# CSI 1401 Week 9 Resources

*by Allen Yan*

Functions are incredibly useful in programming, so much so that CSI 1401 has dedicated three weeks to it. Last week we covered the basics of function definition. This week, we will look at some of the common errors and caveats to defining functions.

As a reminder, I will be leading a group tutoring session on CSI 1401 every Wednesday from 7:00 pm to 8:00 pm (central time). The session will be conducted online via Microsoft Meetings, where I will be available to provide interactive help to students. If the above time window does not work for you, or if you need additional help, Baylor's Tutoring Center also provides 1-on-1 online tutoring appointments. For more information on signing up for group tutoring or individual tutoring, please visit https://www.baylor.edu/tutoring.

## Week 9 Important Topics

**Chapter 6 – Functions**

- Function stubs
- Functions are objects
- Common errors

### 1. Function stubs

We mentioned *modular* or *incremental development* last week. It is a good coding practice involving splitting a large program into small sections of code that can be independently written and tested, before getting assembled into one large program that may be difficult to test. Python has some features that are helpful for incremental development.

The first useful tool for incremental development is the *function stubs*. They are essentially unimplemented functions with just a header and nothing in their definition. When working on a large project, function stubs allow the programmer to write out the basic structures and behaviors of the program without needing to dive deep into the details. See below for an example function stub.

```
def i_am_a_stub(arg1, arg2):
    pass
```

Another useful tool is the `pass` keyword, as shown being used in the above example. The `pass` keyword effectively does nothing, but it serves as a good placeholder in a function stub, because all functions are required to have at least one line of definition, otherwise the program will not be able to run. If you try to run a function stub with the only `pass` keyword as its definition, you will see that it does nothing.

More often than not, it is a good idea to prevent the program from running unimplemented functions at all in order to avoid unexpected behaviors. This can be done by telling the function stub to raise a *NotImplementedError* when the program tries to call on it. See example below.

```
def i_am_a_stub(arg1, arg2):
    raise NotImplementedError
```

## 2. Functions are objects

All functions in Python are also objects, which means they can be treated like a typical variable. Python objects consist of series of *bytecode*, which are low-level operations, such as addition and subtraction at the memory level, that are generated when the function is compiled. Unlike other programming languages, Python compiles the function automatically when evaluating the function definition. See the textbook example below on using functions as objects:

```
def head():
    print('   ||||| ')
    print('   o   o')
    print('     >' )
    print('   ooooo')


def print_figure(face):
    face()  # Print the face
    print('     |')
    print('   --|--')
    print('  /  |  \\')
    print('@    |    @')
    print('     |')
    print('    /|\\')
    print('   @   @')


print_figure(head)
```

## 3. Common errors

Error by returning at the wrong time was already brought up last week, but it is such a common mistake that it deserves a spot in the common errors section. Compared to other mistakes such as accidentally returning the wrong value, or copy and pasting similar functions to develop other functions and forgetting to change some key parts, this error is more prevalent, and sometimes not as easy to spot. See the example below:

```
def add_bad(my_list):  # bad function
    my_sum = 0
    return my_sum  # returns here
    for elem in my_list:  # this loop will never be reached
        my_sum += elem


def add_good(my_list):  # good function
```

```
    my_sum = 0
    for elem in my_list:  # adds properly
        my_sum += elem
    return my_sum  # returns here


nums = [1, 2, 3, 4, 5]
print(add_bad(nums))  # prints 0
print(add_good(nums))  # prints the sum of nums
```

A good thing to always keep in mind when defining functions is that **as soon as a return statement is reached, the function exits**, no matter how many more lines remain in the function definition after the return statement. So always make sure the function has completed all of its intended tasks before the return statement is called.

Another common mistake is calling functions before it is declared. Python interprets the code line-by-line in a sequential manner. If a function is not defined by the time it is called, an error will be triggered. See example below.

```
employee_name = 'N/A'

get_name()
print('Employee name:', employee_name)

def get_name():
    global employee_name
    name = input('Enter employee name:')
    employee_name = name
```

The above code will not run because the function `get_name()` is not defined before the first time it is called. Always make sure to define your functions before calling them. Sometimes this may cause a readability problem by having too much function definition code on top of the main program. In such cases, write the function definitions in a separate file and use the `import` keyword.

## Useful Resources

- Basic Python Tutorial on GeeksforGeeks:
  https://www.geeksforgeeks.org/python-programming-language/
    - This page provides links to detailed explanations to many entry-level Python concepts along with examples. I encourage taking a look at it if the examples in your textbook were not clear enough.
- Official Python Documentation:
  https://docs.python.org/3/
    - This may be a bit heavy-handed for a beginner-level programmer since the official Python documentation is very thorough and technical. However, learning how to read official documentations is crucial to becoming a good programmer, because the official documentation contains information on everything you can find about Python elsewhere and more. Therefore, I encourage slowly easing yourself into learning how to read the official documentation.