Functions

Leen-Kiat Soh

Computer Science & Engineering

University of Nebraska, Lincoln, NE

Consider the following programming code

```
print("Happy Birthday to you!")
print("Happy Birthday to you!")
print("Happy Birthday, dear Emily.")
print("Happy Birthday to you!")
```

What if you are required to do the above sequence of code N times?

Consider the following programming code

```
print("Happy Birthday to you!")
print("Happy Birthday to you!")
print("Happy Birthday, dear Emily.")
print("Happy Birthday to you!")
print("Happy Birthday to you!")
print("Happy Birthday to you!")
print("Happy Birthday, dear Emily.")
print("Happy Birthday to you!")
print("Happy Birthday to you!")
print("Happy Birthday to you!")
print("Happy Birthday, dear Emily.")
print("Happy Birthday to you!")
…
…
…
```

A lot of lines of code repeating almost exactly the same thing …

Consider the following programming code

```
for x in range(0,N):
    print("Happy Birthday to you!")
    print("Happy Birthday to you!")
    print("Happy Birthday, dear Emily.")
    print("Happy Birthday to you!")
```

But, what if we want to print the Happy Birthday songs to different people?

```
for x in range(0,N):
    print("Happy Birthday to you!")
    print("Happy Birthday to you!")
    print("Happy Birthday, dear Emily.")
    print("Happy Birthday to you!")
for x in range(0,N):
    print("Happy Birthday to you!")
    print("Happy Birthday to you!")
    print("Happy Birthday, dear Foobar.")
    print("Happy Birthday to you!")
for x in range(0,N):
    print("Happy Birthday to you!")
    print("Happy Birthday to you!")
    print("Happy Birthday, dear Alibaba.")
    print("Happy Birthday to you!")
…
```

A lot of lines of code repeating almost exactly the same thing …

Now, what if we want to modify parts of the lyrics?

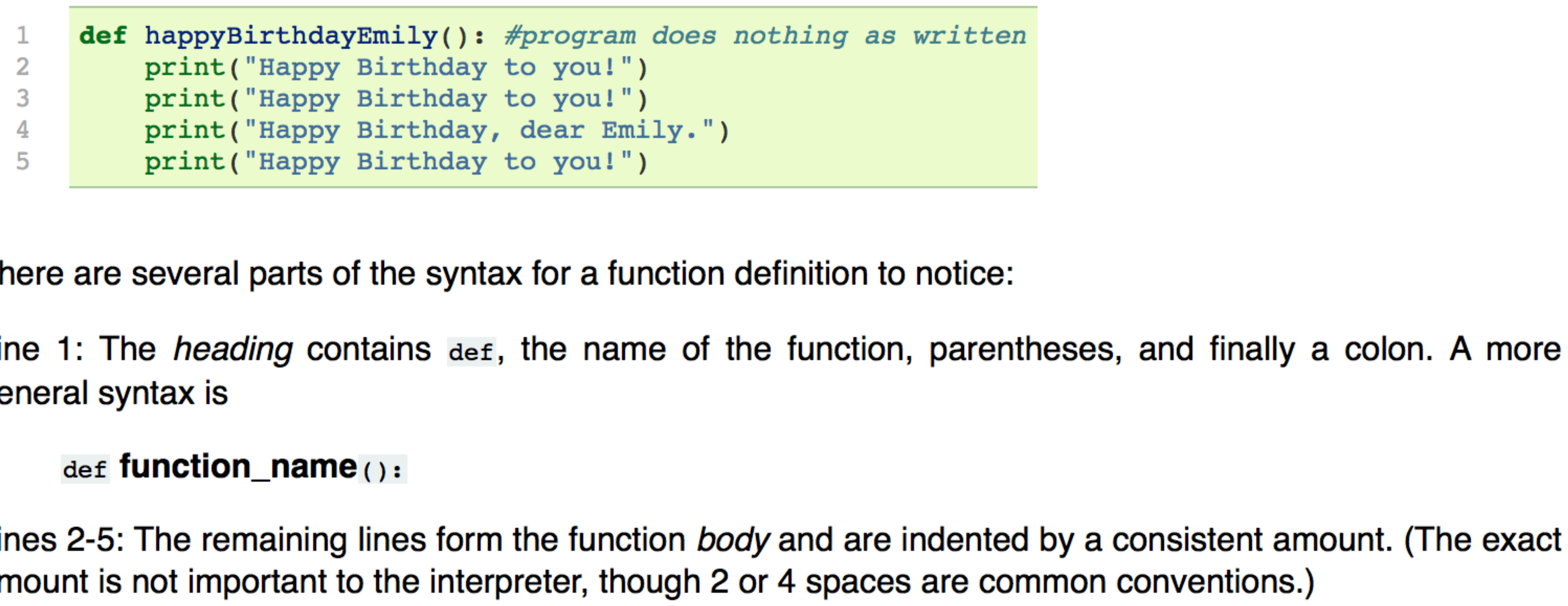
```
for x in range(0,N):
    print("Happy Big Red Day to you!")
    print("Happy Big Red Day to you!")
    print("Happy Big Red Day, dear Emily.")
    print("Happy Big Red Day to you!")
for x in range(0,N):
    print("Happy Big Red Day to you!")
    print("Happy Big Red Day to you!")
    print("Happy Big Red Day, dear Foobar.")
    print("Happy Big Red Day to you!")
for x in range(0,N):
    print("Happy Big Red Day to you!")
    print("Happy Big Red Day to you!")
    print("Happy Big Red Day, dear Alibaba.")
    print("Happy Big Red Day to you!")
…
```

A lot of lines of code to modify … could lead to errors or inconsistencies

Functions: Three Key Advantages

- Modularity
  - Allows us to design our solutions modularly
    - Breaking down a problem using decomposition and mapping it to solution design
  - Protects data (maintains data integrity)
- Reusability
  - Functions can be called or used by other programmers in their program
    - Think about what Python has done for us with their functions: print(), input(), the functions in the Math package, etc.
  - Reduces the lines of code that need to be written, inspected, tested, and compiled
  - Improves the speed of solution implementation
    - Can use previously tested and validated functions
- Maintainability
  - Fewer lines of code to modify or maintain

Functions Syntax



```
1  def happyBirthdayEmily(): #program does nothing as written
2      print("Happy Birthday to you!")
3      print("Happy Birthday to you!")
4      print("Happy Birthday, dear Emily.")
5      print("Happy Birthday to you!")
```

There are several parts of the syntax for a function definition to notice:

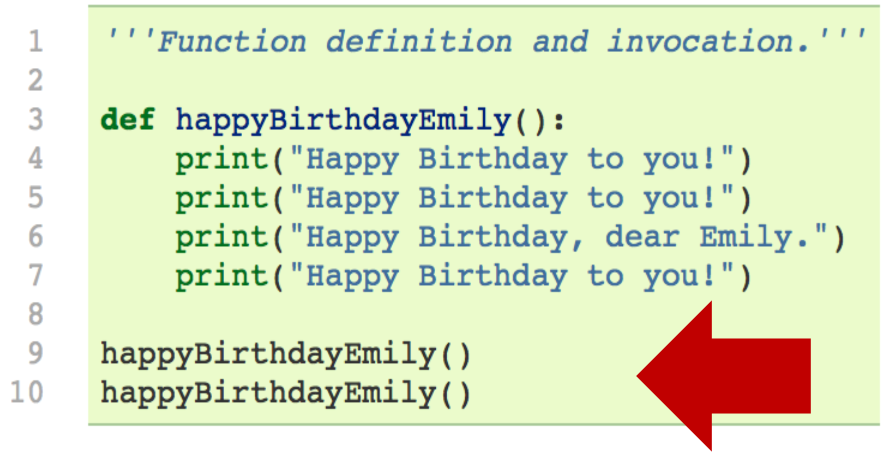
Line 1: The *heading* contains `def`, the name of the function, parentheses, and finally a colon. A more general syntax is

`def` **function_name**`():`

Lines 2-5: The remaining lines form the function *body* and are indented by a consistent amount. (The exact amount is not important to the interpreter, though 2 or 4 spaces are common conventions.)

http://anh.cs.luc.edu/handsonPythonTutorial/functions.html

How to Call a Function Syntax



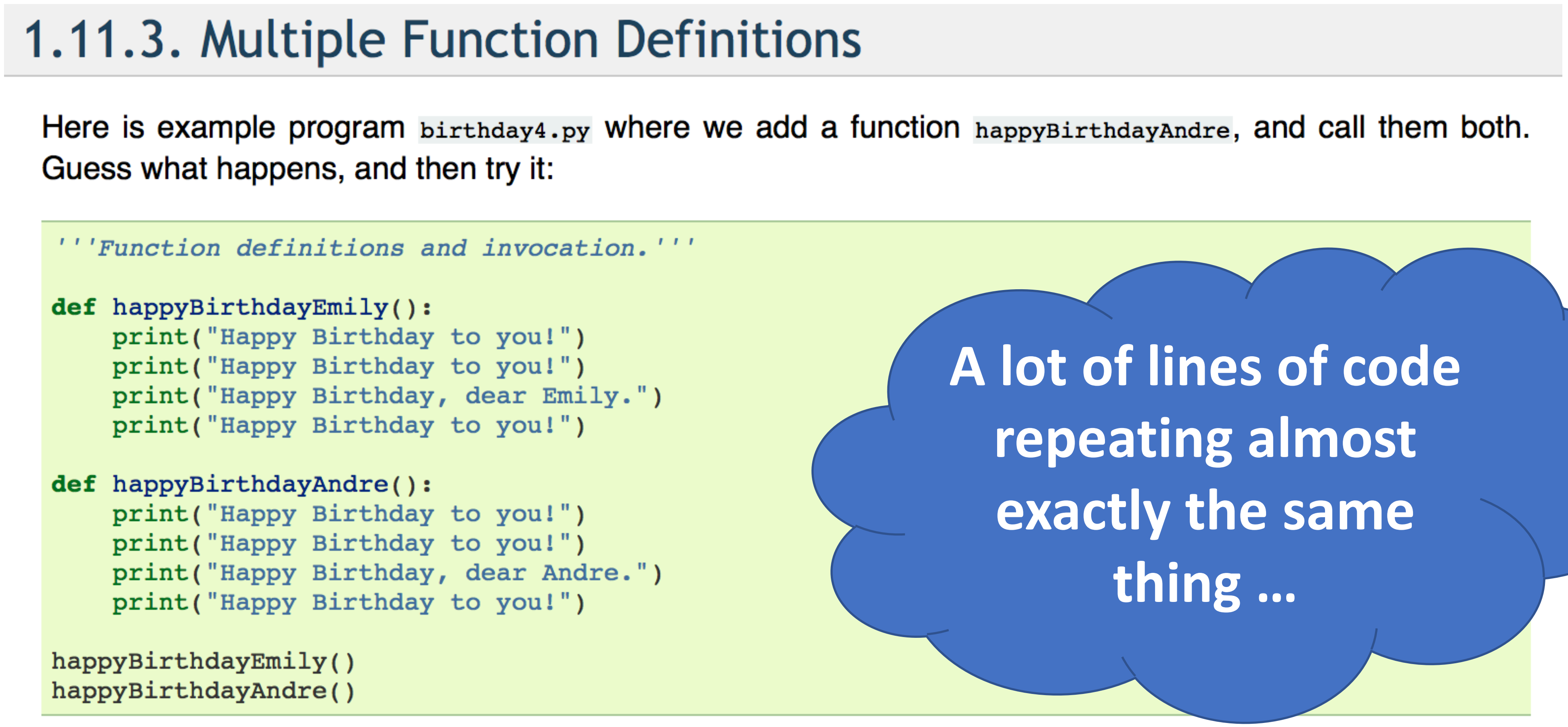
```
1  '''Function definition and invocation.'''
2
3  def happyBirthdayEmily():
4      print("Happy Birthday to you!")
5      print("Happy Birthday to you!")
6      print("Happy Birthday, dear Emily.")
7      print("Happy Birthday to you!")
8
9  happyBirthdayEmily()
10 happyBirthdayEmily()
```

The *execution* sequence is different from the *textual* sequence:

1. Lines 3-7: Python starts from the top, reading and remembering the definition. The definition ends where the indentation ends. (The code also shows a blank line there, but that is only for humans, to emphasize the end of the definition.)
2. Line 9: this is not indented inside any definition, so the interpreter executes it directly, calling `happyBirthdayEmily()` while remembering where to return.
3. Lines 3-7: The code of the function is executed for the first time, printing out the song.
4. End of line 9: Back from the function call. continue on.
5. Line 10: the function is called again while this location is remembered.
6. Lines 3-7: The function is executed again, printing out the song again.
7. End of line 10: Back from the function call, but at this point there is nothing more in the program, and execution stops.

http://anh.cs.luc.edu/handsonPythonTutorial/functions.html

Multiple Functions Syntax



1.11.3. Multiple Function Definitions

Here is example program `birthday4.py` where we add a function `happyBirthdayAndre`, and call them both. Guess what happens, and then try it:

```python
'''Function definitions and invocation.'''

def happyBirthdayEmily():
    print("Happy Birthday to you!")
    print("Happy Birthday to you!")
    print("Happy Birthday, dear Emily.")
    print("Happy Birthday to you!")

def happyBirthdayAndre():
    print("Happy Birthday to you!")
    print("Happy Birthday to you!")
    print("Happy Birthday, dear Andre.")
    print("Happy Birthday to you!")

happyBirthdayEmily()
happyBirthdayAndre()
```

A lot of lines of code repeating almost exactly the same thing …

http://anh.cs.luc.edu/handsonPythonTutorial/functions.html

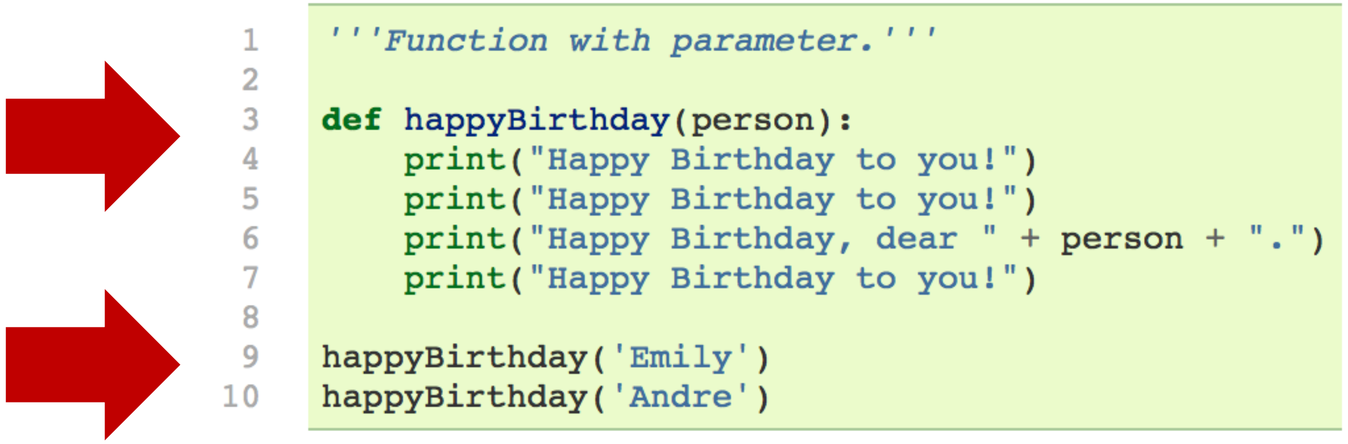
Functions: Parameters and Arguments

- Formal Parameters
  - When you define your function, you can define it to accept values
  - In the function **definition**, a variable that stores a value passed to the function is called a **formal parameter**
- Actual Parameters (a.k.a. Arguments)
  - In the **function call**, a variable passed to the function is called an **actual parameter** or an **argument**

http://anh.cs.luc.edu/handsonPythonTutorial/functions.html

Functions with Parameters Syntax

The function definition indicates that the variable name `person` will be used inside the function by inserting it between the parentheses of the definition. Then in the body of the definition of the function, person is used in place of the real data for any specific person's name. Read and then run example program `birthday6.py`:



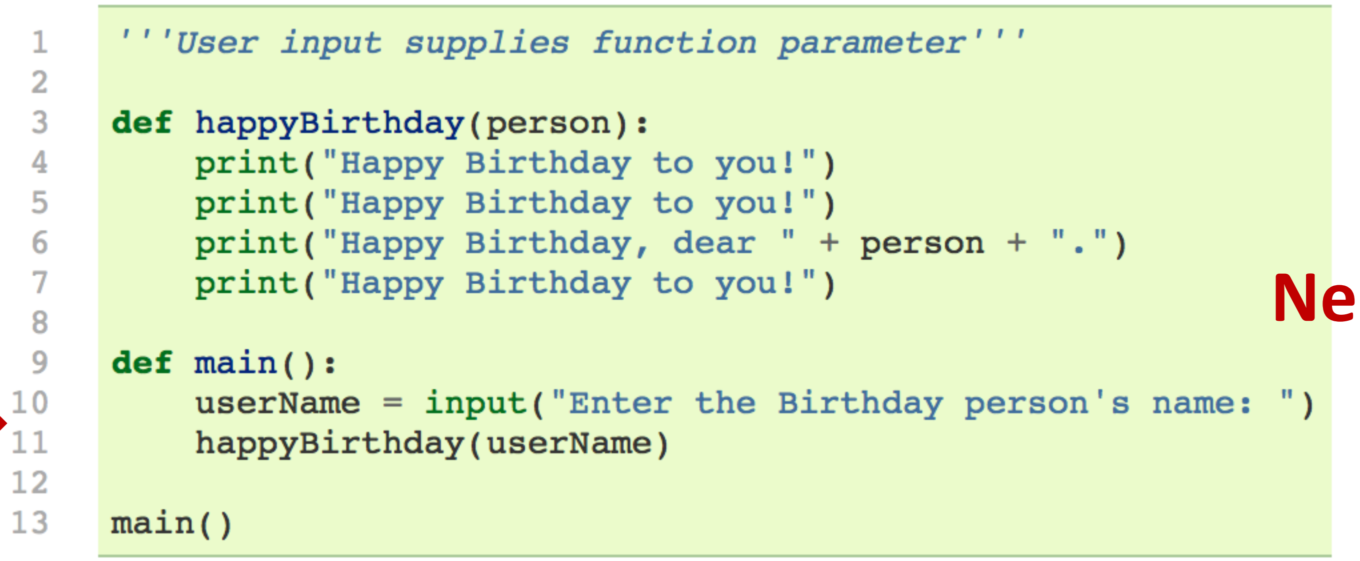
```python
'''Function with parameter.'''

def happyBirthday(person):
    print("Happy Birthday to you!")
    print("Happy Birthday to you!")
    print("Happy Birthday, dear " + person + ".")
    print("Happy Birthday to you!")

happyBirthday('Emily')
happyBirthday('Andre')
```

In the definition heading for `happyBirthday`, `person` is referred to as a *formal parameter*. This variable name is a placeholder for the real name of the person being sung to.

http://anh.cs.luc.edu/handsonPythonTutorial/functions.html

Other Ideas: Nested Function Call

We can combine function parameters with user input, and have the program be able to print Happy Birthday for anyone. Check out the main method and run `birthday_who.py`:

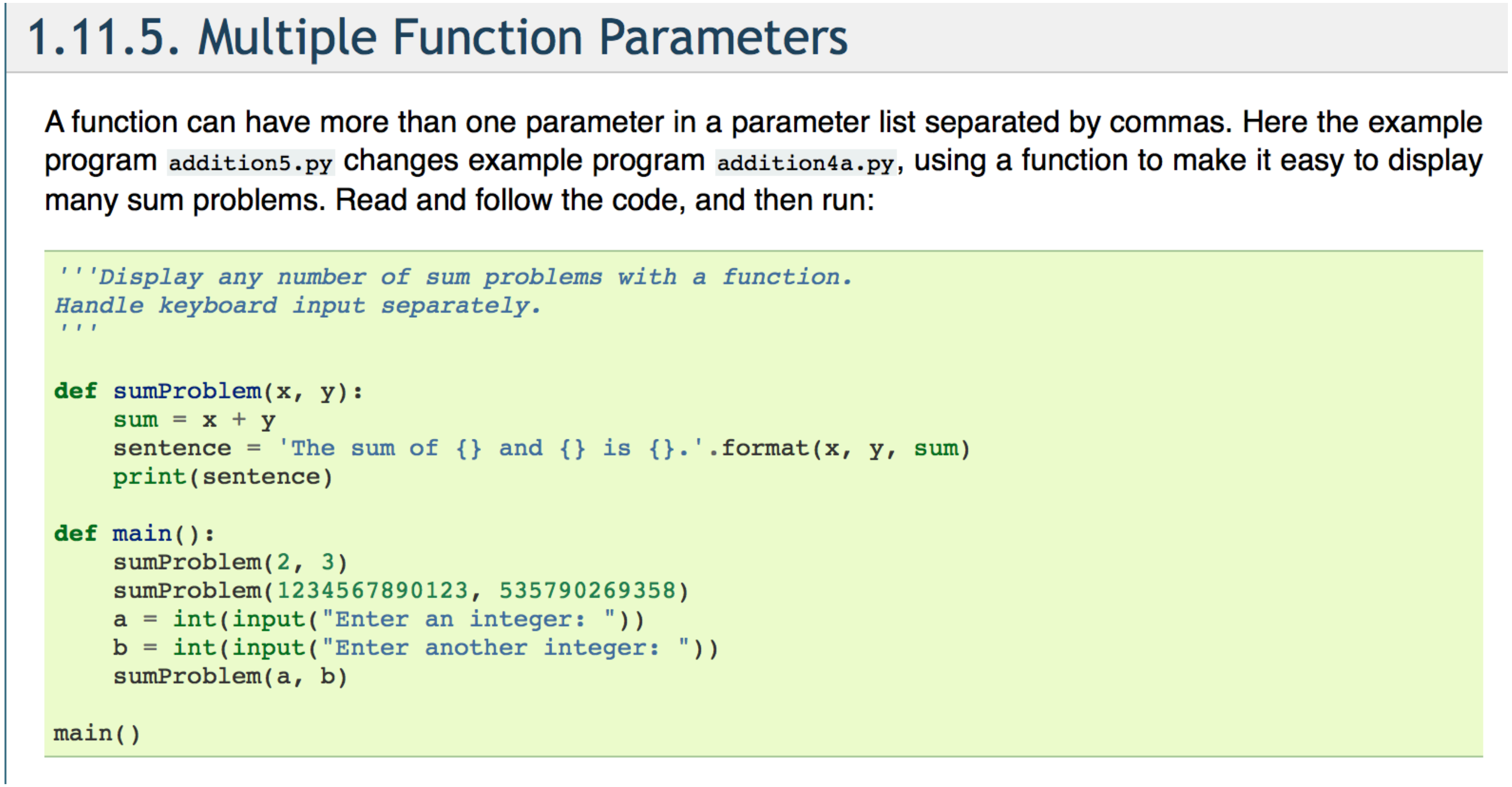


```python
'''User input supplies function parameter'''

def happyBirthday(person):
    print("Happy Birthday to you!")
    print("Happy Birthday to you!")
    print("Happy Birthday, dear " + person + ".")
    print("Happy Birthday to you!")

def main():
    userName = input("Enter the Birthday person's name: ")
    happyBirthday(userName)

main()
```

Nested function call!!

http://anh.cs.luc.edu/handsonPythonTutorial/functions.html

Other Ideas: Multiple Parameters



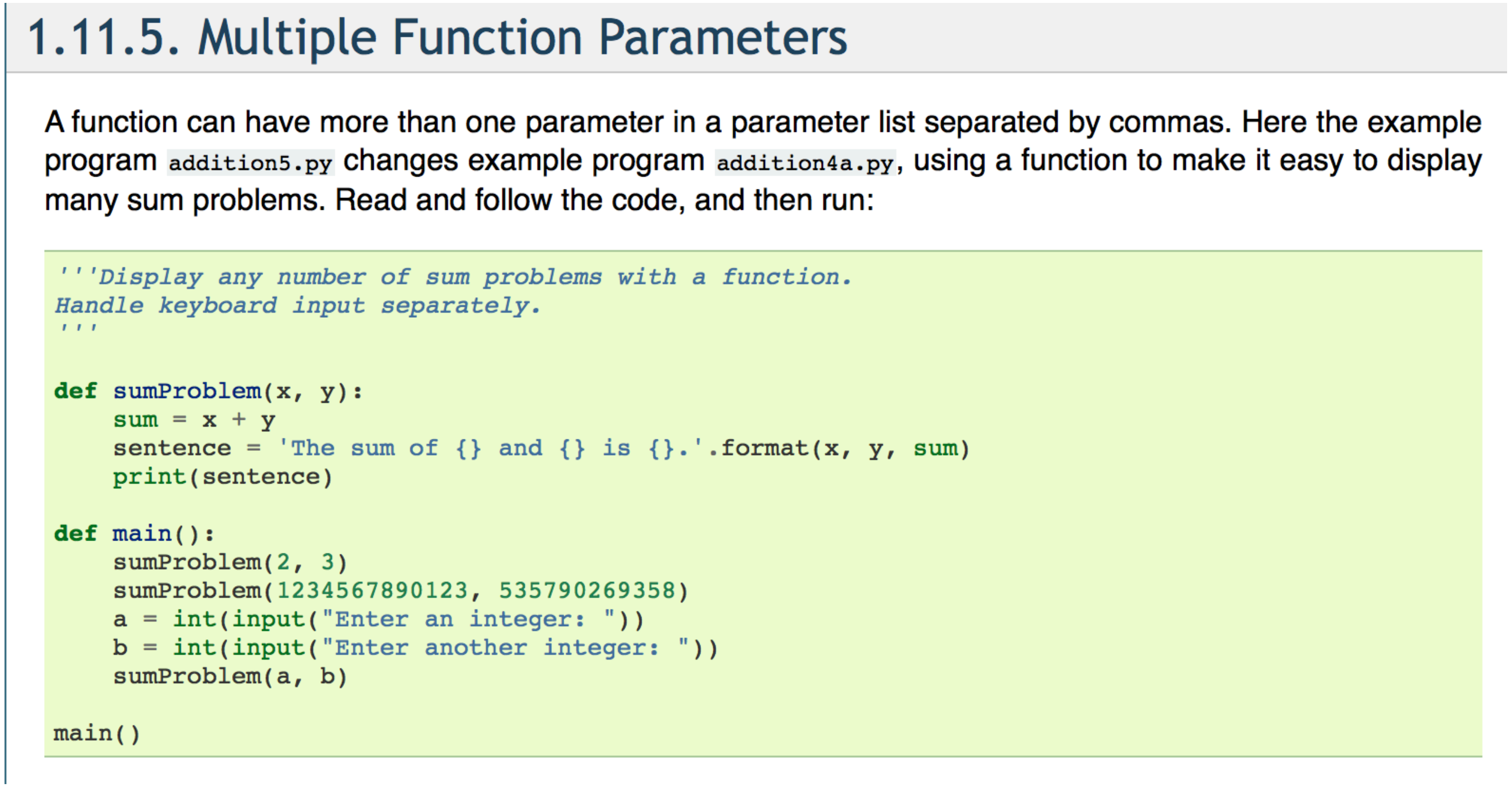
1.11.5. Multiple Function Parameters

A function can have more than one parameter in a parameter list separated by commas. Here the example program `addition5.py` changes example program `addition4a.py`, using a function to make it easy to display many sum problems. Read and follow the code, and then run:

```
'''Display any number of sum problems with a function.
Handle keyboard input separately.
'''

def sumProblem(x, y):
    sum = x + y
    sentence = 'The sum of {} and {} is {}.'.format(x, y, sum)
    print(sentence)

def main():
    sumProblem(2, 3)
    sumProblem(1234567890123, 535790269358)
    a = int(input("Enter an integer: "))
    b = int(input("Enter another integer: "))
    sumProblem(a, b)

main()
```

http://anh.cs.luc.edu/handsonPythonTutorial/functions.html

Other Ideas: Multiple Parameters



1.11.5. Multiple Function Parameters

A function can have more than one parameter in a parameter list separated by commas. Here the example program `addition5.py` changes example program `addition4a.py`, using a function to make it easy to display many sum problems. Read and follow the code, and then run:

```python
'''Display any number of sum problems with a function.
Handle keyboard input separately.
'''

def sumProblem(x, y):
    sum = x + y
    sentence = 'The sum of {} and {} is {}.'.format(x, y, sum)
    print(sentence)

def main():
    sumProblem(2, 3)
    sumProblem(1234567890123, 535790269358)
    a = int(input("Enter an integer: "))
    b = int(input("Enter another integer: "))
    sumProblem(a, b)

main()
```

http://anh.cs.luc.edu/handsonPythonTutorial/functions.html

Functions: To Return or To Not Return …

- So far, we have demonstrated functions that only do things or print thing to the screen, but do not return any values back to the caller
- **However, we have used several Python functions that return values back to the caller**

input()

print()

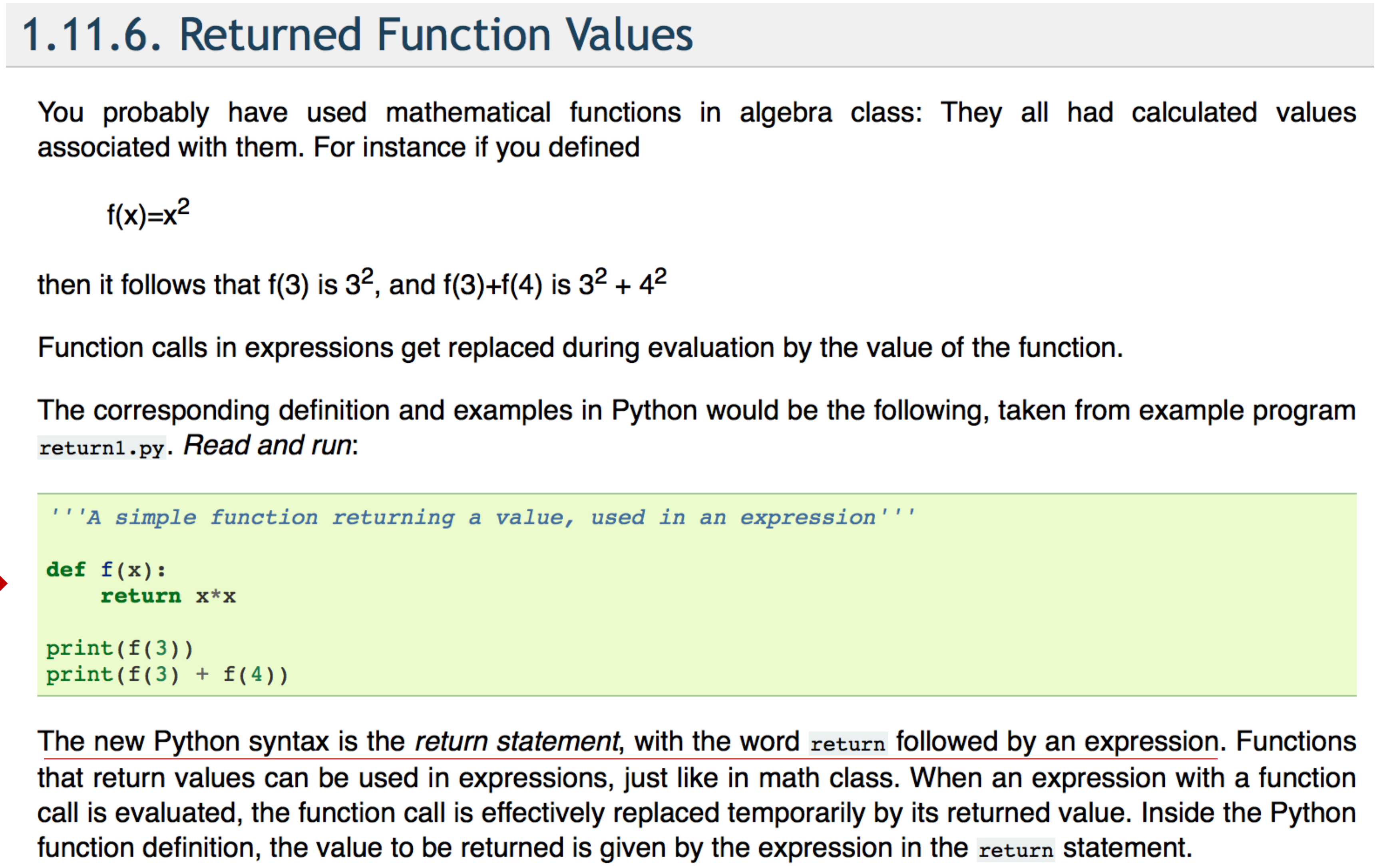
random.randint()

range()

isupper()

Why do we want to have functions that return values?

Function Return Syntax



1.11.6. Returned Function Values

You probably have used mathematical functions in algebra class: They all had calculated values associated with them. For instance if you defined

$$f(x)=x^2$$

then it follows that f(3) is 3^2, and f(3)+f(4) is $3^2 + 4^2$

Function calls in expressions get replaced during evaluation by the value of the function.

The corresponding definition and examples in Python would be the following, taken from example program `return1.py`. *Read and run*:

```python
'''A simple function returning a value, used in an expression'''

def f(x):
    return x*x

print(f(3))
print(f(3) + f(4))
```

The new Python syntax is the *return statement*, with the word `return` followed by an expression. Functions that return values can be used in expressions, just like in math class. When an expression with a function call is evaluated, the function call is effectively replaced temporarily by its returned value. Inside the Python function definition, the value to be returned is given by the expression in the `return` statement.

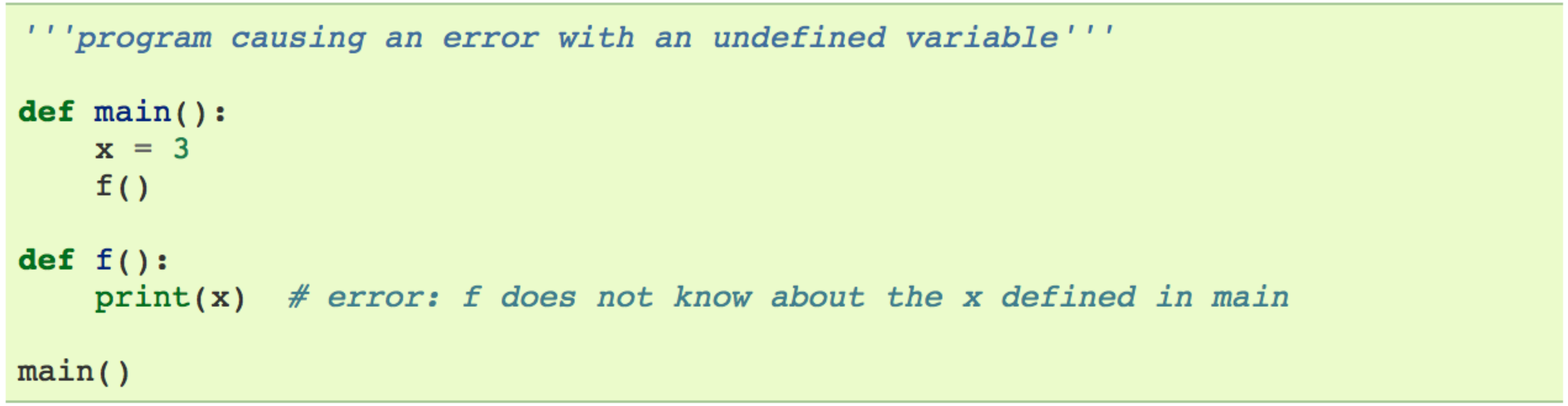
http://anh.cs.luc.edu/handsonPythonTutorial/functions.html

Why Functions that Return Values?

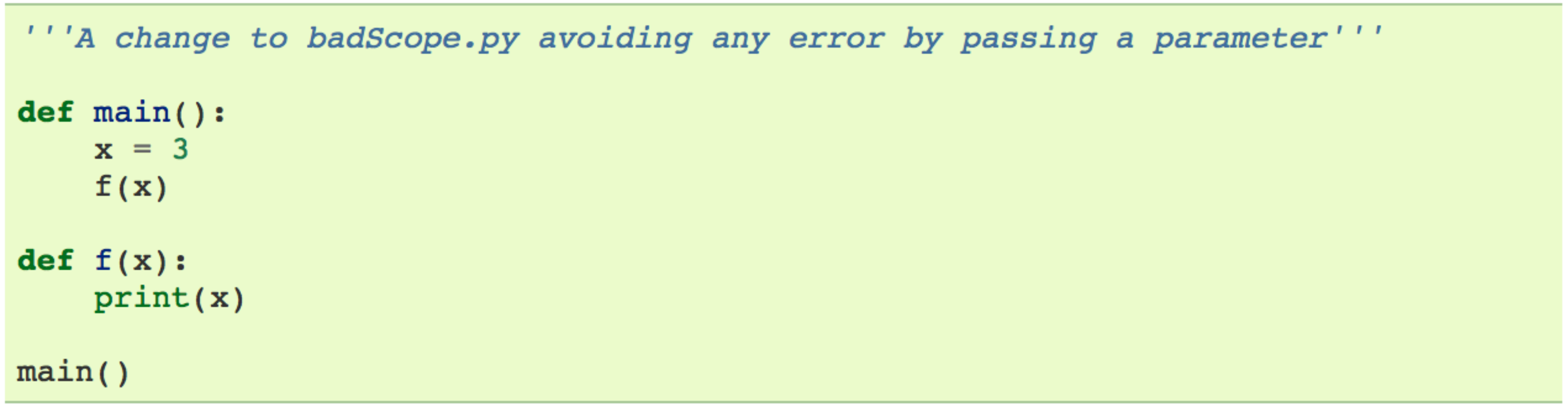
- Allows functions to be stitched (composed) together to make a composite function
  - E.g., $y = f(g(h(x)))$
- Protects data
  - Local scope vs. global scope
  - Changes to data inside the function stay inside the function
    - What happens in Vegas, stays in Vegas

If it is possible for Python built-in functions to change a programmer's variables … *as a programmer, would you use those functions?*

Local Scope vs. Global Scope



```
'''program causing an error with an undefined variable'''

def main():
    x = 3
    f()

def f():
    print(x)  # error: f does not know about the x defined in main

main()
```



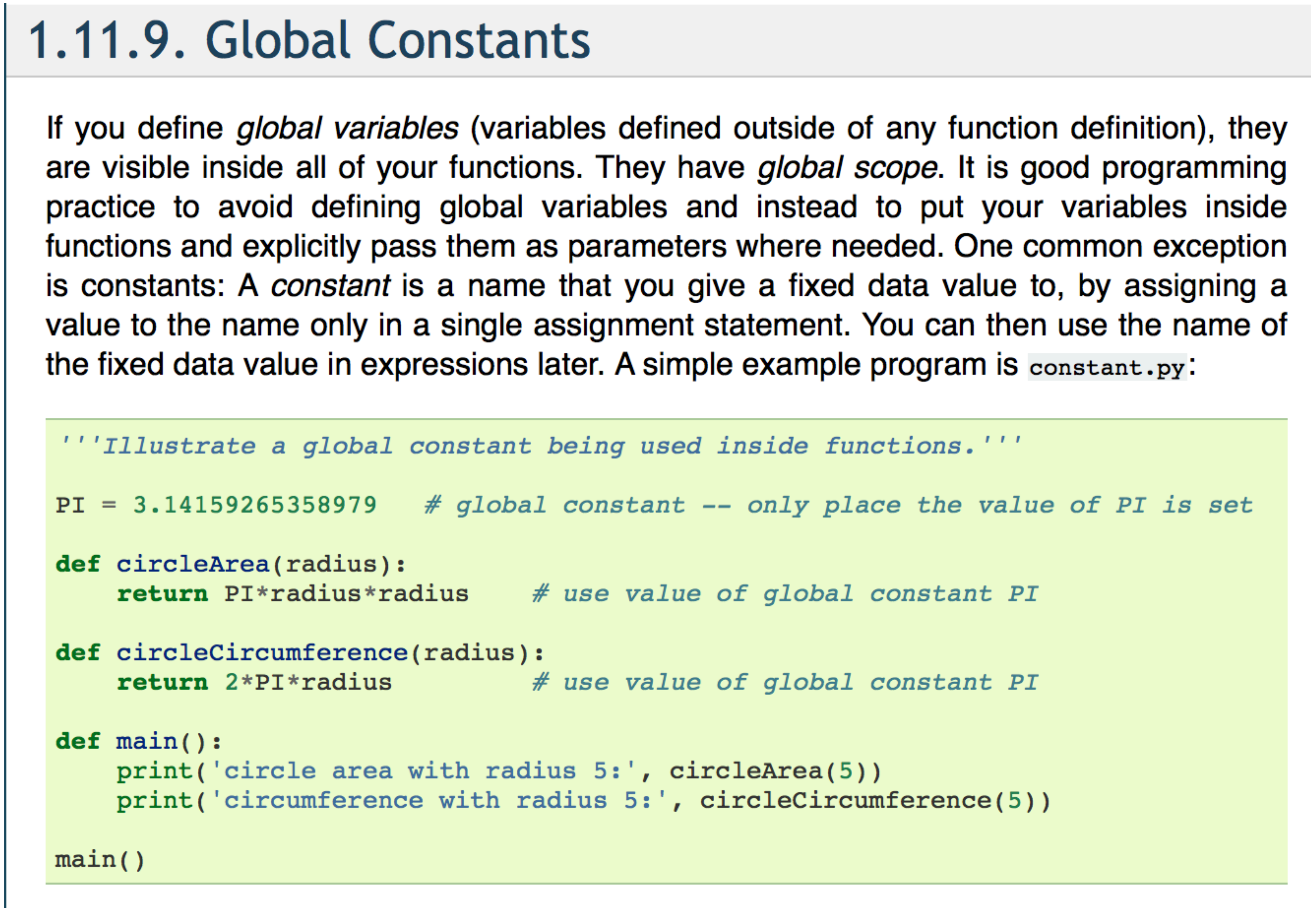
```
'''A change to badScope.py avoiding any error by passing a parameter'''

def main():
    x = 3
    f(x)

def f(x):
    print(x)

main()
```

http://anh.cs.luc.edu/handsonPythonTutorial/files.html

Pass by Value vs. Pass by Reference

- Pass by Value: Most commonly occurring in Python
  - Values are passed, and changes made to a variable stay within the scope of the function
- Pass by Reference: Occurring when “objects” or “structures” are passed …
  - Such as arrays
  - Depending on how the array variable is manipulated in the function
  - See example program passReference.py

Global Constants Syntax



1.11.9. Global Constants

If you define *global variables* (variables defined outside of any function definition), they are visible inside all of your functions. They have *global scope*. It is good programming practice to avoid defining global variables and instead to put your variables inside functions and explicitly pass them as parameters where needed. One common exception is constants: A *constant* is a name that you give a fixed data value to, by assigning a value to the name only in a single assignment statement. You can then use the name of the fixed data value in expressions later. A simple example program is `constant.py`:

```python
'''Illustrate a global constant being used inside functions.'''

PI = 3.14159265358979   # global constant -- only place the value of PI is set

def circleArea(radius):
    return PI*radius*radius     # use value of global constant PI

def circleCircumference(radius):
    return 2*PI*radius          # use value of global constant PI

def main():
    print('circle area with radius 5:', circleArea(5))
    print('circumference with radius 5:', circleCircumference(5))

main()
```

http://anh.cs.luc.edu/handsonPythonTutorial/files.html

Important

- Defining a function does *not* mean calling a function
- Good practice to pass in all the values that a function needs to carry out its actions so that the function does not access global variables
    - Function can use *global constants*
- A function that *prints out* values to the screen is very *different* from a function that *returns* values to the caller
- There are functions that do not return anything
    - Conceptually known as **procedures**
- The returned value of a function *must be stored or assigned to a variable* (Otherwise the returned value is lost)