

Introduction to Computation for the Humanities and Social Sciences



CS 3

Chris Tanner

Lecture 8



Lecture 8

- Functions — Recap
- Indentation
- Data Structures!
 - Lists
- Looping
 - For Loops
 - While Loops

Functions — Recap

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 — Recap

```
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 — Recap

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

```
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 — Recap

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, it's indented, and  
    # continues on next line, like always  
    return a_variable_you_want
```

Functions — Recap

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, it's indented, 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 — Recap

Can also return multiple variables,
just separate them with commas

```
# 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)
```

Functions — Recap

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 — Recap

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

Functions — Recap

```
1  def plus(a, b):  
2      return a + b  
3  
4  def main():  
5      a = 3  
6      b = 5  
7      c = plus(a, b)  
8      print("answer is: " + str(c))
```

```
9  
10 if __name__ == "__main__":  
11     main()  
12
```

these two lines must be at the very
bottom of all of your programs!

Functions — Recap

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


Functions — Recap

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

crashes the program
because it doesn't know
what `subtract()` function is,
as it appears below the 'if
`__name__`' line

Functions

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()
```

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

Functions

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()
```

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

Functions

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()
```

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

Functions

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()
```

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

Functions

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()
```

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

Functions

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
```

variable	value
a	3



```
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

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



Functions

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)
```

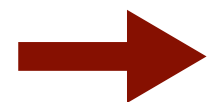
```
14     print(c)
```

```
15
```

```
16 if __name__ == "__main__":
```

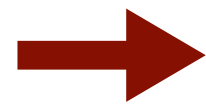
```
17     main()
```

variable	value
a	3
b	5



Functions

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
---	---

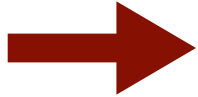
b	5
---	---

a	3
---	---

b	5
---	---

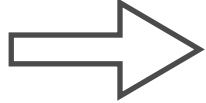
Functions

Variables are localized to a Function



```
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	5



Functions

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
----------	---

Functions

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
----------	----

Functions

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()
```

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

Functions

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)
```

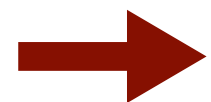
```
14     print(c)
```

```
15
```

```
16 if __name__ == "__main__":
```

```
17     main()
```

variable	value
a	3
b	5
c	14



Functions

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



Functions

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



Functions

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)
```

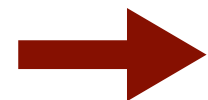
```
14     print(c)
```

```
15
```

```
16 if __name__ == "__main__":
```

```
17     main()
```

variable	value
a	3
b	5
c	15



Functions

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



Functions — Recap

Reasons to use functions

- Functions allow you to re-use code that you may use many times
- Functions help modularize your code, so you can reason about a smaller part of your code at any one time
- Putting all your code into functions helps reduce bugs
 - Variable names used in one function won't conflict with variables named the same thing in another function
 - You can test each function individually to check if it works correctly
 - Critically analyzing a concrete block of code in a function is easier than a long block of code

REAL-TIME CODING

Functions — Recap

Reasons to use functions

- Functions allow you to re-use code that you may use many times
- Functions help modularize your code, so you can reason about a smaller part of your code at any one time
- Putting all your code into functions helps reduce bugs
 - Variable names used in one function won't conflict with variables in another function (even if they have the same name)
 - You can test each function individually to check if it works correctly
 - Critically analyzing a concrete block of code in a function is easier than a long block of code

Lecture 8

- Functions — Recap
- Indentation
- Data Structures!
 - Lists
- Looping
 - For Loops
 - While Loops

Indentations

When to indent code

- Indent code whenever a block of code pertains to the control sequence the resides above — a line ending in a colon
- e.g.,
 - looping constructs (**for** loops, **while** loops)
 - a function (**def** **function_name**)
 - **if**-statements
- Items at the same level of indentation get executed sequentially
- To execute code that is indented one more level from the current indentation, the code needs “permission” to enter such
- When it’s done executing the “inner” indented code, execution resumes at the next-most indented level


Indentations

When to indent code

```
1  def main():
2      a = 1
3      if a < 10:
4          print("value of a: " + str(a))
5          a *= 2
6      b = 5
7      c = 8
8
9  if __name__ == "__main__":
10     main()
```

Indentations

When to indent code

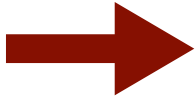


```
1  def main():
2      a = 1
3      if a < 10:
4          print("value of a: " + str(a))
5          a *= 2
6      b = 5
7      c = 8
8
9  if __name__ == "__main__":
10     main()
```

Indentations

When to indent code

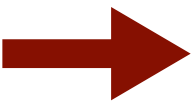
```
1  def main():
2      a = 1
3      if a < 10:
4          print("value of a: " + str(a))
5          a *= 2
6      b = 5
7      c = 8
8
9  if __name__ == "__main__":
10     main()
```



Indentations


When to indent code

```
1  def main():
2      a = 1
3      if a < 10:
4          print("value of a: " + str(a))
5          a *= 2
6      b = 5
7      c = 8
8
9  if __name__ == "__main__":
10     main()
```



Indentations


When to indent code



```
1  def main():
2      a = 1
3      if a < 10:
4          print("value of a: " + str(a))
5          a *= 2
6      b = 5
7      c = 8
8
9  if __name__ == "__main__":
10     main()
```

Indentations


When to indent code



```
1  def main():
2      a = 1
3      if a < 10:
4          print("value of a: " + str(a))
5          a *= 2
6      b = 5
7      c = 8
8
9  if __name__ == "__main__":
10     main()
```

Indentations


When to indent code



```
1  def main():
2      a = 1
3      if a < 10:
4          print("value of a: " + str(a))
5          a *= 2
6      b = 5
7      c = 8
8
9  if __name__ == "__main__":
10     main()
```

Indentations

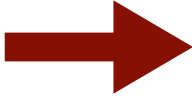
When to indent code



```
1  def main():
2      a = 1
3      if a < 10:
4          print("value of a: " + str(a))
5          a *= 2
6      b = 5
7      c = 8
8
9  if __name__ == "__main__":
10     main()
```

Indentations


When to indent code



```
1  def main():
2      a = 1
3      if a < 10:
4          print("value of a: " + str(a))
5          a *= 2
6      b = 5
7      c = 8
8
9  if __name__ == "__main__":
10     main()
```

Indentations

When to indent code




```
1  def main():
2      a = 1
3      if a < 10:
4          print("value of a: " + str(a))
5          a *= 2
6      b = 5
7      c = 8
8
9  if __name__ == "__main__":
10     main()
```

Indentations

When to indent code

```
1  def main():
2      a = 1
3      if a < 10:
4          print("value of a: " + str(a))
5          a *= 2
6      b = 5
7      c = 8
8
9  if __name__ == "__main__":
10     main()
```



Lecture 8

- Functions — Recap
- Indentation
- Data Structures!
 - Lists
- Looping
 - For Loops
 - While Loops

Data Types vs Data Structures

- So far, we've talked about different *types* of data:
 - **ints, floats, bools, and strings**
- But we've only looked at storing **1 value** for each variable.

<u>variable</u>	value
age	19
name	"sarah johnson"
student_id	217029748

Data Types vs Data Structures

- What if we need to store multiple items?
e.g., names of students in a class, or zip codes in California?

students



ca_zip_codes



REAL-TIME CODING

Data Types vs Data Structures

- There are *many* **structures** of data
- **Lists** and **single-valued** are the most common structures
- Each is useful for its own reasons
- Any structure can contain data of any **type**, but it makes most sense for the type of data to remain **consistent** (homogenous) for a given variable
- e.g., a list of zip codes should contain only numbers
- e.g., a list of student names should contain only strings

Lists

```
my_list = ['a', 'b', 'c']
```

- add items to a list:
 - **my_list.append(x)** tacks item **x** on to the end
 - **my_list.insert(i, x)** adds item **x** into the list at index **i**
- **remove(x)** removes item **x**
- **find(x)** returns the index at which item **x** was found. returns **-1** if it's not in the list.
- **my_list.sort()** sorts the list
- **my_list.reverse()** reverses the list
- **len(my_list)** returns the # of items in the list (i.e., the length)

REMEMBER: indices start at 0, not 1

Lecture 8

- Functions — Recap
- Indentation
- Data Structures!
 - Lists
- Looping
 - For Loops
 - While Loops

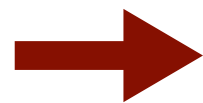
Looping

- **Definition:** A **loop** is a construct of programming languages which provides the ability to repeat an operation a particular number of times
- It's one of the core functionalities of programming languages
- You can either perform a loop **for** a pre-specified number of times (e.g., a **for-loop** which goes through each item in a list) or
- perform a loop indefinitely, **while** a condition is met (**while loops**)

Looping

For Loops

Iterate through each item in a list




```
1  fruits = ['apple', 'litchi', 'rambutan', 'banana']  
2  
3  for fruit in fruits:  
4      print("current fruit: " + fruit)  
5  
6  print("total # of fruits: " + str(len(fruits)))
```

TERMINAL OUTPUT:

Looping

For Loops

Iterate through each item in a list




```
1  fruits = ['apple', 'litchi', 'rambutan', 'banana']
2
3  for fruit in fruits:
4      print("current fruit: " + fruit)
5
6  print("total # of fruits: " + str(len(fruits)))
```

TERMINAL OUTPUT:

Looping

For Loops

Iterate through each item in a list




```
1  fruits = ['apple', 'litchi', 'rambutan', 'banana']  
2  
3  for fruit in fruits:  
4      print("current fruit: " + fruit)  
5  
6  print("total # of fruits: " + str(len(fruits)))
```

TERMINAL OUTPUT:

Looping

For Loops

Iterate through each item in a list



```
1  fruits = ['apple', 'litchi', 'rambutan', 'banana']
2
3  for fruit in fruits:
4      print("current fruit: " + fruit)
5
6  print("total # of fruits: " + str(len(fruits)))
```


TERMINAL OUTPUT:

```
current fruit: apple
```

Looping

For Loops

Iterate through each item in a list



```
1  fruits = ['apple', 'litchi', 'rambutan', 'banana']
2
3  for fruit in fruits:
4      print("current fruit: " + fruit)
5
6  print("total # of fruits: " + str(len(fruits)))
```


TERMINAL OUTPUT:

```
current fruit: apple
```

Looping

For Loops

Iterate through each item in a list



```
1  fruits = ['apple', 'litchi', 'rambutan', 'banana']
2
3  for fruit in fruits:
4      print("current fruit: " + fruit)
5
6  print("total # of fruits: " + str(len(fruits)))
```


TERMINAL OUTPUT:

```
current fruit: apple
current fruit: litchi
```

Looping

For Loops

Iterate through each item in a list



```
1  fruits = ['apple', 'litchi', 'rambutan', 'banana']
2
3  for fruit in fruits:
4      print("current fruit: " + fruit)
5
6  print("total # of fruits: " + str(len(fruits)))
```


TERMINAL OUTPUT:

```
current fruit: apple
current fruit: litchi
```

Looping

For Loops

Iterate through each item in a list



```
1  fruits = ['apple', 'litchi', 'rambutan', 'banana']
2
3  for fruit in fruits:
4      print("current fruit: " + fruit)
5
6  print("total # of fruits: " + str(len(fruits)))
```

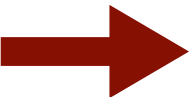
TERMINAL OUTPUT:

```
current fruit: apple
current fruit: litchi
current fruit: rambutan
```


Looping

For Loops

Iterate through each item in a list



```
1  fruits = ['apple', 'litchi', 'rambutan', 'banana']
2
3  for fruit in fruits:
4      print("current fruit: " + fruit)
5
6  print("total # of fruits: " + str(len(fruits)))
```


TERMINAL OUTPUT:

```
current fruit: apple
current fruit: litchi
current fruit: rambutan
```

Looping

For Loops

Iterate through each item in a list



```
1  fruits = ['apple', 'litchi', 'rambutan', 'banana']
2
3  for fruit in fruits:
4      print("current fruit: " + fruit)
5
6  print("total # of fruits: " + str(len(fruits)))
```

TERMINAL OUTPUT:

```
current fruit: apple
current fruit: litchi
current fruit: rambutan
current fruit: banana
```

Looping

For Loops

Iterate through each item in a list

```
1  fruits = ['apple', 'litchi', 'rambutan', 'banana']
2
3  for fruit in fruits:
4      print("current fruit: " + fruit)
5
6  print("total # of fruits: " + str(len(fruits)))
```

TERMINAL OUTPUT:

```
current fruit: apple
current fruit: litchi
current fruit: rambutan
current fruit: banana
total # of fruits: 4
```

Looping

For Loops

range(x) function returns a list of items from 0 to x-1

```
1 for i in range(50):  
2     print("i: " + str(i))  
3     if i % 3 == 0:  
4         print(str(i) + " is divisible by 3!")
```

While Loops

Perform a chunk of a code indefinitely,
while the loop condition is True

```
1  i = 0
2  while i < 10:
3      print("value of i: " + str(i))
4      i += 1
5
6  print("value of i: " + str(i))
```

Lab Time

