Functions

Practice 03.02

By *Marina Barsky*

Functions have to be called in order to work!

```
def greeting (name):
    return 'Happy birthday, dear ' + name
```

- When you define a function above – a special place in memory stores all information about this function under a special variable called *greeting*. At this point your function is just a piece of code (text for Python)

- You need to call the function in order to make it do its work:

```
greeting ('Marina')
```

What do you expect to happen?

Call to function *greeting*

Custom argument to be used as function parameter

There are two types of functions

Math type of functions

```
def double_number (n):
    return 2*n
```

- Functions that report to the caller – **return** value
- This value can be printed or stored in a variable

Imperative type of functions

- Functions that do something but never report back – **do not have return** statement
- If there is no return statement, then we say that the function returns a special value: **None**

```
def print_greeting (name):
    print ('Happy birthday, dear', name)
```

Proper functions

- Functions are small independent pieces of code which can be combined to build a more complex program
- Functions should be treated as a Blackbox:
  - The only input to the function is through its parameters
  - The only output is through its return value
- Samurai principle: *return or die*

Proper function

```
def dbl(x):
    """ doubles x """
    return 2*x


>>> answer = dbl(21)
```

Not a function: sub-program

```
def dblPR(x):
    """ doubles x """
    print (2*x)


>>> answer = dblPR(21)
```

To function or not to function...

- Organize your code into “paragraphs” - capture a complete thought and “name it”
- Don’t repeat yourself - make it work once and then reuse it
- If something gets too long or complex, break it up into logical chunks and put those chunks into functions
- Make a library of common stuff that you do over and over - perhaps share this with your friends...

Function patterns

1. Numeric functions
2. Functions on strings
3. Functions on lists
4. Conditional functions
5. Functions that return Boolean values:
    - point of no return
    - no if required

1. Functions that work on numbers. Function reuse

English				metric		
1 **inch**				=	2.54	cm
1 **foot**	=	12	in.			
1 **yard**	=	3	ft.			
1 **rod**	=	5(1/2)	yd.			
1 **furlong**	=	40	rd.			
1 **mile**	=	8	fl.			

The United States uses the *English* system of (length) measurements. The rest of the world uses the *metric* system.

So, people who travel abroad and companies that trade with foreign partners often need to convert English measurements to metric ones and vice versa.

Develop the functions *inches->cm*, *feet->inches*, *yards->feet*, *rods->yards*, *furlongs->rods*, and *miles->furlongs*.
Then develop the functions *feet->cm*, *yards->cm*, *rods->inches*, and *miles->feet*.

Hint: Reuse functions as much as possible.

2. Functions with strings

- Write a function ***flipside*** *(astring)*, which takes in a string *astring* and returns a string whose first half is *astring*'s second half and whose second half is *astring*'s first half. If *len(astring)* (the length of astring) is odd, the first half of the input string should have one more character than the second half. (Accordingly, the second half of the output string will be one shorter than the first half in these cases.) Here you may want to use the built-in function *len(astring)*, which returns the length of the input string, astring.
- Examples:

```
>>> flipside('homework')
workhome
>>> flipside('carpets')
petscar
```

3. Functions with lists

- The local supermarket needs a program that can compute the value of a bag of coins.
- Define the program ***sum_coins***. It consumes a **list** of four numbers: the number of pennies, nickels, dimes, and quarters in the bag; it produces the amount of money (dollars) in the bag.

4. Conditional paths inside the function

- Some banks pay different levels of interest for saving accounts. The more a customer deposits, the more the bank pays. In such arrangements, the interest rate depends on the interval into which the savings amount falls.

n <= 1000 → .040
n <= 5000 → .045
n <= 10000 → .055
n > 10000 → .060

- To assist their bank clerks, banks use ***interest_rate*** functions.
- Our interest rate function must determine which of several conditions holds for the input. We say that the function is a ***conditional function***, and we formulate the definition of such functions using conditional expressions.

5. Functions that return Boolean values: True or False

- These functions can be written with a single line

```
def is_even (n):
  return n % 2 == 0

def can_vote (age):
  return age >= 18

def is_teenager (age):
  return age < 19
```