More About Functions

CS 8: Introduction to Computer Science, Winter 2019
Lecture #4

Ziad Matni, Ph.D.
Dept. of Computer Science, UCSB

A Word About Registration for CS8

- This class is **FULL**,

  & the waitlist is **CLOSED**.

Administrative

- Lab01 – due Friday *(make sure your submission is on there)*
- Hw02 – due next week on WEDNESDAY
  - Because there's no school next Monday…
- Linux workshop repeat this
  Friday @ 10 AM (Phelps 2510)

Lecture Outline

- Strings & Operations on Strings
- Intro to Lists & Tuple
- Intro to Functions

Yellow Band = Class Demonstration! ☺

Defining Your Own Function

- To define a function in Python, the syntax is:

```
def functionName (list of parameters):
    # a block of statements appear here
    # all of them must be indented (with tabs)
```

  - **`def`** – a mandatory keyword that **defines a function**
  - **`functionName`** – any legal Python identifier (e.g. myLittleFunction)
  - **`( ):`** – mandatory set of parentheses **and** colon
  - ***`list of parameters`*** – object names
    - Local references to objects (i.e. raw data or variables) that are passed into the function
  - e.g. `def myLittleFunction(pony1, pony2, 3.1415):`

Example Definition

```
# My first function! Yay!
def dbl(x):
    """This function returns double its input x"""
    print("Doubling the number to:", x)
    return 2*x    # I need to "return" the result
```

Let's try it out!

FUNCTION RULES!

```
# My first function! Yay!
def dbl(x):
    """This function returns double its input x"""
    print("Doubling the number to:", x)
    return 2*x    # I need to "return" the result
```

Function header
*x is the input **parameter** (also called **argument**)*

Function body

*Indentation: **VERY IMPORTANT***
Achieved with a tab character or just spaces
*All the lines in the function body are indented from the function header, and all to the **same** degree*

***docstring**: a comment that becomes part of Python's built-in help system!*
With each function be sure to include one that:
a) describes overall what the function does, and
b) explains what the inputs mean/are

More Example Definitions

```
# This function calculates the distance between (a,b) and (0,0)
def distance(a, b):
      x = a**2     # Note the tab indent!!!
      y = b**2     # Recall ** means "to the power of"
      z = (x + y) ** 0.5
      return z     # I need to "return" the result
```

!!! Alternatively !!!

```
def distance(a, b):
      return ( (a**2) + (b**2) ) ** 0.5
```

Let's try it out!

Flow of Execution of a Function

```
def dbl(x):
        """This function returns double its input x"""
        print("Doubling the number to:", x)
        return 2*x
```

When you call a function, you have to use its name and its parameter(s) *just like they were defined*

Example:

to call the dbl function on 21, you'd have *to call it* like this:

```
dbl(21)
```

Flow of Execution of a Function

```
def dbl(x):
        """This function returns double its input x"""
        print("Doubling the number to:", x)
        return 2*x
```

When you call a function, Python executes it starting at the first line in its body, and carries out each line in order

Though *some* instructions *can* cause the order to change

… more soon!

Local vs Global Variables

- A **global** variable is defined EVERYWHERE in the program

- A **local** variable is defined within some specific confines of a computer program
  - i.e. not everywhere in the program

Parameters are Specialized Variables

```
def dbl(x):
        """This function returns double its input x"""
        print("Doubling the number to:", x)
        return 2*x
```

When you call a function, the value you put in parenthesis gets put into a special part of computer memory that's labeled with the name of the parameter and is available for use within the function

Example:

in **`dbl(x)`**, the var. **x** can be used several times within that function

BUT! It **can't** be used outside of the **`dbl()`** function

That's because **x** is considered ***local*** to **`dbl()`**

Which of the Following Contains a *Function Call*?

```
1) type(4.5)
2) def dbl(x):
       return 2*x
3) area(2, 9)
4) print("Hello")
```

A. (3) only

B. (2) and (3)

C. (1), (3), and (4)

*D. **All** of them include a function call*

What is/are the Bug(s) in the Following Code?

```
def dbl(x):
return 2*x
y = 2
x = 5
dbl(y)
print(x, y, dbl(y))
```

A. No bugs. The code is fine

B. The function body is not indented

C. We are referring to x outside the definition of the function

D. Both B and C are bugs

Global vs. Local Variables:
What is the Output of this Code?

```
def dbl(x):
    return 2*x
y = 2
x = 5
x = dbl(y)
print(x, y, dbl(y))
```

A. 10 4 8

B. 5 2 4

C. 10 2 4

D. None of the above

Built-In (Fun)ctions for Strings

- Length of string: **len(*string*)**
  - Example: `len("Gaucho Greg")` is 11

- Consider a string called **st3** and that **len(st3) = 7**
  - What is the index of the LAST character in **st3**?

More (Fun)ctions!

- Boolean operators `in` and `not in` are great ways to check if a sub-string is found inside a longer string

Examples:

- `"fun" in "functions"` = True
- `"fun" in "Functions"` = False
- `"Fan" not in "Functions"` = True

*A **method** is like a function that's built-in for a class (like str) They are used with the "dot operator"*

String *Methods*

Try all of these out!

Assume: `name = 'Bubba'`

- `name.center(9)` is `'  Bubba  '` ← centers w/ spaces on each side
- `name.count('b')` is `2` ← counts how many times 'b' occurs
- `name.count('ubb')` is `1` ← counts how many times 'ubb' occurs
- `name.ljust(9)` is `'Bubba    '` ← left justifies name in 9 spaces
- `name.rjust(9)` is `'    Bubba'` ← right justifies name in 9 spaces
- `name.upper()` is `'BUBBA'` ← all uppercase letters
- `name.lower()` is `'bubba'` ← all lowercase letters
- `name.index('bb')` is `2` ← Index of first occurrence of first letter
- `name.find('bb')` is `2` ← Index of first occurrence of first letter
- `name.find('z')` is `-1` if not found, then returns -1
- `name.replace('bb','dd')` is `'Budda'` ← Replaces one sub-string for another

Let's try (some of these) out!

What if There are *Multiple* Parameters??

- When you call a function, the values you put in parenthesis have to be in the order in which they are listed in the definition!

- Example:

```
def subtract(m, n):
    return m - n
```

When you call this function to do a subtraction of 5 – 99, then:
m has to be 5 and **n has to be 99**
So, it's called as:
`subtract(5, 99)`
i.e. **not** `subtract(99, 5)`

What About… NO Parameters?!

Let's try it out!

- Sure, you can do that!

  But you still need the parentheses!

- Example:

```
def fortyTwo():
    return 42
```

All this function does is return the number 42 to whoever called it!

Which way should we call it?

fortyTwo

fortyTwo()

Wow. Functions are Cool.
Can They CALL EACH OTHER????

Yes!!!!!!!!!!!!!!! Careful that you get the order correct…!

```
def halve( x ):
""" returns half its input, x """
    return div(x, 2)

def div( y, x ):
""" returns y / x """
    return y / x
```

What happens when I say:

`>>> halve( 85 )`

A. I get 42
B. I get 42.5
C. 0
D. 0.02352 (i.e., 2 divided by 85)

Let's try it out!

A Function is a Function is a Function

- A function can be user-defined or can be built-into Python modules and classes

Example:

print() is a built-in Python **core** function that can be used in several ways (DEMO!)

sum()
max()
min()

are built-in Python core functions that can be used with lists (DEMO!)

Let's try it out!

Python Modules

- Python is open-sourced: *There are 10,000s of ready-made modules to use!*
- Popular ones include:
  - **math** has basic math/trig functions like sqrt(), sin(), cos(), pow()
  - **fractions** introduces the fraction type of var
  - **turtle** a popular graphic/drawing module

- Every time you want to use a module you have to ***import*** it first
  - Example: `import math`
- Every time you want to use a function that's in the module you have to use the dot operator
  - Example: `a = math.sqrt(5)  # a contains the square-root of 5`

Let's try it out!

YOUR TO-DOs

- ❑ Start reading **Chapter 3**
- ❑ Start on **HW2** (due next **Wednesday**)
- ❑ Do **Lab1** (turn it in by **Friday**)

- ❑ Embrace randomness

</LECTURE>