Introduction to Python

Walid and Jonathan

September 10, 2015

Outline

Expressions

```
1 + 2 * 3 /4
"ab" + "cd" + "ef"
```

Variables

```
a = 1 + 2 * 3 /4
b = "ab" + "cd" + "ef"

print(a)
print(b)

print(type(a))
print(type(b))
```

which results in:

```
2
abcdef
<type 'int'>
<type 'str'>
```

Lists

```python
li = [0, 1, 2, 3, 4]
print(li) # reference to the list

print(li[2]) # the second element in the normal order
print(li[-2]) # the second element in the reverse order
print(li[1:4]) # copy of the list like [2nd, ..., 4th[
print(li[-3:]) # copy of the list like [n-3, ..., n]
print(li[:-2]) # copy of the list like [1st, ..., n-2[

print(li + [5])
print(li[0:1] + li[2:5])
```

which results in:

```
[0, 1, 2, 3, 4]
2
3
[1, 2, 3]
[2, 3, 4]
[0, 1, 2]
[0, 1, 2, 3, 4, 5]
[0, 2, 3, 4]
```

Dictionnaries

- Dictionnary are data-structures that associates a value to a key.
- Access to a value is in O(1) (good for building indexes).

```python
di = {"key1": "value1", "key2": "value2"} # Declaring a dict
di_has_key1 = "key1" in di # Checking if "key1" is in the dict
value_key1 = di["key1"] # Accessing value of "key1"
print(("It is %s that \"key1\" is in di,"
       " whose value is %s") % (di_has_key1, value_key1))
di["key3"] = 42 # Setting "key3" to 42
```

which results in:

```
It is True that "key1" is in di, whose value is value1
```

If then else

- Keyword for declaring a control structure are if, elif and else.
- if and elif take a boolean expression as parameter, which can be a combination of and and or operators.

```python
# test the following with a switching between {1,2,3}
a = 1
if a == 0 or (a > 0 and a < 2):
    print("foo")
elif (a > 1 and a < 3):
    print("boo")
else:
    print("bar")
```

which results in:

```
foo
```

While

- Keyword for declaring a conditional loop is while.
- while take a boolean expression as parameter.

```
a = 1
print("initial value: %i" % (a))
while a < 10:
   a += 1
print("final value: %i" % (a))
```

which results in:

```
initial value: 1
final value: 10
```

For

- Keyword for declaring an iterative loop is for.
- for take a boolean expression as parameter.

```
for i in range(1, 5):
    print(i)
```

which results in:

```
1
2
3
4
```

For (dictionnaries)

```python
di = {"key1": "python", "key2": "cobra", "key3": "boa"}
for key in di:
    print(di[key])
```

which results in:

```
boa
cobra
python
```

Declaration of functions

Functions are declared with:

- the def keyword
- the name of the function
- the arguments of the function

```python
def foo(x):
  if x>0:
    return x+1
  else:
    return x-1
```

Evaluation of functions

The following code call the function foo

```
a = foo(10)
print(a)
```

which results in:

```
11
```

lambdas

Lambdas are mechnaisms from functional programming, which enables to do higher-order functions.

```
li = [0, 1, 2, 3, 4]
multiplie_par_2 = lambda x: 2*x
li2 = map(multiplie_par_2, li)

filter_higher_than_4 = lambda x: x >= 4
li3 = filter(filter_higher_than_4, li2)

print(li2)
print(li3)
```

which results in:

```
[0, 2, 4, 6, 8]
[4, 6, 8]
```

decorators

- A function f can be wrapped by a function g: g is a decorator of f.
- The decorator is applied one time, no matter how many times the decorated function is called.

```
def multiplie_par_2_decorator(func):
   def wrapper(x):
      return func(x) * 2
   return wrapper

@multiplie_par_2_decorator
def foo(x):
   return x

print(foo(1))
```

which results in:

```
2
```

classes

- The keyword class is used to define a new class. Is recommended to inherit from object.

```python
# Defining a new Class
class Person(object):
   """ Simple Person class """
   counter = 0  # Class attribute

   def __init__(self, name, age=0):
      """ This method is called when this class is instantiated """
      self.name = name  # Instance attribute
      self.age  = age   # An other instance attribute

   def say_hello(self):
      return "Hello I am : " + self.name
```

Example

```
# Object instantiation
x = Person("toto")

# Attributes R/W
x.name = "toto2"
x.age = 20

print(x.name)
print(x.say_hello())
```

which results in:

```
toto2
Hello I am : toto2
```

Advanced classes

```python
class Student(Person):
 """ Student class that herite from Person """

 def __init__(self, name, sexe, school=None):
  # Call to super constructor
  super(Student, self).__init__(name)

  self.sexe = sexe
  self.school = school

 def __str__(self):
  """ return a string when str() is called on instance of this class
   (equivalent to toString method in java)
  """
  return ("Name : %s, Age : %s, Sexe : %s, School : %s "
            % (self.name, self.age, self.sexe, self.school))
```

Example

```python
toto = Student("toto", "M")
titi = Student("titi", "F", school="EMN")

print(toto)
print(titi)
```

which results in:

```
Name : toto, Age : 0, Sexe : M, School : None
Name : titi, Age : 0, Sexe : F, School : EMN
```

Meta Programming (1/2)

```
bob = Person("bob")
alice = Person("alice")

print("Person has a counter (%s) and a name (%s)"
      % (hasattr(Person, "counter"), hasattr(Person, "name")))
print("Bob has a counter (%s) and a name (%s)"
      % (hasattr(bob, "counter"), hasattr(bob, "name")))

getattr(bob, "age") # or: bob.age
setattr(bob, "foo", "M") # or : bob.foo = "M"
```

The result is :

```
Person has a counter (True) and a name (False)
Bob has a counter (True) and a name (True)
```

Meta Programming (2/2)

```python
bob = Person("bob")
alice = Person("alice")

# Getting class information
print("bob’s class: %s " % (bob.__class__))

# Overwriting methods at runtime
print(bob.say_hello())
Person.say_hello = lambda self: "hacked say_hello(...)"
print(bob.say_hello())
```

The result is :

```
bob’s class: <class ’__main__.Person’>
Hello I am : bob
hacked say_hello(...)
```

Try/except

- Use try, except and finally to handle codes that can fail.

```
try:
   1 / 0
except Exception as e:
   print(e)
```

The result is :

```
integer division or modulo by zero
```

Try/except (advanced)

- The traceback enables to have more details about the source of the exception.

```python
import sys, traceback

try:
  1 / 0
except Exception as e:
  traceback.print_exc(file=sys.stdout)
```

The result is :

```
Traceback (most recent call last):
  File "<stdin>", line 4, in <module>
ZeroDivisionError: integer division or modulo by zero
```

Threads (1/2)

- Threads can be used thank to the Thread class provided by the threading module.

```python
from threading import Thread
import time

class Counter(Thread):

    def __init__(self, num):
        Thread.__init__(self)
        self.num = num

    def run(self):
        i = 0
        while i < 5:
            print("thread-%i: %i" % (self.num, i))
            i += 1
            time.sleep(0.1)
```

Threads (2/2)

```
counter1 = Counter(1)  # Declare two counters
counter2 = Counter(2)

counter1.start() # Start the counting threads
counter2.start()

counter1.join() # Wait untill the thread has finished
counter2.join()
```

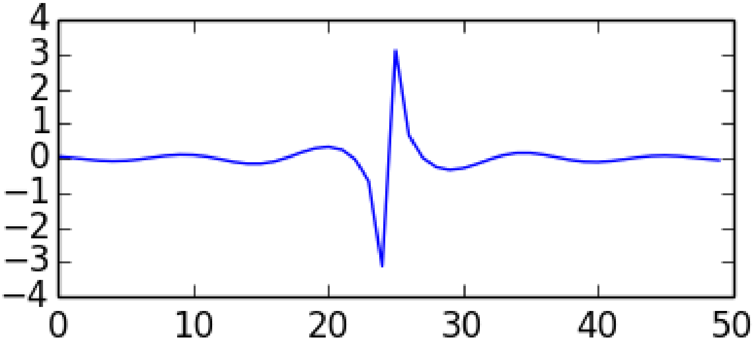
which results in:

```
thread-1: 0
thread-2: 0
thread-2: 1
thread-1: 1
thread-2: 2
thread-1: 2
thread-2: 3
thread-1: 3
thread-1: 4
thread-2: 4
```

Plotting with matplotlib

```python
import matplotlib, numpy
matplotlib.use('Agg')
import matplotlib.pyplot as plt
fig=plt.figure(figsize=(4,2))
x=numpy.linspace(-15,15)
plt.plot(numpy.sin(x)/x)
fig.tight_layout()
plt.savefig('python-matplot-fig.png')
return 'python-matplot-fig.png' # return filename to org-mode
```

Result


4
3
2
1
0
−1
−2
−3
−4
0
10
20
30
40
50