric types - operators

- `+, - * /` the same as in C
- `//` division between integers
- `%` modulo
- `divmod(x,y)` returns a pair `(x//y, x%y)`
- `x**y` the same as `pow(x,y)`

Conversions

- `int(x)` convert into integer,
- `float(x)` convert into float,
- `complex(x,y)` create complex number `x+yj`,
- `str(x)` convert into a string.

Variables

- No declaration. Variables are defined at affectation with =

```
x = 25
text = 'My text'
```

- a variable can change type during execution:
  - `x = 'My text'`
- Multiple affectation
  - `x = y = 0`
- Parallel affectation

```
x, y = 10, 20
```

- reading a non-defined variable raises an exception

```
>>> print(z)
NameError: name 'z' is not defined
```

String

- Object belonging to class `str`.
- `lower()`, `upper()`, `capitalize()` change the case
- `replace(old,new[,count])` replace occurrences of `old` by `new`,
- `find(sub[,start[,end]])` find first occurrence of `sub`.
- `strip([chars])` erase spaces or `chars` at beginning and end of string, also `rstrip`, `lstrip`
- `split(sep)` extract elements of a string separated by `sep`

```
>>> '10,20,30'.split(',')
['10', '20', '30']
```

- `join([strings])` do the opposite operation.

```
>>> ','.join(['10', '20', '30']
'10,20,30'
```

String formatting

- string%parameters
- Parameters can be
  - a value
  - a sequence,
  - a dictionary

```
>>> s = "Mr %s is %i year old."
>>> s%('Dupond', 30)
'Mr Dupond is 30 years old.'
>>> d={'name':'Durand', 'age':45}
>>> s='Mr %(name)s is %(age)d year old'
>>> s%d
'Mr Durand is 45 year old'
```

String formatting

- Formatting flags
  - `%s` display result of `str()`
  - `%r` display result of `repr()`
  - `%d,i` display decimal integer
  - `%f,g,e` display floating point number
  - `%x,X` hexadecimal
  - `%o` octal

Instruction `if`

```
if boolean_expression :
    indented_conditional_instruction
```

or

```
if boolean_expr : conditional_instr
```

- Example

```
if i>8:
    print('i is greater than 8.')
    if i > 22:
        print('i is greater than 22.')
```

Instruction `if`

- Conditional expressions can be built with boolean operator

  `and, or, not`

- with `and` and `or` expressions are evaluated only if necessary.

Instructions `if, elif, else`

```
if X>0:
   print('x positive')
elif X==0:
   print('x equal 0')
else:
   print('x negative')
```

Conditional expression

- value `if` condition `else` other value
- example

```
'positive' if X >= 0 else 'negative'
```

Instruction `while`

- Iterate while a condition is true

```
i=0
while i<5:
    print(i)
    i+=1
else: print('end')
```

- `break` get out of the loop,
- `continue` go to next iteration.

```
0
1
2
3
4
end
```

Instruction `for`

- Iterate over a sequence

```
L=[0,10,20,30]
for e in L:
    print(e)
```

```
0
10
20
30
```

- Iterate over a sequence of integers

```
for i in range(0,5):
    print(i)
```

```
0
1
2
3
4
```

List Comprehensions

- Syntax:

  expression `for` target `in` sequence `if` condition

- "`if` condition" is not mandatory
- Example

```
>>> L=[x**2 for x in range(10) if x%2==0]
>>> L
[0,4,16,36,64]
```

Function

- Functions are objects that can be manipulated as such
- Definition

```
def function_name(arg1, arg2):
    …
    return …
```

- Call

```
function_name(x,y) or
function_name(arg1=x,arg2=y)
```

- Example

```
>>> def sum(a,b):
>>>     c=a+b
>>>     return c
>>> A=2; B=3; sum(A,B)
5
```

Tuples and lists

- Tuples and lists are iterable containers
  - access by index starting from 0,
  - elements can be of different types.
- lists are defined by `[]`,

```
>>> L = [10, 'toto', 20]
>>> L[1]
'toto'
```

- tuple are defined by `()` and are not modifiable

```
>>> T = (10, 'toto', 20)
>>> T[1] = 2
TypeError: 'tuple' object does not support item assignment
```

Tuples and lists

- Function `len()` returns the size

```
>>> L = [10, 'toto', 20]
>>> len(L)
3
```

- Function `min()` and `max()` return the min and max values

```
>>> print(min(L),max(L))
(10,'toto')
```

- `sorted(sequence)` return a sorted list of the sequence

```
>>> for i in sorted(L):
>>>     print(i)
10
20
'toto'
```

Tuples and lists

- `in`, `not in` test whether an element belongs to a sequence

```
>>> if 10 in L: …
```

- `==`, `<`, `>`, `<=`, `>=`, `!=` lexicographic comparison:

```
>>> (5,3,1) < (5,2,10)
False
```

- `count(value)` method returns the number of occurrences of `value` in the sequence
- `index(value)` method returns the index of the first occurrence of `value` in the sequence, raises an exception if `value` is not in the sequence

Modifying a list

- `L.append(x)` add element `x` at end of list `L`.
- `L1.extend(L2)` add list `L2` at the end of list `L1`.
- `L.insert(i, x)` insert `x` at position `i`.
- `del L[i]` remove value at position `i`.
- `L.pop(i)` remove and return value at position `i`
- `L.sort()` sort a list,
- `list(sequence)` convert a sequence into a list
- `tuple(sequence)` convert a sequence into a tuple.

Slices

- `L[i:j:k]` extract the sub-sequence starting at `i` ending at `j-1`, by steps of `k`.
  - if `k` not specified, `k=1`,
  - if `j` not specified, up to end of list,
  - if `i` not specified, `i=0`,
  - if `i` or `j < 0`, from end of list
  - if `j<i`, empty sequence,
  - if `i` or `j` out of range, replaced by beginning or end.

Slices

```
>>> L=[0,10,20,30,40,50,60,70,80,90]
>>> L[2:8]
[20, 30, 40, 50, 60, 70]
>>> L[2:8:2]
[20, 40, 60]
>>> L[8:]
[80, 90]
>>> L[::3]
[0, 30, 60, 90]
>>> L[-3:-1:]
[70, 80]
>>> L[-3:]
[70, 80, 90]
>>> L[::-1]
[90, 80, 70, 60, 50, 40, 30, 20, 10, 0]
```

Slices

- Slices and parallel affectation

  `a,b=L[2:4]` equivalent to `a=L[2];b=L[3]`

- Affectation with a sequence or parallel affectation of different size

```
>>> L[:3]=('a','b') #affectation from a tuple
>>> L[-2:]='y','z' #parallel affectation
>>> L
['a', 'b', 30, 40, 50, 60, 70, 'y', 'z']
```

  `0,10,20` replaced by `'a','b'`; `80,90` by `'y','z'`.

Lists and copy

- By default, lists are not copied except when slicing

```
>>> L=[0,10,20,30,40,50,60,70,80,90]
>>> L2=L
>>> L3=L[:]
>>> L2[3]='copy'
>>> L3[4]='copy'
>>> L
[0, 10, 20, 'copy', 40, 50, 60, 70, 80, 90]
```

- L2 is a reference to L while L3 is a copy of L.

Dictionary

- Mapping (key, value)
  - key can be any immutable object,
  - value can be any object.
  - `items()` method returns a *list* of tuples `(key,value)`,
  - `keys()` method returns the *list* of keys,
  - `values()` method returns the *list* of values,
  - `copy()` method returns a copy of the dictionary.

Dictionary

- Mapping (key, value)
  - key can be any immutable object
  - value can be any object

```
>>> D={}
>>> D['name']='Koch'
>>> D[(7,'avenue du Colonel Roche')]='LAAS'
>>> D[(14,'avenue Edouard Belin')]='CNRS-DR14'
>>> D
{(7, 'avenue du Colonel Roche'): 'LAAS', 'name': 'Lamiraux', (14, 'avenue Edouard Belin'):
'CNRS-DR14'}
>>> D[(14,'avenue Edouard Belin')]
'CNRS-DR14'
>>> for k,v in D.items():
>>>     print(k,v)
(7, 'avenue du Colonel Roche') LAAS
name Lamiraux
(14, 'avenue Edouard Belin') CNRS-DR14
```

Set

- Represent sets in the mathematical meaning
  - created by `set()`,
  - `add()` method adds an element,
  - `remove()` method removes an element,
  - operators `in`,`<`,`>`,`<=`,`>=`,`-`,`|`,`&`,`^`,
  - methods `isdisjoint`, `issubset`, `issuperset`, `union`, `intersection`, `difference`, `symmetric_difference`, `copy`

Set

```
>>> S1=set([1,2,3,4,5,6]) # or S1={1,2,3,4,5,6}
>>> S2=set([5,6,7,8])
>>> S1.union(S2)
set([1, 2, 3, 4, 5, 6, 7, 8])
>>> S1.intersection(S2)
set([5, 6])
>>> 5 in S1
True
>>> S2-S1
set([8, 7])
>>> S1^S2
set([1, 2, 3, 4, 7, 8])
```

Modules

- A module is a python file (or a shared object)
- `import mod` command import all objects defined in file `mod.py`.
- Objects defined in `mod.py` are accessible through namespace `mod`.
- Example

```
#file mod.py
a=0
>>> import mod
>>> mod.a
0
>>> import mod as m
>>> m.a
0
>>> from mod import * # generally try to avoid this, explicit is better!
>>> a
0
```

Class

- A class may have
  - a constructor,
  - instance methods and members,
  - class methods and members,
- Definition

```
class ClassName:
    # class declaration
```

- Construction of an instance

```
>>> a = ClassName()
```

Instance method

- function that applies to the object that calls it
- the first parameter is the object and is denoted by `self`

```
class ClassName:
    def method(self):
        …
>>> a = ClassName()
>>> a.method()
```

Constructor

- Constructor is a method denoted by `__init__` and called at instance creation
- It can have parameters and usually defines instance attributes.

```
class ClassName:

    def __init__(self, data):

        self.my_data = data

        ...
```

Class inheritance

- Allow to create a class that inherits another class methods and members,
- methods can be redefined
- constructor may call parent constructor

```
class Child(Parent):
    def __init__(self):
        Parent.__init__(self)
        …
```

- It is recommanded to make base class derive from `object`. Class is thus said “new style” class.

Special methods

- `__lt__`, `__le__`, `__eq__`, `__ne__`, `__gt__`, `__ge__(self,other)` overload operators `<`, `<=`, `==`, `!=`, `>`, `>=`
- `__str__(self)` define the conversion of the object as a string (print).

Exceptions

- Errors in python are dealt with using exceptions
- Some are defined by the language, but exception classes can be defined
- Example

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

Exceptions

- Exceptions derive from `BaseException` class and more frequently from `Exception` sub-class.
- It is possible to catch an exception in order to handle an error
- Exceptions not caught terminate execution.

Catching exceptions

- `try`/`except` catches exceptions
- Example

```
try:
    f = open('myFile', 'r')
except IOError as exc:
    print(exc)
```

- In this example, only `IOError` are caught.

Defining new exceptions

```
Class MyException(Exception):
    def __init__(self, msg):
        self.message = msg
    def __str__(self):
        return self.message
```

- Raising an exception

```
try:
    raise MyException('this is my error')
except MyException as exc:
    print(exc)
```

try, except, else, finally

- `else` clause is executed if no exception is caught,
- `finally` clause is executed whatever happens:

```
def divide(x,y):
    try:
        result = x/y
    except ZeroDivisionError:
        print('division by 0')
    else:
        print('result is %s'%result)
    finally:
        print('finally')
```

Exercise 1

- Define two lists: `l1` with names and `l2` with ages,
- Define a function taking two arguments: a name and an age and print

```
Mr name is age year(s) old.
```

- Use this function to print the above sentence for each name in `l1` with ages in `l2`.

Exercise 1: correction

```
def printAge(name, age):
    year = 'years' if age >= 2 else 'year'
    print('Mr %s is %i %s old'%(name, age, year))

l1 = ['Dupond', 'Durand', 'Dubois']
l2 = [1, 52, 36]

for index in range(len(l1)):
    printAge(name=l1[index], age=l2[index])
```

Exercise 2

- Compute prime numbers up to 1000:
  - define a function that returns the set of multiples not greater than 1000 of an integer,
  - From the set of integers, successively remove multiples of 2, 3, 5, 7...

Exercise 2: correction

```
def multiples(number, upper):
    return set(range(2*number,upper,number))

primeNumbers = set(range(2,1000))

for i in range(2,500):
    if i in primeNumbers:
        primeNumbers = primeNumbers - multiples(i, 1000)

primeNumbers = list(primeNumbers)
primeNumbers.sort()
print(primeNumbers)
```

Manipulating files

- `read (n)` reads n characters
- `write (s)` writes string s
- `readline ()` reads one line
- `close ()` closes the file
- Instruction `for` reads a file line by line:

```
f = open('myFile', 'r')
for line in f:
    print(line)
```

Instruction with

- `with` automatically calls methods `__enter__()` and `__exit__()` even if an exception is raised.
- Class file implements these methods. Thus, we can write

```
with open('myFile', 'r') as f:
    for line in f:
        print(line)
```

- No need to call `close` () anymore.