Functions in Python

L435/L555

Dept. of Linguistics, Indiana University
Fall 2016
What is a function?

Definition

A function is something you can call (possibly with some parameters, i.e., the things in parentheses), which performs an action and returns a value.

Example

def hello(name, greeting):
    return greeting + "", + name
print(hello( 'Markus', 'privet'))

Define first, then call!

In python, a function must be defined before you can call it. e.g., define it on line 10, call it on line 15.

(http://www.greenteapress.com/thinkpython/html/thinkpython004.html)
Function calling

We’ve seen functions many times before:
- Built-in functions: \(\text{int}()\), \(\text{type}()\), ...
- Module-based functions: \(\text{math.log}()\), \(\text{random.choice}()\), ...

We have also seen **function composition**
- \(\text{int}(	ext{input("Enter a number:"))}\)
- The output of the inner function (e.g., \(\text{input}()\)) is passed as the input to the outer function (e.g., \(\text{int}()\))
Why use functions?

Functions are extremely useful because:

- They make code reusable
- They make a program more structured, making the logic clearer
- They make a program more readable, especially when it gets longer
- They make it is easier to work with several programmers
Functions calling functions

```python
def print_lyrics():
    print("Stonehenge! ’Tis a magic place")
    print("Where the moon doth rise with a dragon’s face")

def repeat_lyrics():
    print_lyrics()
    print_lyrics()

repeat_lyrics()
```
Parameters (arguments)

Definition
Parameters (also known as arguments) are inputs to functions.

Example
When you use the `min()` function, you pass the function a list as a parameter

▶ e.g., `min([8,6,7])` returns 6
Local scope

Variables and parameters in functions have local scope.

```python
def change_name(name):
    name = 'The Thamesmen'

name = 'The New Originals'
change_name(name)
print(name)  # the value of this 'name' is unchanged

def again():
    mypi = 3.11
print(mypi)  # this gives an error: 'mypi' is undefined
```
Mutable types

Mutable data structures change in functions.

```python
def change(lst):
    lst.extend(["and","they","do","live","well"])

mylist = ["where","the","banshees","live"]
change(mylist)
print(mylist)
```
Three types of parameters

**positional**  Positional parameters must be entered in the correct order
   hello (name, greeting)

**keyword**  Keyword parameters can be entered in any order
   hello (greeting=’Ni Hao’, name=’Marty’)

**collected**  Parameters can also be collected by a function, allowing the user to input any number of parameters to the function

```python
def hello2 (*collectedParams):
    print ("Intermediate value:", collectedParams)
    return ', '.join ([str(x) for x in collectedParams])
print (hello2 ( 'tonight ', 'x ', 4))
```
Parameter types

Definition
Any kind of variable can be passed to a function (string, integer, float, list, dict, tuple, object). Your function must use these as the right type though.

Example

```python
def sortPeople(people):
    return sorted(people)

spinalTarp= 'Nigel, David, and Derek'
print(sortPeople(spinalTarp))
spinalTap=[ 'Nigel', 'David', 'Derek']
print(sortPeople(spinalTap))
```
Comment your code!

You must know/remember which types work for a function, so it makes sense to add comments that specify the types of the parameters and of the return value.

Example

```python
# function sortPeople sorts the input & returns it
# input: people – list or string
# output: list
#         (list of characters if input=string)
def sortPeople(people):
    return sorted(people)
```
Default values

Parameters can be assigned a default value, used only if no value is passed in

```python
def myadd(D, key, value=1):
    if key in D:
        D[key] += value
    else:
        D[key] = value
```
Return values

Definition
Parameters are inputs to functions. Return values are outputs.

Multiple return values
To return more than one value, put them in a tuple

```python
def rhymes():
x = 'cakes'
y = 'aches'
return (x, y)
```

```python
foo = rhymes()
one, two = rhymes()
```
Tip on printing

Avoid the following

Printing out stuff in functions (unless debugging)

```python
def hello():
    print("hello, world")
```

Instead, do the following

Returning stuff in functions and printing later

```python
def hello():
    return "hello, world"
print(hello())
```
Recursion

A function calls (other) functions

- **Recursion** is when a function calls itself

```python
def countdown(n):
    if n <= 0:
        print(’Blastoff!’)
    else:
        print(n)
        countdown(n-1)

countdown(5)
```
Fibonacci numbers

Iterative version

```python
def fib_iter(n):
    f_minus2 = 0
    f_minus1 = 1
    if n == 1:
        f_i = 0
    elif n == 2:
        f_i = 1
    else:
        for i in range(2, n):
            f_i = f_minus2 + f_minus1
            f_minus2 = f_minus1
            f_minus1 = f_i
    return f_i
```
Fibonacci numbers

Iterative version

```python
def fib_recur(n):
    if n == 1:
        return 0
    elif n == 2:
        return 1
    else:
        return fib_recur(n-1) + fib_recur(n-2)
```
Recursion notes

A few points regarding recursion:

1. Whatever parameters are passed must move towards some completion, e.g., integers get smaller \((n-1)\)

2. Recursive functions have two parts:
   
   2.1 **Base case(s):** what to do when you reach the “bottom” (e.g., if \( n == 1 \))
   
   2.2 **Recursive case:** what to do in moving from one value to another