Introduction to Computation for the Humanities and Social Sciences

CS 3

Chris Tanner

Lecture 7

Can't Spell Functions without Fun.

Lecture 7

- Functions
- Documentation
- Testing Code

Why create functions?

- Helps us **re-use code** instead of writing same parts over and over
- Allows our code to be **modular**
- Increases **readability** (less messy)
- Promotes good problem solving design
- Allows you to reason and think about smaller blocks of code at the same time

Functions

What are they, again?

- Functions define a computation or set of computations, so as to perform a specific task.
- Functions are defined by:
  - their input
  - their computations they contain/execute
  - their output
  - a name to identify the function

What do they look like?

Example: if you wanted to add 2 numbers together, a function could be defined abstractly:

$$\mathbf{f}(a,b) = a + b$$

or in Python, we could write:

```
def addTwoNumbers(a, b):
    return a + b
```

Functions

What do they look like?

```
def your_function_name(input1, input2, …, inputN):
    // code goes here, it's indented, and
    // continues on next line, like always
    return a_variable_you_want
```

Functions

What do they look like?

name it whatever you want, but with good style: use_lower_cased_words

```
def your_function_name(input1, input2, …, inputN):
    // code goes here and
    // continues on next line, like always
    return a_variable_you_want
```

Functions

What do they look like?

name it whatever you want, but with good style: use_lower_cased_words

however many inputs you want, each separated by a comma. these are called **parameters**

```
def your_function_name(input1, input2, …, inputN):
    // code goes here and
    // continues on next line, like always
    return a_variable_you_want
```

Functions

What do they look like?

name it whatever you want, but with good style: use_lower_cased_words

however many inputs you want, each separated by a comma. these are called **parameters**

```
def your_function_name(input1, input2, …, inputN):
    // code goes here and
    // continues on next line, like always
    return a_variable_you_want
```

a **return** statement is mandatory, and must be at the end of your function. it represents your **output**. you can even return several variables if you want, a la: **return (**variable1, variable2, variable3**)**

Functions

What do they look like?

```
# calculates the dri for a person, given their
# weight in kg, height in cm, age, and activity level
def calculate_dri(weight_kg, height_cm, age, act_level):
    bmr = (10*weight_kg + 6.25*height_cm - 5*age + 5)
    dri = bmr * (1.2 + .175*act_level)
    return dri
```

Functions

What do they look like?

```
# calculates the dri and bmr for a person, given their
# weight in kg, height in cm, age, and activity level
def calculate_dri_and_bmr(weight_kg, height_cm, age, act_level):
    bmr = (10*weight_kg + 6.25*height_cm - 5*age + 5)
    dri = bmr * (1.2 + .175*act_level)
    return (dri, bmr)
```

What do they look like?

- All functions start with "**def**" and end that 1st line with a "**:**" colon
- A function should be self-sufficient. Any variables it needs access to should be passed-in as parameters
- In rare situations, it's appropriate for functions to use variables outside of its definition. These accessed variables are called **global variables,** but you should use them sparingly.

Functions

How to create one

- When you write functions, they should conform to a **contract**
- Your code that calls these functions expects them to conform to the contract
- By creating a clear purpose for this function, it's use inside your program remains clear

Contract

Every contract has three parts...

_____ : _____ → _____

function name *domain* *range*

Domain=types of possible input (string, int, Boolean, etc.). Range=types of potential output.

How to create one

- Describe in words what the function does

Purpose Statement

What does the function do?

Functions

How to create one

Examples

Provide some example inputs and outputs that fit your contract and make sense with what you want your function to do...

EXAMPLE: def _____(_____): _____

function name *input(s)* *output*

EXAMPLE: def _____(_____): _____

function name *input(s)* *output*

EXAMPLE: def _____(_____): _____

function name *input(s)* *output*

How to create one

- Create a function for any chunk of code which seems to do a specific task that can be made to be disjoint and only hinges on inputs and outputs.

- Start designing your entire program around this idea of thinking, e.g., “how can I divide my main goal into discrete, separate chunks of computing stuff?”

Functions

How to create one

- After you have a clear idea of what the function is supposed to do and how it will work

- Write your definition, as it would look in Python code, with a return statement of the output variable

Definition

Write the definition, giving variable names to all your input values ...

```
def _____(_____):
    return _____
```

function name *variable name(s)*

variable name for something in the range

This is your gateway to writing actual code

The *main* function

- If all your code is now in functions, where does your program start?
- We use a special function called **main** as our starting point
- Place this code at the very bottom of your code

```
def main():
    # put highest-level logic here
    # which will probably include calling
    # other functions
    # no return statement needed

if __name__ == "__main__":
    main()
```

The following slides are very important and can help save you a LOT of future headache and trouble

Functions

Control Flow

```
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(a)
    print(b)
    print(c)

if __name__ == "__main__":
    main()
```

Control Flow

```
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(a)
    print(b)
    print(c)

if __name__ == "__main__":
    main()
```

meh, don't have permission to execute this function's code :-(

Control Flow

```
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(a)
    print(b)
    print(c)

if __name__ == "__main__":
    main()
```

meh, don't have permission to execute this function's code :-(

Functions

Control Flow

```
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(a)
    print(b)
    print(c)

if __name__ == "__main__":
    main()
```

oh, an entry point into the program! i can execute this block of code!

Functions

Control Flow

```
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(a)
    print(b)
    print(c)

if __name__ == "__main__":
    main()
```

oh, an entry point into the program! i can execute this block of code!

Functions

Control Flow

```
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(a)
    print(b)
    print(c)

if __name__ == "__main__":
    main()
```

yay, permission to execute this function's code now!

Functions

Control Flow

```
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(a)
    print(b)
    print(c)

if __name__ == "__main__":
    main()
```

Functions

Control Flow

```
1   def plus(a, b):
2       a = 2 * a
3       b = 3 + b
4       c = a + b
5       return c
6
7   def main():
8       a = 3
9       b = 5
10      c = plus(a, b)
11      c += 1
12      print(a)
13      print(b)
14      print(c)
15
16  if __name__ == "__main__":
17      main()
```

Functions

Control Flow

```
1   def plus(a, b):
2       a = 2 * a
3       b = 3 + b
4       c = a + b
5       return c
6
7   def main():
8       a = 3
9       b = 5
10      c = plus(a, b)
11      c += 1
12      print(a)
13      print(b)
14      print(c)
15
16  if __name__ == "__main__":
17      main()
```

Functions

Control Flow

```
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(a)
    print(b)
    print(c)

if __name__ == "__main__":
    main()
```

our code calls another function that we made, "plus()"

yay, permission to execute plus()'s code now!

Functions

Control Flow

```
1   def plus(a, b):
2       a = 2 * a
3       b = 3 + b
4       c = a + b
5       return c
6
7   def main():
8       a = 3
9       b = 5
10      c = plus(a, b)
11      c += 1
12      print(a)
13      print(b)
14      print(c)
15
16  if __name__ == "__main__":
17      main()
```

Functions

Control Flow

```
1   def plus(a, b):
2       a = 2 * a
3       b = 3 + b
4       c = a + b
5       return c
6
7   def main():
8       a = 3
9       b = 5
10      c = plus(a, b)
11      c += 1
12      print(a)
13      print(b)
14      print(c)
15
16  if __name__ == "__main__":
17      main()
```

Functions

Control Flow

```
1   def plus(a, b):
2       a = 2 * a
3       b = 3 + b
4       c = a + b
5       return c
6
7   def main():
8       a = 3
9       b = 5
10      c = plus(a, b)
11      c += 1
12      print(a)
13      print(b)
14      print(c)
15
16  if __name__ == "__main__":
17      main()
```

Control Flow

```
1   def plus(a, b):
2       a = 2 * a
3       b = 3 + b
4       c = a + b
5       return c
6
7   def main():
8       a = 3
9       b = 5
10      c = plus(a, b)
11      c += 1
12      print(a)
13      print(b)
14      print(c)
15
16  if __name__ == "__main__":
17      main()
```

now, our plus() function returns its output

Functions

Control Flow

```
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(a)
    print(b)
    print(c)

if __name__ == "__main__":
    main()
```

we receive that output and update our local variable named c

Functions

Control Flow

```
1   def plus(a, b):
2       a = 2 * a
3       b = 3 + b
4       c = a + b
5       return c
6
7   def main():
8       a = 3
9       b = 5
10      c = plus(a, b)
11      c += 1
12      print(a)
13      print(b)
14      print(c)
15
16  if __name__ == "__main__":
17      main()
```

Functions

Control Flow

```
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(a)
    print(b)
    print(c)

if __name__ == "__main__":
    main()
```

Functions

Control Flow

```
1   def plus(a, b):
2       a = 2 * a
3       b = 3 + b
4       c = a + b
5       return c
6
7   def main():
8       a = 3
9       b = 5
10      c = plus(a, b)
11      c += 1
12      print(a)
13      print(b)
14      print(c)
15
16  if __name__ == "__main__":
17      main()
```

Functions

Control Flow

```
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(a)
    print(b)
    print(c)

if __name__ == "__main__":
    main()
```

Variables are localized to a function.

For a given function, we can name its variables whatever we want, and it does not affect variables outside of the function, even if they have the same name.

Variable Scope

- Variables defined within a function only exist temporarily
- **if-statements are similar! Never declare <u>new</u> variables in an if-statement. Updating a variable is fine.**
- Variables defined outside of a function are called globals
- Unless handled specially, global variables are read only (keep them that way!) e.g. `cm_per_inch = 2.54`
- Global variables are defined at the top of a program, not in any function

Variables are localized to a Function

```
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(a)
    print(b)
    print(c)

if __name__ == "__main__":
    main()
```

for this function, we chose to name its passed-in variables **a** and **b**, but we could have named them <u>anything</u>, and as long as we used them the same way, our entire program would yield identical results

Variables are localized to a Function

```
def plus(x, y):
    x = 2 * x
    y = 3 + y
    zzz = x + y
    return zzz

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(a)
    print(b)
    print(c)

if __name__ == "__main__":
    main()
```

for this function, we chose to name its passed-in variables **a** and **b**, but we could have named them <u>anything</u>, and as long as we used them the same way, our entire program would yield identical results

Functions

Variables are localized to a Function

```
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(a)
    print(b)
    print(c)

if __name__ == "__main__":
    main()
```

for this function, we chose to name its passed-in variables **a** and **b**, but we could have named them <u>anything</u>, and as long as we used them the same way, our entire program would yield identical results

Functions

Variables are localized to a Function

```
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(a)
    print(b)
    print(c)

if __name__ == "__main__":
    main()
```

although the exact names don't matter, the ORDER matters. when you call a function, the 1st value gets aligned to the 1st parameter in the function, and so on.

Variables are localized to a Function

```
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(a)
    print(b)
    print(c)

if __name__ == "__main__":
    main()
```

meh, don't have permission to execute this function's code :-(

Variables are localized to a Function

```
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(a)
    print(b)
    print(c)

if __name__ == "__main__":
    main()
```

meh, don't have permission to execute this function's code :-(

Functions

Variables are localized to a Function

```
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(a)
    print(b)
    print(c)

if __name__ == "__main__":
    main()
```

oh, an entry point into the program! i can execute this block of code!

Functions

Variables are localized to a Function

```
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(a)
    print(b)
    print(c)

if __name__ == "__main__":
    main()
```

oh, an entry point into the program! i can execute this block of code!

Functions

Variables are localized to a Function

```
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(a)
    print(b)
    print(c)

if __name__ == "__main__":
    main()
```

yay, permission to execute this function's code now!

Variables are localized to a Function

```
1   def plus(a, b):
2       a = 2 * a
3       b = 3 + b
4       c = a + b
5       return c
6
7   def main():
8       a = 3
9       b = 5
10      c = plus(a, b)
11      c += 1
12      print(a)
13      print(b)
14      print(c)
15
16  if __name__ == "__main__":
17      main()
```

variable	value
a	3

Variables are localized to a Function

```
1   def plus(a, b):
2       a = 2 * a
3       b = 3 + b
4       c = a + b
5       return c
6
7   def main():
8       a = 3
9       b = 5
10      c = plus(a, b)
11      c += 1
12      print(a)
13      print(b)
14      print(c)
15
16  if __name__ == "__main__":
17      main()
```

variable	value
a	3
b	5

Variables are localized to a Function

```
1   def plus(a, b):
2       a = 2 * a
3       b = 3 + b
4       c = a + b
5       return c
6
7   def main():
8       a = 3
9       b = 5
10      c = plus(a, b)
11      c += 1
12      print(a)
13      print(b)
14      print(c)
15
16  if __name__ == "__main__":
17      main()
```

variable	value
a	3
b	5

Variables are localized to a Function

```
1   def plus(a, b):
2       a = 2 * a
3       b = 3 + b
4       c = a + b
5       return c
6
7   def main():
8       a = 3
9       b = 5
10      c = plus(a, b)
11      c += 1
12      print(a)
13      print(b)
14      print(c)
15
16  if __name__ == "__main__":
17      main()
```

variable	value
a	3
b	5
a	3
b	5

Functions

Variables are localized to a Function

```
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(a)
    print(b)
    print(c)

if __name__ == "__main__":
    main()
```

variable	value
a	3
b	5
a	6
b	5

Functions

Variables are localized to a Function

```
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(a)
    print(b)
    print(c)

if __name__ == "__main__":
    main()
```

variable	value
a	3
b	5
a	6
b	8

Variables are localized to a Function

```
1   def plus(a, b):
2       a = 2 * a
3       b = 3 + b
4       c = a + b
5       return c
6
7   def main():
8       a = 3
9       b = 5
10      c = plus(a, b)
11      c += 1
12      print(a)
13      print(b)
14      print(c)
15
16  if __name__ == "__main__":
17      main()
```

variable	value
a	3
b	5
a	6
b	8
c	14

Variables are localized to a Function

```
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(a)
    print(b)
    print(c)

if __name__ == "__main__":
    main()
```

when we return/ finish running plus(), all of its variables are deleted, and we only carry on its output

Variables are localized to a Function

```
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(a)
    print(b)
    print(c)

if __name__ == "__main__":
    main()
```

variable	value
a	3
b	5
c	14

Variables are localized to a Function

```
1   def plus(a, b):
2       a = 2 * a
3       b = 3 + b
4       c = a + b
5       return c
6
7   def main():
8       a = 3
9       b = 5
10      c = plus(a, b)
11      c += 1
12      print(a)
13      print(b)
14      print(c)
15
16  if __name__ == "__main__":
17      main()
```

variable	value
a	3
b	5
c	15

Variables are localized to a Function

```
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(a)
    print(b)
    print(c)

if __name__ == "__main__":
    main()
```

variable	value
a	3
b	5
c	15

Variables are localized to a Function

```
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(a)
    print(b)
    print(c)

if __name__ == "__main__":
    main()
```

variable	value
a	3
b	5
c	15

Variables are localized to a Function

```
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(a)
    print(b)
    print(c)

if __name__ == "__main__":
    main()
```

variable	value
a	3
b	5
c	15

Lecture 7

- Functions
- Documentation
- Testing Code

Docstrings

- Functions require their own special documentation called **docstrings**
- Docstrings communicate the information you wrote in your worksheet
- The specific format is conducive for others to use your function

Docstrings

```
def function_name(param1, param2,…):
    """ The purpose statement of the function
        goes on the first line

    Keyword arguments:
    param1 -- Type and expected values of input_parameter1
    param2 -- Type and expected values of input_parameter2

    Returns:
    output_variable The type and its expected values
    """

    # Function code goes here
    return output_variable
```

Documentation

Docstrings

```
def function_name(a, b):
    """ if the function is short, 1 line desc. is fine """
    return a*2 + b*b
```

Lecture 7

- Functions
- Documentation
- Testing Code

Testing

- Designing a good program requires testing the program for correctness
- To test a program, you generate sample input and output pairs called **test cases**
- The examples you created before designing your program can serve as a starting point for test cases
- Extreme input examples that stress your program are called **corner cases.** Include many corner cases

Testing

- Testing a program that produces output based on a randomized input is difficult because the result always changes
- To test our randomized program, you can temporarily hard code the computer's choice
- **Start small!** If your program requires reading a text file, don't give it the entire file at first. Test it with just 1 or 2 lines of text. Check intermediate values of the variables, etc.

Conclusions

- Use functions to modularize your code
- This allows you to reason and think about smaller blocks of code at the same time
- Document your functions to communicate what they are doing
- Don’t forget the colons!

Lab Time