Python

Syntax and basic concepts: a comparison with Matlab

EMBS UP | 08.11.2016

Introduction

Introduction to Python

More general than Matlab – more capabilities

Less oriented towards matrix computation

Offers various packages for specific fields:

- numpy (linear algebra)
- matplotlib (graphic display)
- skimage (image processing)
- sklearn (machine learning)

```
x = ['Hello','World']

for word in x:
   print (word),
```

```
Hello World
```

Introduction to Python

Important:

- Indentation: fundamental in python;
- One command per line of code;

(Semicolons and brackets are seldom used, only in data structures like dictionaries)

```
x = ['Hello','World']

for word in x:
   print (word),
```

```
Hello World
```

Variable types

Variable types

```python
# Strings:
a = "Hello"
b = 'World'

# Numbers (integers, floats, etc.):
c = 7.3

# Lists (can have many types inside):
d = [1,2,4,8,16]

# Booleans:
e = True
f = False
```

```matlab
% Strings:
a = 'Hello';
b = 'World';

% Numbers:
c = 7.3;

% Arrays (only one type inside):
d = [1,2,4,8,16];

% Booleans:
e = true;
f = false;
```

Unlike C++ or Java (and just as Matlab), Python detects the type of variable automatically (it does not have to be specified), and each variable can change type along the code.

Variable types

```python
# Tuples (read-only lists):
g = (1,2,4,8,16)

# Sets (list that can have duplicates)
h = [1,1,4,8,8]

# Dictionaries:
i = {'name':'Sophie', 'age':34}
```

```matlab
% Structs:
h = struct('name','Sophie','age',34);
```

Attention: In lists, tuples, and sets, unlike Matlab, the first index is not 1 but 0.

Using Numbers

```python
# Any to integer:
int(x, base) # or int(x)

# Any to long int:
long(x, base) # or long(x)

# Any to float:
float(x)

# Create complex number:
complex(real, imag)
```

```
% Many options:
num2int(x);
str2num(x);
str2double(x);
% (…)

complex(real, imag);
```

Using Strings

```python
# Convert to string:
str(x)

# Evaluate expression:
y = eval("x + 2")

# 1st to 4th characters of a string:
x[0:4]

# Concatenate strings:
x + y
x += y
```

```
% Various options like:
num2str(x);


eval('y = x + 2');


x(1:4);


strcat(x, y);
```

Using Lists and Tuples

```python
# Convert to list:
list(s)

# Convert to tuple:
tuple(s)

# Convert to set (can have duplicates)
set(s)

# Modify entry
l[i] = x

# Append to list:
l.append(x)
```

```
% Also various options…

% Not applicable

% Not applicable


li(i) = x;


li(end + 1) = x;
```

Using Dictionaries

```
dict = {'ab':1,'bc':'cd','de':21}

# Accessing a key's value:
print dict['bc']

# Creating a new key-value entry:
dict['ef'] = 'abc'
```

cd

```
dict = struct('ab', 1, 'bc', 'cd',
'de', 21);


disp(dict.bc);


dict.ef = 'abc';
```

cd

In the case of Matlab, they are called structs.

Other useful things

Print

```
x = [1, 33, 'dff', 3.98]
y = {'name':'John','age':22}

print x
print y
```

```
[1, 33, 'dff', 3.98]
{'age': 22, 'name': 'John'}
```

```
x = [1, 33, 'dff', 3.98];
y = struct('name','John','age',22);

disp(x);
disp(y);
```

```
!dff
  name: 'John'
   age: 22
```

`print` alone can display almost any type of variable on the screen.

Time

```
from datetime import datetime

now = datetime.now()

print '%s/%s/%s' % (now.month,
now.day, now.year)

print '%s:%s:%s' % (now.hour,
now.minute, now.second)
```

```
11/8/2016
15:55:12
```

```
t = datetime('now');

disp(datestr(t,'mm/dd/yyyy'));


disp(datestr(t,'HH:MM:SS'));
```

```
11/08/2016
15:55:12
```

Important: notice how to import external libraries in python (or code in other files).

Inputs

```
name = input('What's your name?')

print ("Hello " + name + "!")
```

```
What's your name? "Helen"
Hello Helen!
```

```
name = input('What's your name?');

disp(["Hello ", name, "!"]);
```

```
What's your name? 'Helen'
Hello Helen!
```

Writing files

```python
# Creating a list:
my_list = [i**2 for i in range(1,6)]

# Opening file:
my_file = open("output.txt", "w")

# Writing to file:
for item in my_list:
    my_file.write(str(item)+"\n")

# Closing file:
my_file.close()
```

```matlab
my_list = [1,4,9,16,25];


fileID = fopen('output.txt','w');


for i = 1:5
   fprintf(fileID, '%i', my_list(i));
end

fclose(fileID);
```

The second parameter of `open` can be `"w"` – write, `"r"` – read, or `"r+"` – read and write.

Reading files

```
my_file = open("output.txt", "r")

print my_file.read()

my_file.close()
```

```
1
4
9
16
25
```

```
fileID = fopen('output.txt','r');

a = fscanf(fileID, '%i');

fclose(fileID);

disp(a);
```

```
[1,4,9,16,25]
```

Operations

Arithmetic operations

```python
# Addition:
x + b

# Subtraction:
x - b

# Multiplication:
x * b

# Division:
x / b
```

```
x + b;

x - b;

x * b;

x / b;
```

Arithmetic operations

```python
# Exponent:
x ** b

# Floor:
x // b

# Modulus:
x % b
```

```
x ^ b;

x % b;
```

Comparison operations

```
# Equal:
x == b

# Not equal:
x != b

# Greater or lesser:
x <> b

# Greater:
x > b
```

```
x == b;

x ~= b;

x < b || x > b;

x > b;
```

Comparison operations

```python
# Lesser:
x < b

# Greater or equal:
x >= b

# Lesser or equal:
x <= b
```

```
x < b;

x >= b;

x <= b;
```

Assignment operations

```python
# Addition:
x += b

# Subtraction:
x -= b

# Multiplication:
x *= b

# Division:
x /= b
```

```
x = x + b;

x = x - b;

x = x * b;

x = x / b;
```

Assignment operations

```python
# Exponent:
x **= b

# Floor:
x //= b

# Modulus:
x %= b
```

```
x = x ^ b;

x = x % b;
```

Logical operations

```python
# And:
x and b
x & b

# Or:
x or b
x | b

# In:
x in b

# Not in:
x not in b
```

```
x & b;


x || b;


find(b == x);


~find(b == x);
```

Conditionals and Loops

If statements

```python
list = [2, 4, 6]

if (list[0] == 2):
   if 8 in list:
      print list
   else if 6 in list:
      list.append(8)
   else:
      pass
```

```
list = [2, 4, 6];

if (list(1) == 2)
   if find(list == 8)
      disp(list);
    elseif find(list == 6)
      list(end+1) = 8;
    end
end
```

`pass` is used when a statement is needed, but nothing should be performed by it.

For loops

```python
list = [2, 4, 6]

for item in list:
   if item ** 2 > 12:
      print item

# Or, alternatively:
for n in range(3):
   if list[n] ** 2 > 12:
      print list[n]
```

```
4
6
```

```matlab
list = [2, 4, 6];

% Not available in Matlab



for n = 1:3
   if list(n)^2 > 12
      disp(list(n));
   end
end
```

```
4
6
```

`range(a, b, c)` returns a list that includes all numbers from a to b-1, with incremente of c between them. The second method uses reference instead of copy, and thus allows for direct editiing of the variable.

While loops

```
list = [2, 4, 6, 8]
high = False
n = 0

while high is False:
   if list[n] > 5:
      high = True
      break
   else:
      print list[n]
   n += 1
```

```
2
4
```

```
list = [2, 4, 6, 8];
high = false;
n = 1;

while high == false
   if list(n) > 5
      high = true;
      break
   else
      disp(list(n));
   end
   n = n + 1;
end
```

```
2
4
```

Functions

A simple function

```python
# Adds 23% tax to a bill.
def tax(bill):
    wTax = bill * 1.23
    print "Bill: %f,with tax: %f" %
(bill,wTax)
    return wTax
```

```
function [wTax] = tax(bill)
    wTax = bill * 1.23;
    disp(["Bill: ", num2str(bill),
",with tax: ", num2str(wTax)]);
end
```

Also important: the way we can put variables inside formatted strings.

Calling the function

```python
# Adds 23% tax to a bill.
def tax(bill):
    wTax = bill * 1.23
    print "Bill: %f,with tax: %f" %
(bill,wTax)
    return wTax

# Calling the function:
bill = 35.60
tax(bill)
```

```
Bill: 35.600000,with tax: 43.788000
```

```octave
function [wTax] = tax(bill)
    wTax = bill * 1.23;
    disp(["Bill: ", num2str(bill),
",with tax: ", num2str(wTax)]);
end


bill = 35.60;
tax(bill);
```

```
Bill: 35.6,with tax: 43.788
```

Classes

Classes

```python
class Slave(object):
   def __init__(self):
      self.salary = 0
   def __init__(self, val):
      self.salary = val
   def displaySalary(self):
      print self.salary
```

```
classdef Slave
   properties
      salary = 0;
   end
   methods
      function obj = Slave(val)
         if nargin > 0
            obj.salary = val;
         end
      end
      function displaySalary()
         disp(obj.salary);
      end
   end
end
```

The “top classes” always inherit from `object`.
Classes in Matlab are almost never used, so the comparison stops here.

Inheritance

```
# (...)

class GoodSlave(Slave):
   def __init__(self, bonus):
      self.salary = 0
      self.bonus = bonus
   def __init__(self, val, bonus):
      self.salary = val
      self.bonus = bonus
   def displaySalary(self):
      print (self.salary + self.bonus)
```

Children classes include the name of the parent, in parenthesis, in its header.

Using classes

```
# (…)

jack = Slave(25)
philip = GoodSlave(25,45)

print jack.salary

philip.displaySalary()
```

```
25
70
```

Example code, explanations, and much more:
https://www.tutorialspoint.com/python/
https://docs.python.org/3/tutorial/

Online interactive courses on python basics:
https://www.codecademy.com/learn/python
http://www.learnpython.org/

João Ribeiro Pinto | EMBS UP | 08.11.2016