Tips&Tricks

29th January 2015
BARCELONA

About Us

meetup: http://www.meetup.com/PyLadies-BCN/

twitter: @PyLadiesBCN

e-mail list: pyladies-bcn@googlegroups.com

Why a Tips&Tricks session?

- Beginners friendly session
- Show typical Python “beginner” mistakes
- Practicing
- New format meetups

- **Afterbeers**

Structure

- Some theory explanations (short)
- Examples
- Exercices
- More practice

ARE YOU READY??

For … in statement

In Python basically exist two types of control flow statements: **while** and **for**

At first sight for seems inoffensive but is wide used by Python programmers.

Python’s for statement iterates over the items of any sequence (a list or a string), in the order that they appear in the sequence.

Example:

```
words = ['cat', 'window', 'window’']
for w in words:
    print w, len(w)
```

For … in statement

But, what happens when we want to create a loop but we don't have any element to iterate over it?

We have to create an iterable element (for example, using range or variants):

Example:

```
for i in range(0, 100, 2):  # odd numbers until 100 (not included)
    print i
```

For … in statement

Try this tricks and learn to no to reinvent the wheel:

```
for i in reversed(range(1, 10, 2)):
    print i
```

```
basket = [‘apple’, ’orange’, ’apple’, ’pear’, ’orange’, ’banana’]
for f in sorted(set(basket)):
    print f
```

For … in statement

Have you ever written something like this?

```
x = [1, 5, 7, 3, 8]

i = 0
for e in x:
    print 'x[%d]=%d' %(i,e)
    i+=1
```

In idiomatic Python:

```
for i,e in enumerate(x):
    print 'x[%d]=%d' %(i,e)
```

For … in statement

Take a look at **itertools** module!!

“Functions creating iterators for efficient looping”

https://docs.python.org/2/library/itertools.html

Unpacking variables

In other languages assigning to a variable is like putting the value in a box.

To specify that the box ‘a’ contains an integer we have to write: int a = 1

Then, when we assign one variable to another we make a copy of the value and we put it in the new box: int a = 2 int b = a

Unpacking variables

In Python variables are identifiers attached to objects using “tags”: a = 1

If we assign one name to another, we're just attaching another name tag to an existing object: b = a

Sounds interesting, but why is this important?

In python we don’t need auxiliary variables anymore!!

Unpacking variables

Example:

```
a = 2
b = 5
a, b = b, a
```

What is the value of a? And the value of b?

Unpacking variables

Exercise

We know that a DinA0 paper is 841x1189 mm. Knowing the relation between different DinA sizes (take a look at the picture) can you calculate DinA1 to Dina10 sizes?

Unpacking variables

Solution:

```
H, L = 841, 1189
for i in range(1, 11):  # from 1 to 11 (11 not included)
    H, L = L/2, H
    print 'DINA-%d: %d x %d' %(i,H,L)
```

That’s not magic! How it really works?

Unpacking variables

Python automatically generates tuples when it finds comma separated values, even if they are not delimited by parentheses (coma is tuples constructor, not parentheses).

Packing allows us to simulate multiple variables return on functions or swapping the values of two variables without creating a third one:

```
var1, var2 = x, y
```

Here python creates a tuple (x,y), **packing**, and assigns the first element to var1 and the second to var2, **unpacking**.

Simple, isn’t it?

Unpacking variables

Example:

```
def my_fuction():
   x = 10
   y = x*2
   return x, y  # packing: python creates the tuple (x, y)

var1, var2 = my_function()
# unpacking: var1 gets the value of x, and var2 the value of y
```

Unpacking lists

```
def func1(x, y, z):
    print x
    print y
    print z

def func2(*args):
    args = list(args) # Convert args tuple to a list so we can modify it
    args[2] = 'awesome!!!'
    func1(*args)

func2('This', 'is', 'boring')
```

Unpacking lists

Exercise

Write a function that given a variable of numbers returns their sum.

Sorry!!

It’s a quite stupid exercise because method sum do exactly the same!!

Then try:

sum(1,2,3,5)

Unpacking lists

Solution:

```
def sumFunction(*args):
    result = 0
    for x in args:
        result += x
    return result
```

What it does return ? sumFunction(1,1,8,9) and sumFuntion(3,7,9,22,9,6)

Unpacking dictionaries

What it happens when you enter this code?

Example:

```
def func(required_arg, *args, **kwargs):
        print required_arg

        if args:  # if args is not empty.
            print args

        if kwargs:  # if kwargs is not empty.
            print kwargs
```

Unpacking dictionaries

Test your previous function:

```
func('Arguments')
func('Arguments', (1, 8, 'bc'), name='Marta', age=28)
func('Arguments', (1,8, 'bc', 6.89), name='Marta', age=28, telephone = 905330173)
```

Try it again using this new function definition:

```
def func2(required_arg, args, kwargs)
```

What does this function return?

List Comprehensions

List comprehensions are syntax shortcuts for this general pattern:

Example:

```
my_list = [1,3,9,7]
new_list = []
for x in my_list:
    new_list.append(x * 2)
print new_list

print [x * 2 for x in my_list]
```

or

```
print [x * 2 for x in [1,3,9,7]]
```

What does this function do?

List Comprehensions

More examples (not only mathematics):

```
print [str(x) for x in range(20)]
```

```
text = "My hovercraft is full of eels."
first_chars = [word[0] for word in text.split()]
```

What does this code do?

```
string = "Hello 12345 World"
print [x if x.isdigit() else '-' for x in string]
```

List Comprehensions

```
result = [ x ** y for x in [10, 20, 30] for y in [2, 3, 4]]
```

```
for x in [10, 20, 30]:
    for y in [2, 3, 4]:
        result.append(x ** y)
```

We can have multiple for-loops and if-conditions if the conditions are complex regular *for* loops should be used.

Always choose the more readable way.

Lambda function

Syntax to define one-line mini-functions ‘on the fly’

Remember: lambda functions are only a matter of style.

Without lambda function:

```
def f(x):
    return x*2
```

With lambda function:

```
g = lambda x: x*2
g(3)
```

Without assigning it to a variable:

```
(lambda x: x*2)(3)
```

Lambda function

Examples:

```
sentence = 'It is raining cats and dogs'
words = sentence.split()
print words

lengths = map(lambda word: len(word), words)
print lengths
```

Notice that **map** is a built-in function that applies an operation to each item of the list and collects the result.

Lambda function

Same example in a single line?

```
print map(lambda w: len(w), 'It is raining cats and dogs'.split())
```

THANKS!!

Keep practicing and having fun!

Did you enjoy it?