02IntroToFunctions

January 30, 2022

1 Culture and Coding: Python - Unit 1 Functions

The following 'book' and 'slides' is actually a jupyter notebook, which allows for Markdown and Python side by side. You may be reading this in the exported PDF form.

The notebook file is available on the CS0: Culture and Coding Github.

1.1 Introducing functions

1.1.1 Algebra introduced you to "functions"

- $f(x) = x^2$
- $f(3) = 9$
- $f(2) = 4$
- $f(5) = 25$
- But in coding, they are so much more!

1.1.2 Less technical

- A function is a repeatable and named set of instructions
  - We define a block of code
  - We give it a name
  - We allow values to be *passed* into the code
  - We return an answer (usually)

1.1.3 Why functions

- Keep the code we write in smaller chunks
- Helps with debugging and grading both

[ ]:
```
# The following is a valid python program

first = float(input("Enter your first number"))
second = float(input("Enter your second number"))
print("The first value is", first, "and the second value is", second)
print("The first times the second is", first*second)
print("The first divided by the second is", first / second)
print("The second divided by the first is", second / first)

first = float(input("Enter your first number"))
```

```
second = float(input("Enter your second number"))
print("The first value is", first, "and the second value is", second)
print("The first times the second is", first*second)
print("The first divided by the second is", first / second)
print("The second divided by the first is", second / first)

first = float(input("Enter your first number"))
second = float(input("Enter your second number"))
print("The first value is", first, "and the second value is", second)
print("The first times the second is", first*second)
print("The first divided by the second is", first / second)
print("The second divided by the first is", second / first)

first = float(input("Enter your first number"))
second = float(input("Enter your second number"))
print("The first value is", first, "and the second value is", second)
print("The first times the second is", first*second)
print("The first divided by the second is", first / second)
print("The second divided by the first is", second / first)

first = float(input("Enter your first number"))
second = float(input("Enter your second number"))
print("The first value is", first, "and the second value is", second)
print("The first times the second is", first*second)
print("The first divided by the second is", first / second)
print("The second divided by the first is", second / first)
```

```
The first value is 10.0 and the second value is 20.0
The first times the second is 200.0
The first divided by the second is 0.5
The second divided by the first is 2.0
The first value is 15.0 and the second value is 5.0
The first times the second is 75.0
The first divided by the second is 3.0
The second divided by the first is 0.3333333333333333
The first value is 100.0 and the second value is 99.0
The first times the second is 9900.0
The first divided by the second is 1.0101010101010102
The second divided by the first is 0.99
The first value is 2.0 and the second value is 3.0
The first times the second is 6.0
The first divided by the second is 0.6666666666666666
The second divided by the first is 1.5
The first value is 12.0 and the second value is 23.0
The first times the second is 276.0
The first divided by the second is 0.5217391304347826
The second divided by the first is 1.9166666666666667
```

Great! but… * My client just told me the output should look like..

```
10.0 * 20.0 is 200.0
10.0 / 20.0 is 0.5
20.0 / 10.0 is 2.0
```

- Now I have to change *every* item I copied and pasted my code!

1.2 Functions a better way to write code

- Allows for **resuable** blocks of code
- These resuable blocks of code
  - Are independant of the rest of the blocks (for now)
  - Allows for *isolation* of variables
  - Allows to easily fix and debug code
  - Keeps my code into smaller snippets or tasks
    - One task at a time.. means a zen to coding

1.2.1 Function format in python

```
def name(parameters):
  # a line of code
  # a line of code
```

- `def` means "**def**ine function"
- Name can be any valid identifier
- Parameters are values the function can use
  - They are variables that are **passed** into the function, like x in $f(x)$
  - You can have more than one, seperated by commas
- Let's write one

[ ]:
```
def multiplyDividePrint(first, second):
    print("The first value is", first, "and the second value is", second)
    print(first, "*", second,"is", first*second)
    print(first, "/", second,"is", first / second)
    print(second, "/", first, "is", second / first)
```

- Running this code has no output!
- What happens is:
  - `multiplyDividePrint(value1, value2)` is stored in memory
  - When I need to run those lines of code
    - Call or "invoke" the function using `multipleDividePrint(val, val)`
    - Similiar to how you have already been using `print()` (which is a function someone else wrote!)

[ ]:
```
val1 = float(input("Enter your first value: "))
val2 = float(input("Enter your second value: "))

multiplyDividePrint(val1, val2)  #the value in val1 and val2 gets copied to the␣
 ↪function
```

```
The first value is 42.0 and the second value is 21.0
42.0 * 21.0 is 882.0
42.0 / 21.0 is 2.0
21.0 / 42.0 is 0.5
```

1.3 Multiple Functions

You should liberally use functions, as you need. Take the above code, and it can be its own function

[ ]:
```
def tinyProgram():
    val1 = float(input("Enter your first value: "))
    val2 = float(input("Enter your second value: "))
    multiplyDividePrint(val1, val2)
```

And now, I can call `tinyProgram()` multiple times

[ ]:
```
tinyProgram()
print("-----") # just hear to make output cleaner
tinyProgram()
print("-----")
tinyProgram() # notice I need the parens - that is critical, even without␣
 ↪parameters
```

```
The first value is 42.0 and the second value is 21.0
42.0 * 21.0 is 882.0
42.0 / 21.0 is 2.0
21.0 / 42.0 is 0.5
-----
The first value is 42.0 and the second value is 21.0
42.0 * 21.0 is 882.0
42.0 / 21.0 is 2.0
21.0 / 42.0 is 0.5
-----
The first value is 22.0 and the second value is 11.0
22.0 * 11.0 is 242.0
22.0 / 11.0 is 2.0
11.0 / 22.0 is 0.5
```

1.4 Return Types

- A key feature of functional programming
  - functions should ideally return answers
  - Go do the work, give me an answer to use
- `return` is a command to:
  - return the value of whatever is after it (variable usually)
  - the value is then used in place of the function in the calling code

[ ]:
```
def getDivideString(firstValue, secondValue):
```

```
    rtn_str = str(firstValue) + " / " + str(secondValue) + " is " + str(␣
 ↪firstValue / secondValue ) + "."
    return rtn_str
```

Note: we will come up with a better way to write the string concatination next lecture.

[ ]:
```
def tinyProgramWithReturns():
    val1 = int(input("Enter your first value: "))
    val2 = int(input("Enter your second value: "))
    answer = getDivideString(val1, val2)
    print(answer)
    print(getDivideString(val2, val1))

tinyProgramWithReturns()
```

```
42 / 21 is 2.0.
21 / 42 is 0.5.
```

In `tinyProgramWithReturns()` we * asked for client input, storing the answer * notice both int() and input() are functions, that both return values * input returns the client input as a string * int() takes the string, and returns an whole number * called `getDivideString` with val1 being copied for firstValue, and val2 with secondValue * we stored the returned string into `answer` * printed the value stored in `answer` * called `getDivideString` with val1 being copoied into SecondValue, and val2 with firstValue * notice: the same function is used, but in different ways * the value returned is passed into the `print()` function

Notice We can use the function in a variety of ways, but we just **focus** on a value is being given back to use.

1.5 Inclass Coding Activity: Your First Functions

There is an inclass activity in your textbook. Go to canvas to click through the link. * Second one person at your table to program * Ideally the person with the *least* experience * Everyone work and guide that person in solving the problem! * For this activity, you will write two functions to help build a working program.

1.6 Important "Gotchas" to watch for

- Python processes lines in order
  - which means you want to define a function *before* using it!
- the *scope* of a function is local
  - meaning the variables only exist in the function
  - unless you return them (or do something special)

[ ]:
```
first = 10

def myfunction():
    first = "hello"
    print(first)  # what is printed to the screen
```

```
myfunction()
print(first) # what is printed to the screen
```

```
hello
10
```

1.7 Thinking Further: Functional Programming

- Focus on divide (decomposition)
  - your problem is large, break into smaller parts
- Functions should be "pure"
  - use parameters
  - return values
  - only one task
  - don't modify other code (no side affects)
- Helps with maintainablity
  - hot code deployment
  - unit testing is easier
  - lazy loading
- The most common programming model for web application programming
  - is a functional model
  - objects that load at different rates
  - and must be maintained based on the return values
- Overall: It is a way to **think** about programming
  - we move from steps to repeatable actions that build