Obroni Computer Club – Functions in Python

Joachim Breitner

SOS Hermann Gmeiner International College

October 3th, 2006

Today's topics

1. Prelude
   - Our next steps
2. Functions in Python
   - Defining functions
   - Function parameters
   - Return values
3. Dictionaries in Python
   - Creating dictionaries

Prelude

What can we expect now?

We finally can write interesting programs. Where do we want to go from here?

What can we expect now?

We finally can write interesting programs. Where do we want to go from here?

Maybe already next school week, we have a look at how to make our python programs network aware. This means, that you can run your little text-based games as servers, and everyone can connect to them, even from windows.

A bit later, if you want, we can see how we use our programs to serve to web browsers, creating programs that everyon can use from any computer, and that are not text based any more.

Today, we'll have a look at a construct that makes complex programs easier to write and maintain: Functions.

Functions in Python

Over and over again...

Often, a certain piece of code has to be used in more than one place in your program, or you just want to order your code in a more concise manner. For that, you might want to use functions.

def-statement

```
1 def my_function():
2     code
```

After you have defined a function like this, you can use it anywhere in your program, by writing my_function()
The name of the function (here "my_function") is a python identifier, so the same rules apply as for variables. You can't have a variable and a function of the same name.

Example: Noisy hostels

Assuming, we have a maze-like game with hostels. Some hostels are very noisy, so when you enter them, we print some noise.

Example: Noisy hostels

Assuming, we have a maze-like game with hostels. Some hostels are very noisy, so when you enter them, we print some noise.
Consider the following function:

```
1 def noise():
2     print "Swooooooooooooosh"
3     print "Bang!"
4     print "Crash."
5     print "Beeeeeeeeeeeeeeeeeeeeeeeeeep"
```

Example: Noisy hostels (ctnd.)

And this code fragment:

```
1 | if    current_hostel == "Nile":
2 |       noise()
3 | elif  current_hostel == "Niger":
4 |       noise()
5 | elif  current_hostel == "Volta":
6 |       noise()
7 |       noise()
```

See how that saves you a lot of work? Also, if you want to add more noises, or change some, you only have to do it at one place. The code is also easier to read.

Some variations...

Functions would be boring if they would be doing the same thing every time. Luckily, we can pass information to the function, using parameters.

Assume we want to have a function that prints everything twice:

```
1 | def print2(text): # text is a parameter
2 |     print text
3 |     print text
```

Now we can use this function with all kind of texts:

```
1 | print2("Hello!")
2 | name = raw_input("What is your name? ")
3 | print2("Nice to meet you, "+name)
```

Some more variations...

You can have more than one parameter, just separate them with commas:

```
1  def my_print(important, text):
2      if important:
3          print "IMORTANT! "+text+" IMPORTANT!"
4      else:
5          print text
6
7  # later in the code, e.g. a nuclear plant controller
8  my_print(heat > 1000, "Core: "+str(heat)+" degrees")
9  # or a email client
10 my_print(sender == 'Mom', 'You got a new e-mail!')
```

Exercise: Custom print function

Write a nice print function, for example one that puts something like '`-->`' before each line, or puts lines like '`_____ ... ____`' before and after it. Be creative!

Take one of your old programs (e.g. the hello world program, or the Fahrenheit conversion program), and make it use your new function.

Getting something back for what you give

You might have noticed that I have called `int()`, `str()` and `raw_input()` functions before. Because that is what they are. What makes them different is that you can use them **as values**. In other words: these functions return someting.

The **return**-statement

```
1 def some_function(param1, param2):
2     some code
3     return value
```

Where value can be anything (a string, a number, a calculation, a variable...). The **return** statement does not have to be at the end, it can be anywhere, and when the program comes to such a statement, the function ends and the code goes on where the function is called.

Example: string repeater

Here we have a function that takes a number and a string and returns the string repeated that often.

```
1 def repeater(number, string):
2     result = '' # an empty string
3     while (number > 0):
4         result = result + string
5         number = number - 1
6     return result
```

The statement

```
1 print repeater(3, 'hi')
```

will now output “hihihi”.

Example: better string repeater

The code has a problem: Negative numbers don't work well. Also, there is a nicer way to write a = a + b.

```
1 def repeater(number, string):
2     if (number < 0):
3         raise 'Negative input to repeater!'
4     result = '' # an empty string
5     while (number > 0):
6         result += string
7         number -= 1
8     return result
```

raise causes a program error message with the given string, and can be used to abort a program **in case of an programming error**.

Exercise: Fahrenheit again...

We want to write a second version of the Fahrenheit program that uses functions. Here is the main part of the program, and you should now write the functions to complete this:

```
1 | #!/usr/bin/python
2 | what = ask_what()
3 | inp  = ask_degree()
4 | if user_wants_fahrenheit (what):
5 |     out = f2c(inp)
6 | else:
7 |     out = c2f(inp)
8 | give_answer(out)
```

Hint: There are 6 functions to write, some with parameters, some without, some with return values, some without. If you need help with the program itself, you can find the old version in `K:\Others\OCC\OCC-5.pdf`.

Dictionaries in Python

lists considered unfortunate

You might remember how we had a list of lists for the connected rooms in our maze last time.

```
1  rooms = ['a', 'b', 'c', 'd']
2  connected = [
3      ['b', 'd'],
4      ['c', 'a'],
5      ['d', 'b'],
6      ['a', 'c'],
7  ]
8  # later
9      room_number = rooms.index(position)
10     ways = connected[room_number]
```

This is not nice. We have to "manually" match the rooms to the entries in the connected list, and the code to get the list is ugly.

Better lists

The problem is that lists always are accessed using numbers. **Dictionaries**, though, are a bit like lists, but you can use strings to address the values. Then, the code looks like:

```
1 rooms = ['a', 'b', 'c','d']
2 connected = {
3       'a': ['b','d'],
4       'b': ['c','a'],
5       'c': ['d','b'],
6       'd': ['a','c'],
7 }
8 # later
9       ways = connected[position]
```

Much nicer, isn't it? Now the order in "connected" does not matter (we are using names), and we can get the right list immediately.

More fun with dictionaries


```
1  # Createing a dictionary
2  rating = { 'Hip Hop': 1, 'Rock' : 9}
3  # Reading a value
4  print 'I rate Hip Hop at: ' + str(rating['Hip Hop'])
5  # Writing a value
6  rating['Rock'] = 10
7  # Adding a value
8  rating['Classic'] = 3
9  # Checking if we have a rating
10 if 'RNB' in rating:
11     print 'We have rated RNB'
```

Removing a key?

Can you also remove a key? Sure you can! Find out yourself, using the

Python Library Reference

`http://python.org/doc/2.3/lib/` Bookmark this!

There you can find all kinds of things you can do with python, to explore yourself, including file access, networking, colors on the terminal etc., for your own experiments.

Removing a key?

Can you also remove a key? Sure you can! Find out yourself, using the

Python Library Reference

`http://python.org/doc/2.3/lib/` Bookmark this!

There you can find all kinds of things you can do with python, to explore yourself, including file access, networking, colors on the terminal etc., for your own experiments.

You can delete a key using "**del** rating['Hip Hop']", as described on `http://python.org/doc/2.3/lib/typesmapping.html`

Any Questions?

If we have time left, feel free to “program around a little bit”, so soon you can show off your work that all the others who have no idea how how to do such things.

Please do remove the OCC posters from the notice boards, thanks!