Functions in Python

October 15

Administrivia

Project 1 available

Makes use of functions - a lot

Due Date

Get started early!!!

Functions in general

A function is the idealization of the relationship between one quantity and another

- The *location* of a planet is a function of *time* - if we know how much *time* has passed, we know where the planet is

In math, a function is a mapping from one space to a second space - the same or different

- $f(x) = 3x2 + x - 2$ maps from the x-axis to the y-axis. If you prefer it maps from real numbers to real numbers

Functions in programming

A block of code that takes a set of *inputs* or *parameters;* does something; and may also return a set of *outputs* or *results*

Python comes with a whole set of built-in functions - somebody else wrote them and included them with the interpreter

- print() - the inputs are whatever you put between the parentheses; the function writes something to the screen or somewhere else;
- input () - the input to this function are whatever you put in between the parentheses; the function prints that on the screen and waits for the user to do something. The function returns whatever was typed, encoded as a string

Other Python built in functions

int() - takes some value and returns its integer representation

str() - takes some value and returns its string representation

...

Writing your own functions in Python

Python wouldn’t be a very useful language if it didn’t let you write your own functions to do what you need

There are two parts - *defining* the function and *calling* or *invoking* the function

Defining the function means writing out the code that will be executed

- Start with the reserved word ‘def’
- Then have the function name
- Then the parameters, enclosed in parentheses
- Then the code

Example: A function to reverse the characters in a string

```
#  First the function definition
def reverse_word (word)
# now the code. DON'T FORGET TO INDENT!!
      i = 0
      reversed_word = ‘’  #Do not use the function name here!!!!
      while i < len(word):
            reversed_word = reversed_word + word [i]
      print (“ The word “, word, “ reversed is “, reversed_word)
```

Calling your function

In the main body of your program:

```
animals = [“dog”, “cat”, “platypus”, “ocelot”]
for critter in animals:
        reverse_word(critter)
```

Note:

- The function *must* be defined before it can be called. The “def” statement has to come earlier in your code than the call. Otherwise, the Python interpreter won’t know what you’re talking about when you call the function

When would you use a function:

When you want to be able to do something multiple times in this program and in other programs

- When it’s more complicated than just a simple loop can handle
- When doing this in the main body of your code results in a twisted mess of nested loops!!
-

Variables and Scope

“Scope” of a variable refers to the parts of your program where that variable is defined and can be used

Functions have two types of variables:

parameters - those things passed in during the function call

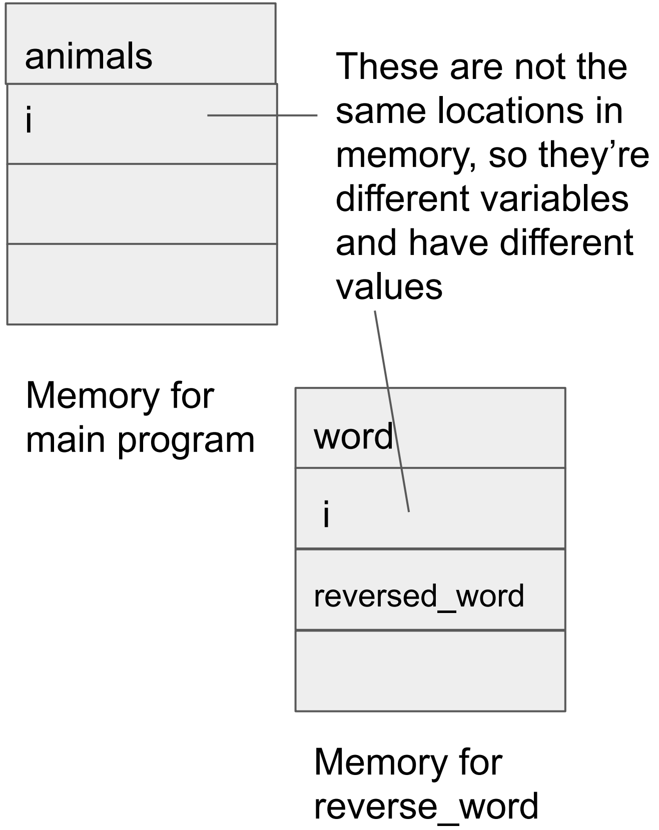
local variables - those things defined and use in the function’s code

Both parameters and local variables are only known within the function. That is, they only have meaning when the function is running. If there’s another variable of the same name somewhere else in the program, that’s okay. Python can tell the difference

Memory management for functions

```
#  First the function definition
def reverse_word (word)
# now the code. DON'T FORGET TO INDENT!!
        i = 0
        reversed_word = ''  #Do not use the function name here!!!!
        while i < len(word):
                reversed_word = reversed_word + word [i]
        print (" The word ", word, " reversed is ", reversed_word)
…

animals = ["dog", "cat", "platypus", "ocelot"]
        for i  in animals:
                reverse_word(i)
```


animals
i
These are not the same locations in memory, so they're different variables and have different values
Memory for main program
word
i
reversed_word
Memory for reverse_word

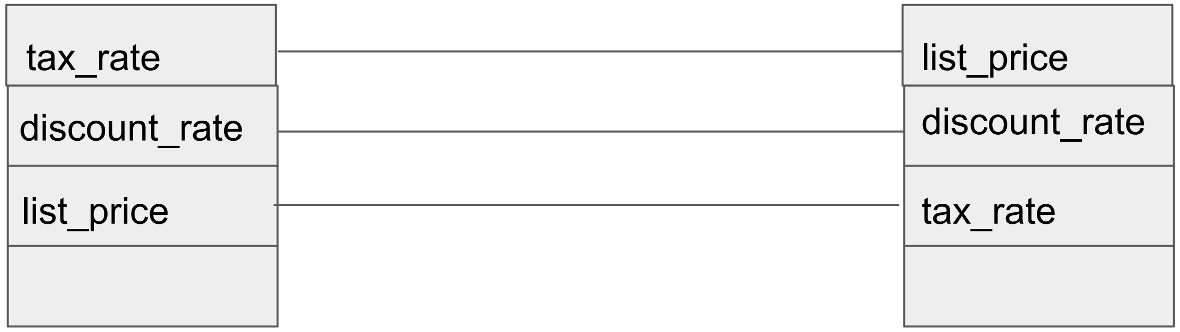
Order of parameters counts!!

Python matches parameters by the order in the call, NOT by the name

```
# A function to calculate a student discount price
# You get a discount off the list price of a book. But you have to
# pay sales tax on the full list price
def discount_price (list_price, discount_rate, tax_rate)
        amount_off = discount_rate * list_price
        price_paid = list_price * (1 + tax_rate) - amount_off
        print("Your final price is ", price_paid)
#now we'll try calling with different parameter orders
list_price = 100.00
discount_rate = 0.2 # 20 percent off
tax_rate = 0.06      # 6 percent sales tax
discount_price(list_price, discount_rate, tax_rate)
discount_price (discount_rate, tax_rate, list_price)
discount_price (tax_rate, discount_rate, list_price)
```

What’s happening in memory

The Python interpreter is matching the first parameter in the call with the first parameter in the definition. It is *not* matching on names!!



Memory for
main program

Memory for
discount_price

“Call by value” vs “Call by reference”

“Call by value” means that when you call a function, the values of each parameter are copied into the function’s memory and the original variables are left untouched.

“Call by reference” means that when you call a function, you pass it the actual memory locations holding the parameters and thus the function can change the main program.

Python mostly uses “call by value.” For 201 and 202, this is good enough. Much later in your programming career you’ll discover that I’m lying here. But the actual way that Python passes parameters is pretty complex, and you don’t need to understand it until much later.