Introduction to Computation for the Humanities and Social Sciences

CS 3
Chris Tanner
Lecture 8

- Functions — Recap
- Indentation
- Data Structures!
  - Lists
- Looping
  - For Loops
  - While Loops
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.
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
name it whatever you want, but with good style: use_lower_cased_words

```python
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

```python
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
```

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
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
```

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

if you want to output a value, 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)
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

```python
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)
```
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?”
def plus(a, b):
    return a + b

def main():
    a = 3
    b = 5
    c = plus(a, b)
    print("answer is: " + str(c))

if __name__ == "__main__":
    main()
Functions — Recap

```python
1  def plus(a, b):
2      return a + b

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

9  if __name__ == "__main__":
10     main()

these two lines must be at the very bottom of all of your programs!
```
def plus(a, b):
    return a + b

def main():
    a = 3
    b = 5
    c = plus(a, b)
    print("answer is: " + str(c))
    d = subtract(c, a)

if __name__ == "__main__":
    main()

def subtract(a, b):
    return a - b
Functions — Recap

```python

def plus(a, b):
    return a + b

def main():
    a = 3
    b = 5
    c = plus(a, b)
    print("answer is: " + str(c))
    d = subtract(c, a)

if __name__ == "__main__":
    main()

def subtract(a, b):
    return a - b
```

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

```python
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
Variables are localized to a Function

```python
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

```python
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

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

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

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

Variables are localized to a Function

```python
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!
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

Variables are localized to a Function

```python
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()
```

<table>
<thead>
<tr>
<th>variable</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>5</td>
</tr>
</tbody>
</table>

Functions

Variables are localized to a Function

```python
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()
```

<table>
<thead>
<tr>
<th>variable</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>5</td>
</tr>
</tbody>
</table>
Functions

Variables are localized to a Function

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

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

<table>
<thead>
<tr>
<th>variable</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>5</td>
</tr>
<tr>
<td>a</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>5</td>
</tr>
</tbody>
</table>

Variable values: a = 3, b = 5
Functions

Variables are localized to a Function

```python
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()
```

<table>
<thead>
<tr>
<th>variable</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>5</td>
</tr>
<tr>
<td>a</td>
<td>6</td>
</tr>
<tr>
<td>b</td>
<td>5</td>
</tr>
</tbody>
</table>

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
Functions

Variables are localized to a Function

```python
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

Variables are localized to a Function

```python
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()
```

<table>
<thead>
<tr>
<th>variable</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>5</td>
</tr>
<tr>
<td>a</td>
<td>6</td>
</tr>
<tr>
<td>b</td>
<td>8</td>
</tr>
<tr>
<td>c</td>
<td>14</td>
</tr>
</tbody>
</table>

Variables are localized to a Function
Functions

Variables are localized to a Function

```python
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

Variables are localized to a Function
Functions

Variables are localized to a Function

```python
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

Variables are localized to a Function

```python
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()
```

<table>
<thead>
<tr>
<th>variable</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>5</td>
</tr>
<tr>
<td>c</td>
<td>15</td>
</tr>
</tbody>
</table>
Functions

Variables are localized to a Function

```python
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()
```

<table>
<thead>
<tr>
<th>variable</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>5</td>
</tr>
<tr>
<td>c</td>
<td>15</td>
</tr>
</tbody>
</table>

Variables are localized to a Function
Functions

Variables are localized to a Function

```python
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()
```

<table>
<thead>
<tr>
<th>variable</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>5</td>
</tr>
<tr>
<td>c</td>
<td>15</td>
</tr>
</tbody>
</table>
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()
```
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
When to indent code

- Indent code whenever a block of code pertains to the control sequence it 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
When to indent code

```python
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  if __name__ == "__main__":
9      main()
10```
When to indent code

```python
def main():
    a = 1
    if a < 10:
        print("value of a: " + str(a))
        a *= 2
    b = 5
    c = 8

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

if __name__ == "__main__":
    main()
When to indent code

```python
def main():
a = 1
if a < 10:
    print("value of a: " + str(a))
a *= 2
b = 5
c = 8

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

if __name__ == "__main__":
    main()
When to indent code

```python
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()
```
When to indent code

```python
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 if __name__ == "__main__":
9      main()
```
When to indent code

```python
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()
```
When to indent code

```python
def main():
    a = 1
    if a < 10:
        print("value of a: " + str(a))
        a *= 2
    b = 5
    c = 8

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

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

<table>
<thead>
<tr>
<th>variable</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>age</td>
<td>19</td>
</tr>
<tr>
<td>name</td>
<td>“sarah johnson”</td>
</tr>
<tr>
<td>student_id</td>
<td>217029748</td>
</tr>
</tbody>
</table>
Data Structures

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

“lydia” “claire” “sarah” “christina” ...

ca_zip_codes

90027 90091 90180 90210 ...
REAL-TIME CODING
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
List Properties

• Lists are **ordered** collections of items
  
  • doesn’t imply it’s always sorted, but that there’s a concept of which item is before or after another item in the list.

• You can store duplicate values if you want:

  • `students = [“John”, “Mary”, “John”, “Stephanie”]`
Data Structures

List Functions

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`
- index(x) returns the index at which value `x` was found. Returns -1 if it’s not in the list. (it “finds” the value)
- `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)
Iterate through each item in a list

```python
fruits = ['apple', 'litchi', 'rambutan', 'banana']
for fruit in fruits:
    print("current fruit: "+ fruit)
print("total # of fruits: "+ str(len(fruits)))
```
For Loops

Iterate through each item in a list

```python
fruits = ['apple', 'litchi', 'rambutan', 'banana']

for fruit in fruits:
    print("current fruit: " + fruit)

print("total # of fruits: " + str(len(fruits)))
```

TERMINAL OUTPUT:
For Loops

Iterate through each item in a list

```python
fruits = ['apple', 'litchi', 'rambutan', 'banana']

for fruit in fruits:
    print("current fruit: ", fruit)

print("total # of fruits: ", str(len(fruits)))
```

TERMINAL OUTPUT:
Iterate through each item in a list

```python
fruits = ['apple', 'litchi', 'rambutan', 'banana']
for fruit in fruits:
    print("current fruit: " + fruit)
print("total # of fruits: " + str(len(fruits)))
```

**TERMINAL OUTPUT:**

current fruit: apple
Looping

For Loops

Iterate through each item in a list

```python
fruits = ['apple', 'litchi', 'rambutan', 'banana']

for fruit in fruits:
    print("current fruit: " + fruit)

print("total # of fruits: " + str(len(fruits)))
```

TERMINAL OUTPUT:

current fruit: apple
For Loops

Iterate through each item in a list

```python
fruits = ['apple', 'litchi', 'rambutan', 'banana']
for fruit in fruits:
    print("current fruit: "+fruit)
print("total # of fruits: " + str(len(fruits)))
```

TERMINAL OUTPUT:

current fruit: apple
current fruit: litchi
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
For Loops

Iterate through each item in a list

```python
fruits = ['apple', 'litchi', 'rambutan', 'banana']

for fruit in fruits:
    print("current fruit: " + fruit)

print("total # of fruits: " + str(len(fruits)))
```

**TERMINAL OUTPUT:**

```
current fruit: apple
current fruit: litchi
current fruit: rambutan
```
For Loops

Iterate through each item in a list

```python
fruits = ['apple', 'litchi', 'rambutan', 'banana']

for fruit in fruits:
    print("current fruit: ", fruit)

print("total # of fruits: ", str(len(fruits)))
```

Terminal Output:

current fruit: apple
current fruit: litchi
current fruit: rambutan
For Loops

Iterate through each item in a list

```python
fruits = ['apple', 'litchi', 'rambutan', 'banana']

for fruit in fruits:
    print("current fruit: " + fruit)

print("total # of fruits: " + str(len(fruits)))
```

TERMINAL OUTPUT:

current fruit: apple
current fruit: litchi
current fruit: rambutan
current fruit: banana
For Loops

Iterate through each item in a list

```python
fruits = ['apple', 'litchi', 'rambutan', 'banana']

for fruit in fruits:
    print("current fruit: " + fruit)

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
range(x) function allows you to loop through numbers from 0 to x-1. Look up the function for other options.

```
for i in range(50):
    print("i: " + str(i))
    if i % 3 == 0:
        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