Topic 5: Functions

Goals: By the end of this topic, we will discuss…

- *how to write and call functions*
- *passing parameters to and returning values from functions*
- *see how using functions can results in better code reuse*

Acknowledgements: These class notes build on the content of my previous courses as well as the work of R. Jordan Crouser, and Jeffrey S. Castrucci.

Recall: Math

We have already used many built-in functions in this course:
print(…)
input(…)
eval(…)
round(…)

We have also used function in math.py and random.py.
Perhaps unsurprisingly, Python lets us write custom functions as well (like the **main()** function).

Functions

A **function** is a procedure / routine that takes in some input and does something with it (just like in math).
Note: We will examine a more precise definition later.

```
def doSomething():
     # perform some operations, like:
     x = 2 + 3

     # send stuff back to the main program
     return x
```

- We define a function with the keyword `def`.
- `doSomthing` is the function name.
  - Convention: use `_uderscores` or `camelCase` to name functions.
  - The `()` after the name indicate that it is a function.
- The Body of the function is indented.
- A `return` value/variable is optional.

Question: What happens if we run the program above.

Answer: Nothing, because a “function definition” is a description, but not a directive.

We need to add a function call.
`doSomething()` outside the def is a function call.

```
def doSomething():
     x = 2 + 3
     return x

y = doSomething()
print(y)
```

Functions explained through minions (and Gru)….

Functions are like minions… see: https://www.imdb.com/title/tt2293640/

* All functions have names…

Functions only operate (or work) when you CALL them…

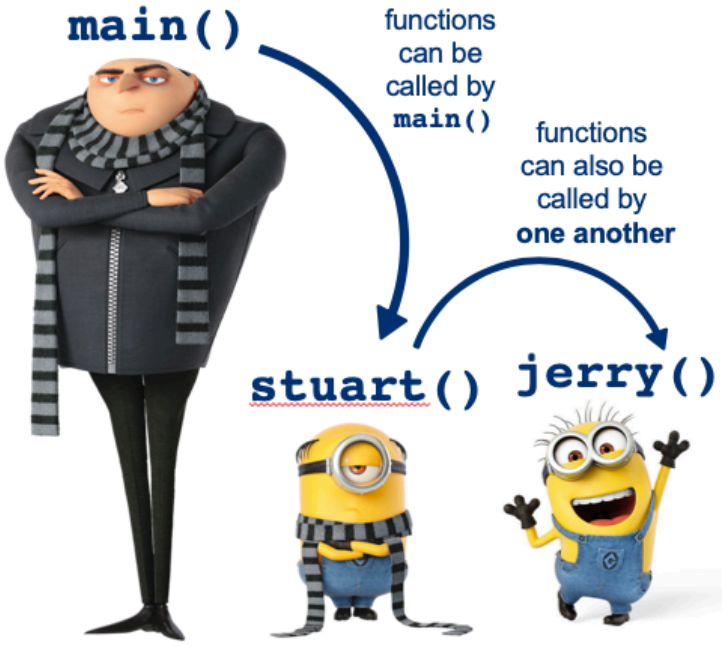
Functions can be called by `main()` and by each other…

Parameters

Functions can either do the same thing every time or they can adjust their behavior based on what we give them.

```
def printStars():
    print("*"*25)
printStars()
printStars()
```

```
def printStars(x)
    print("*"*x)
printStars(5)
printStars(32)
```



`printStars()` `printStars(1527)`

- x is considered a "parameter" that is passed into the function.

Exercise: Happy Birthday

Write a function called happyBirthday(name) that takes in a string name and prints out the lyrics to the song "Happy Birthday" with the name inserted:

Happy birthday to you!
Happy birthday to you!
Happy birthday, dear NAME
Happy birthday to you!

Call this function from inside the main() function, and use input(…) to get the value of name from the user.

Multiple and Default Parameters

- Functions can be defined to take in multiple parameters:

```
def emphasize(word, char):
    print(char.join(list(word)))
emphasize("Monday", "-")
```

Output:
M-o-n-d-a-y
`word = "Tuesday"` and `char = "-"` inside `emphasize`.

- We can include a “default” value for some (or all) of them:

```
def emphasize(word, char = "*"):
    print(char.join(list(word)))
emphasize("Monday")
```

In this case we can call emphasize with only one parameter.
Output:
M*o*n*d*a*y

Returning Values

We may want to **return** the results rather than print them:

```
def emphasize(word, char = "*"):
     return char.join(list(word))

boom = emphasize("Monday")
```

- The result of the return statement in emphasize() is that the "M*o*n*d*a*y" is assigned to boom.

Advanced Topic: Chaining Functions

Return values allow us to call functions inside other function calls:

```
>>> n = eval(input("Enter an integer: "))
>>> n = eval("3")
```

Similarly,

```
boom = emphasize(input("Enter a word: "))
```

Note: Count the number of open and closed bracket you have.

Bonus Activity (aka if time allows):

Start with your "create a header" code from a previous lab, and put the code into functions that get the user input and create the visual header, using the same four variables (name, filename, section, and date). What happens if you used a default parameter for when you don't have a partner.

Start on paper/whiteboard and figure out what functions you need and what variables will be passed to and returned from each function.

Definitions vs. Calls

Function Definition

•Step-by-step instructions for how to perform a given set of operations

•Analogy: a recipe

•Think of the function's name as shorthand (i.e. "okay, when I say do_something(), here's what I want you to do")

Function Call

•An actual request to perform the operations

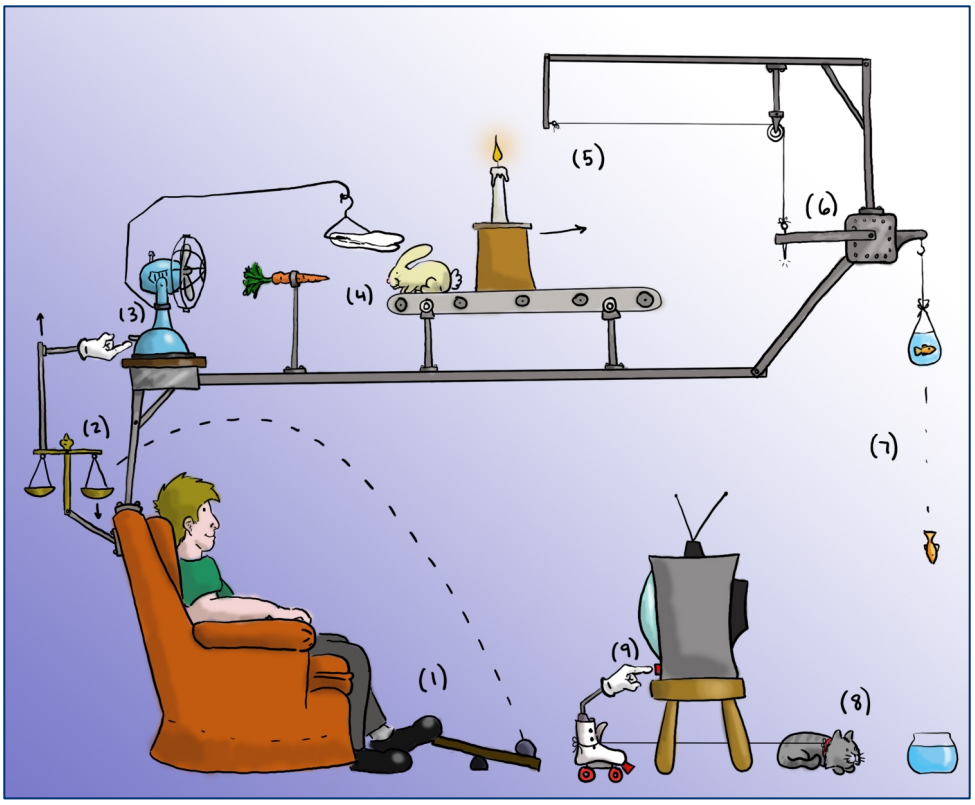
•Control is turned over to the "minion" (temporarily)

•Once complete, we go back to the exact place in the program where the call was issued

Question: What does it mean for a function to return a value?

And what's the difference between return and print(…)?

Class Activity: Human Rube Goldberg Machine



[Citation: “Remote” - by philister]

- We have four functions.

def addOne(x):
  Take the value of x,
  and add 1 to it
  return the modified value
	def print(x):
	  Look at the value of x
	  Write it on the board in big
	  letters
	  return nothing

def doubleIt(x):
  Take the value of x, and double
  it (i.e. multiply by 2)
  return the modified value
	def woohoo():
	  Say out loud a single,
	  loud “BINGO!”
	  return nothing

Consider the following program:

```
def main():
     a = addOne(5)
     print(a)
     woohoo()
     b = doubleIt(a)
     print(b)
     print(doubleIt(addOne(b)))
main()
```

Do you see the difference between return and print?

What is the point of the yarn (i.e., the thread)?

This demonstrates the trace of the program.