

Tutorial 2 – Week 3

We'll be starting at the 10 minute mark

if nothing else, write #cleancode

Introduction – TA



Ali Howidi

(TUT0109)

Current studies: Second year MASc BME student

Research/other interests: Signal analysis and machine learning for biosignals

Agenda

1. Lab 1 Review
 - a) Review of errors
2. Lecture Review
 - a) Defining and using functions
 - b) Functions: Tips & Misconceptions
 - c) Built-in functions
 - d) Importing and Using Modules
3. Practice Questions
4. Questions?

Upon completing this tutorial, the students should be able to:

1. Understand the general concept of "function"
2. Recognize Python function calls
3. Recognize Python function definitions
4. Know how to define a function (header + body)
5. Understand the use of Python *return* statement
6. Know how to return a value from a function using a return statement
7. Understand the general concept of "variable scope"
8. Understand the general concept of built-in "function"
9. Know how to import modules
10. Know how to access a function defined in a module they imported

Learning Objectives

Review of Lab 1

Errors

if nothing else, write `#cleancode`

Lab 1: Review of Errors¹

Syntax Errors: occur when your code breaks the syntactic rules of the language (e.g., your code contains unbalanced brackets, identifiers, such as variable or function names, that do not obey the rules)

- Example: **alex.circle(50 180)**
 - Corrected version: **alex.circle(50, 180)**
 - Explanation: a syntax rule was broken, more specifically the **RULE** that function arguments must be separated by commas

Command	Action
alex.up()	Lift the tail (stop drawing)
alex.down()	Lower the tail (start drawing)
alex.right(<i>d</i>)	Turn right by <i>d</i> degrees
alex.left(<i>d</i>)	Turn left by <i>d</i> degrees
alex.forward(<i>s</i>)	Move <i>s</i> steps in the current direction
alex.backward(<i>s</i>)	Move <i>s</i> steps backwards with the current heading
alex.setheading(<i>d</i>)	Change heading to direction <i>d</i> (0: east, 90: north, 180: west, 270: south)
alex.goto(<i>x, y</i>)	Move to coordinates (<i>x,y</i>)
alex.circle(<i>r, d</i>)	Move in circle with <i>r</i> radius for <i>d</i> degrees (counterclockwise if <i>d</i> > 0)

Added comma missing between arguments



Lab 1: Review of Errors²

Semantic Errors: occur when the semantic rules of the language are broken (e.g., calling a function with the wrong argument type or with the wrong number of arguments)

Command	Action
alex.up()	Lift the tail (stop drawing)
alex.down()	Lower the tail (start drawing)
alex.right(<i>d</i>)	Turn right by <i>d</i> degrees
alex.left(<i>d</i>)	Turn left by <i>d</i> degrees
alex.forward(<i>s</i>)	Move <i>s</i> steps in the current direction
alex.backward(<i>s</i>)	Move <i>s</i> steps backwards with the current heading
alex.setheading(<i>d</i>)	Change heading to direction <i>d</i> (0: east, 90: north, 180: west, 270: south)
alex.goto(<i>x</i> , <i>y</i>)	Move to coordinates (<i>x</i> , <i>y</i>)
alex.circle(<i>r</i> , <i>d</i>)	Move in circle with <i>r</i> radius for <i>d</i> degrees (counterclockwise if <i>d</i> > 0)

- Example: `alex.up(5)`
 - Corrected version: `alex.up()`
 - Explanation: a semantic rule was broken, i.e., method `up()` was called with the wrong number of arguments, more specifically `up()` accepts fewer arguments than were passed to it.

Removed argument



Lab 1: Review of Errors³

Logical Errors: occur when the meaning/semantics of a program is not the intended one (e.g., a mathematical formula is implemented incorrectly, a block of instructions is mis-indented)

- **Example:** `alex.right(2 * math.pi)`
 - **Corrected version:** `alex.right(360)`
 - **Explanation:** a higher level semantic rule was broken, i.e., function `right()` assumes that the values passed to it represent angles expressed in **degrees** (not **radians !**) and was passed a value that represents the value of an angle expressed in radians.

Command	Action
<code>alex.up()</code>	Lift the tail (stop drawing)
<code>alex.down()</code>	Lower the tail (start drawing)
<code>alex.right(d)</code>	Turn right by d degrees
<code>alex.left(d)</code>	Turn left by d degrees
<code>alex.forward(s)</code>	Move s steps in the current direction
<code>alex.backward(s)</code>	Move s steps backwards with the current heading
<code>alex.setheading(d)</code>	Change heading to direction d (0: east, 90: north, 180: west, 270: south)
<code>alex.goto(x, y)</code>	Move to coordinates (x, y)
<code>alex.circle(r, d)</code>	Move in circle with r radius for d degrees (counterclockwise if $d > 0$)

Corrected the value
of the argument

Review of Lecture

Defining and Using functions

if nothing else, write `#cleancode`

Defining Functions

Function Elements:

1. Header
2. Body
 - Docstring
 - Statements implementing the purpose (intended semantics) of the function

```
week3.py|  
function_name ↴  
#function syntax  
  
def function_name (parameters): #function heading  
    #docstring  
    '''  
        Type contract (ie all input parameter types -> return type)  
        Describe your function here  
    '''  
  
    # function body goes here  
  
    # return statement at the end to exit the function
```

Practice question:

I implement a function that calculates the area of a rectangle.

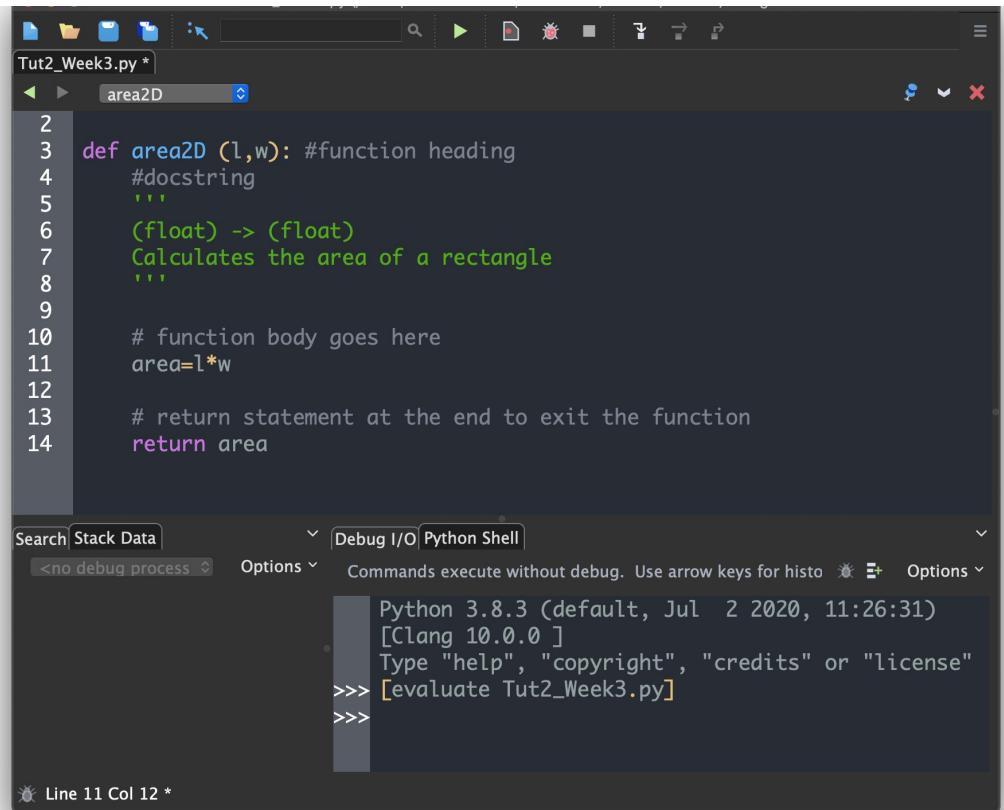
Example of a function

Function Elements:

1. Header
2. Body
 - Docstring
 - Statements implementing the purpose (intended semantics) of the function

Questions:

- What happens when I run this code?
- How do/can I use this function ?



The screenshot shows a Python development environment with two main panes. The top pane is a code editor titled "Tut2_Week3.py" containing the following code:

```
2
3 def area2D (l,w): #function heading
4     #docstring
5     """
6         (float) -> (float)
7         Calculates the area of a rectangle
8         """
9
10    # function body goes here
11    area=l*w
12
13    # return statement at the end to exit the function
14    return area
```

The bottom pane is a Python Shell window titled "Python Shell". It displays the following session:

```
Search Stack Data Debug I/O Python Shell
<no debug process <> Options Commands execute without debug. Use arrow keys for histo ⌘ E+ Options
Python 3.8.3 (default, Jul 2 2020, 11:26:31)
[Clang 10.0.0 ]
Type "help", "copyright", "credits" or "license"
>>> [evaluate Tut2_Week3.py]
>>>
```

At the bottom of the shell window, it says "Line 11 Col 12 *".

Calling functions

Note: variables length and width are **parameters** of the area2D function.

Questions:

- Q1: Why isn't the area of the rectangle with length 5 and width 7 displayed in the Python shell?
- Q2. How can I display the value returned by the function area2D ?

```
▶ (bottom) ▾  
#function syntax  
def area2D(l,w): #function heading  
    #docstring  
    '''  
    (float), (float) -> (float)  
    Calculates the area of a rectangle  
    '''  
  
    #function body goes here  
    area=l*w  
  
    #return statement at the end to exit the function  
    return area  
  
length=5  
width=7  
area2D(length,width) #calling your funciton
```

Functions and variables

- **Q1 Answer:** the value returned by the function `area2D` is "stored" in variable `rectangle`, but the program does not contain an instruction for displaying the value of variable `rectangle`.

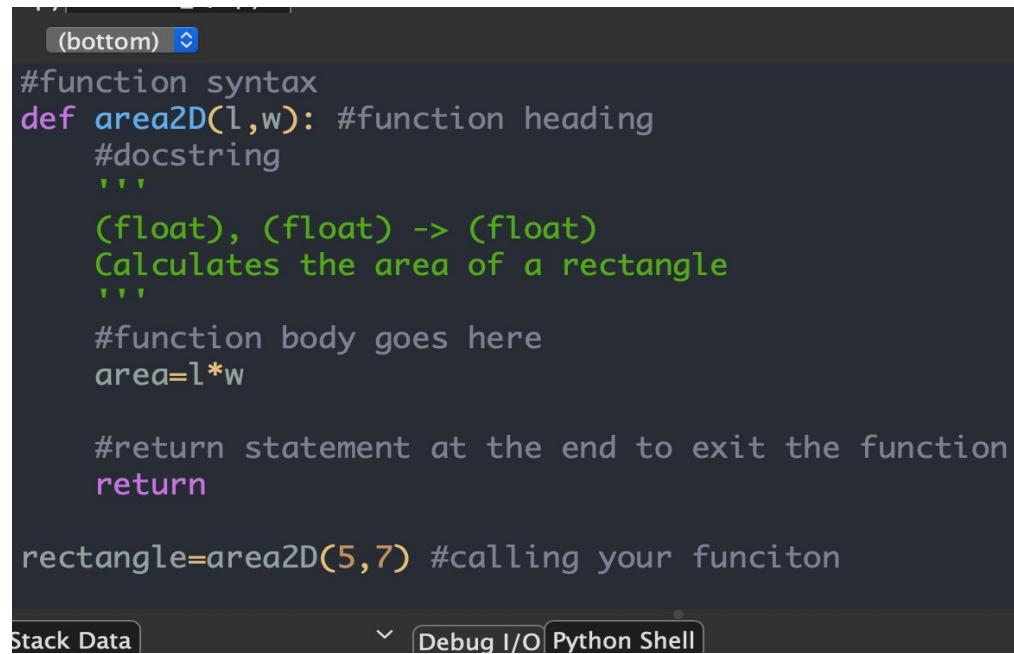
- Try typing the following statements in the python shell to see what happens.

```
>>> print(rectangle)
```

```
>>> rectangle
```

- Try adding the following statements to the Python script:

```
print(rectangle)  
rectangle
```



```
(bottom) ↴  
#function syntax  
def area2D(l,w): #function heading  
    #docstring  
    ''  
    (float), (float) -> (float)  
    Calculates the area of a rectangle  
    ''  
    #function body goes here  
    area=l*w  
  
    #return statement at the end to exit the function  
    return  
  
rectangle=area2D(5,7) #calling your funciton
```

Stack Data

Debug I/O Python Shell

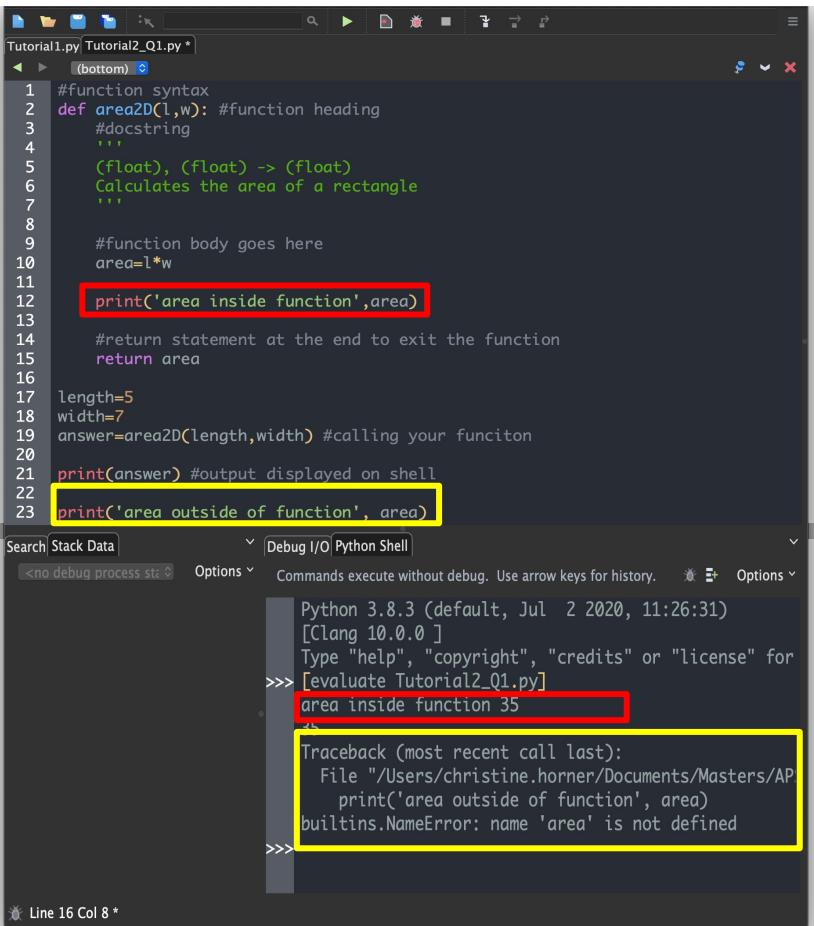
Review of Lecture

Functions: Tips & Misconceptions

if nothing else, write `#cleancode`

Variable Scope

- Variables declared and initialized **inside** a function are only accessible, aka visible, within the body of the function.
- Their values can be accessed from inside the function, e.g., they can be displayed in the shell by passing them as arguments to `print()` (**red case**), but cannot be accessed from outside the function, e.g., from the main program body (**yellow case**)



The screenshot shows a code editor and a terminal window. The code editor displays a Python script named `Tutorial1.py` with the following content:

```
1 #function syntax
2 def area2D(l,w): #function heading
3     #docstring
4     """
5         (float), (float) -> (float)
6         Calculates the area of a rectangle
7     """
8
9     #function body goes here
10    area=l*w
11
12    print('area inside function',area)
13
14    #return statement at the end to exit the function
15    return area
16
17 length=5
18 width=7
19 answer=area2D(length,width) #calling your funciton
20
21 print(answer) #output displayed on shell
22
23 print('area outside of function', area)
```

The terminal window below shows the execution of the script:

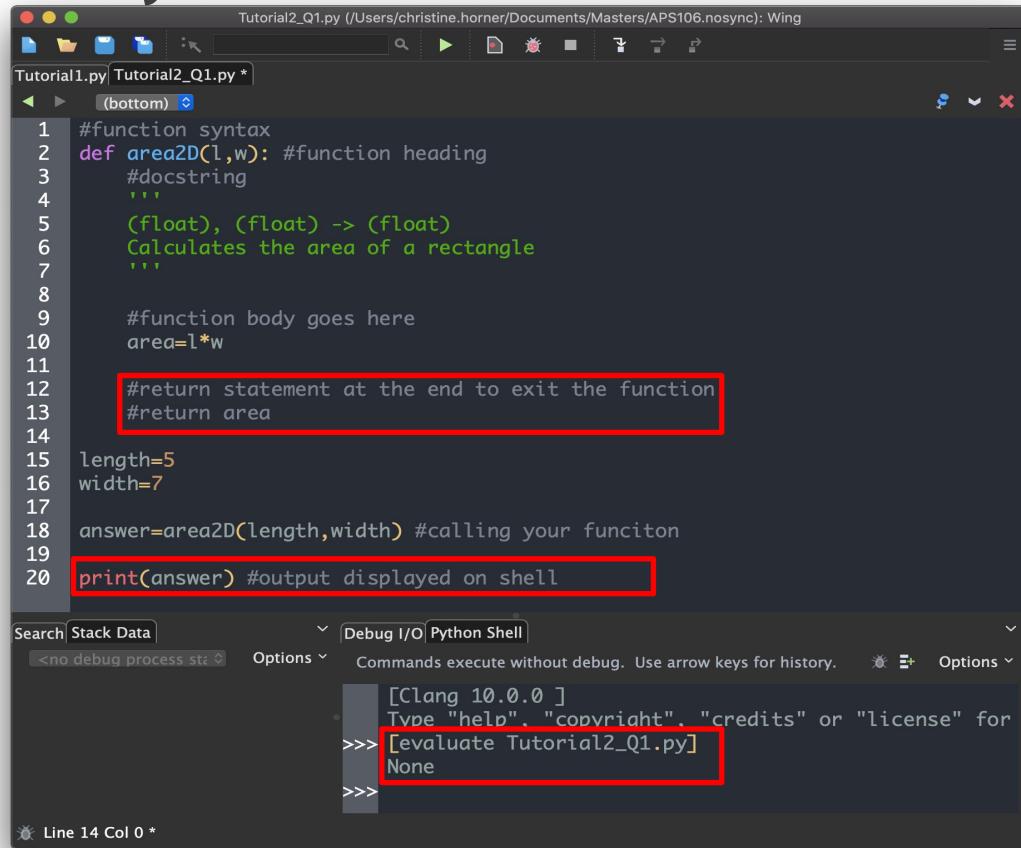
```
<no debug process selected> Options Commands execute without debug. Use arrow keys for history. Options
Python 3.8.3 (default, Jul 2 2020, 11:26:31)
[Clang 10.0.0]
Type "help", "copyright", "credits" or "license" for
>>> [evaluate Tutorial1.py]
area inside function 35
Traceback (most recent call last):
  File "/Users/christine.horner/Documents/Masters/AP
      print('area outside of function', area)
builtins.NameError: name 'area' is not defined
>>>
```

Annotations highlight specific parts of the code and output:

- A red box highlights the line `print('area inside function',area)` in the code editor.
- A red box highlights the output `area inside function 35` in the terminal.
- A yellow box highlights the error message `builtins.NameError: name 'area' is not defined` in the terminal.

Python Functions Always Return a Value !¹

A Python function whose body does not contain a return statement will return by default the value **None**.



The screenshot shows the Wing IDE interface with the following details:

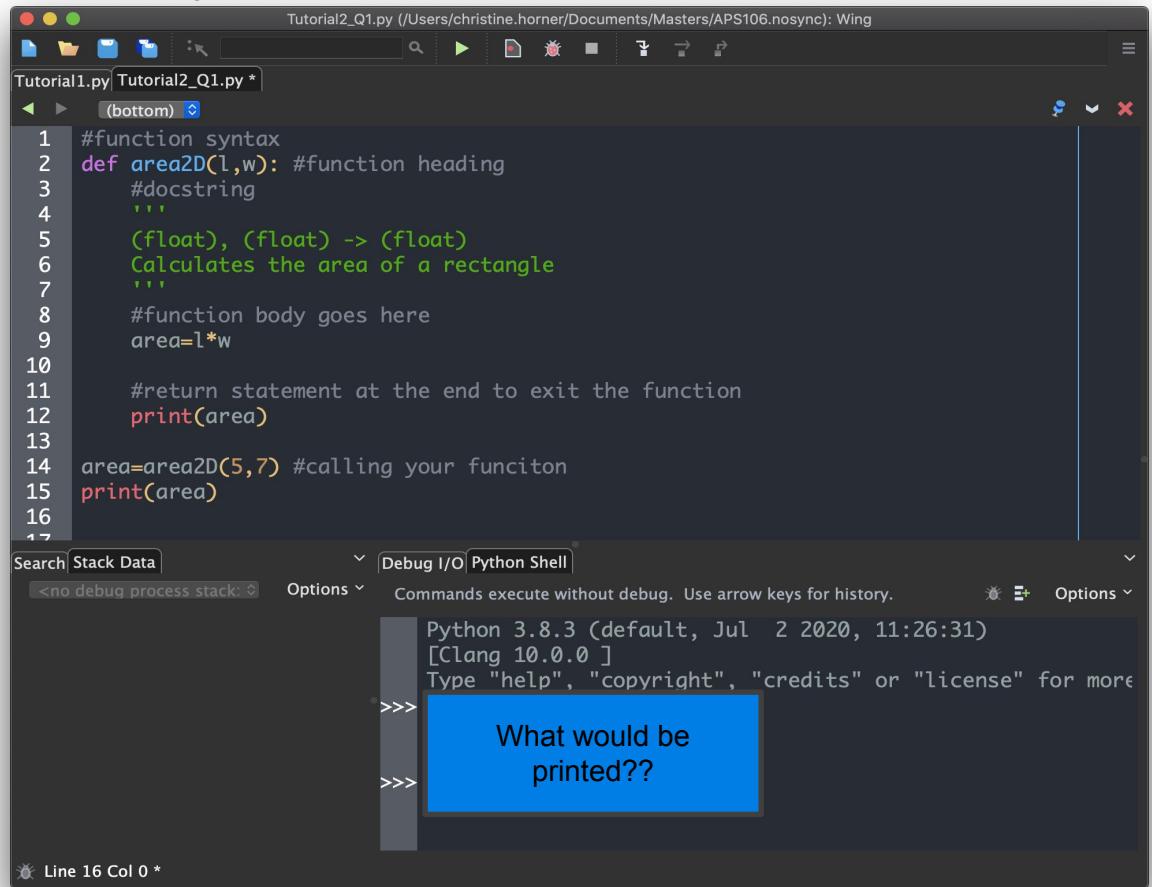
- Code Editor:** The file `Tutorial2_Q1.py` is open. The code defines a function `area2D` that calculates the area of a rectangle. The code includes a docstring and a comment indicating it calculates the area of a rectangle. It contains a `return` statement at the end of its body.
- Terminal:** The Python Shell tab is active. The user has run the command `[evaluate Tutorial2_Q1.py]`. The output shows the prompt `>>>` followed by `None`, which is highlighted with a red box.
- Status Bar:** The status bar at the bottom left indicates "Line 14 Col 0 *".

Python Functions Always Return a Value !²

What would be printed in the Python shell ?

If you get this, you understand how functions work 😊

If not, ask more questions !



The screenshot shows the Wing IDE interface. The top menu bar has 'File', 'Edit', 'View', 'Tools', 'Help', and a 'Wing' icon. The title bar says 'Tutorial2_Q1.py (/Users/christine.horner/Documents/Masters/APS106.nosync): Wing'. The main area displays Python code:

```
1 #function syntax
2 def area2D(l,w): #function heading
3     #docstring
4     ...
5     (float), (float) -> (float)
6     Calculates the area of a rectangle
7     ...
8     #function body goes here
9     area=l*w
10
11    #return statement at the end to exit the function
12    print(area)
13
14 area=area2D(5,7) #calling your funciton
15 print(area)
16
```

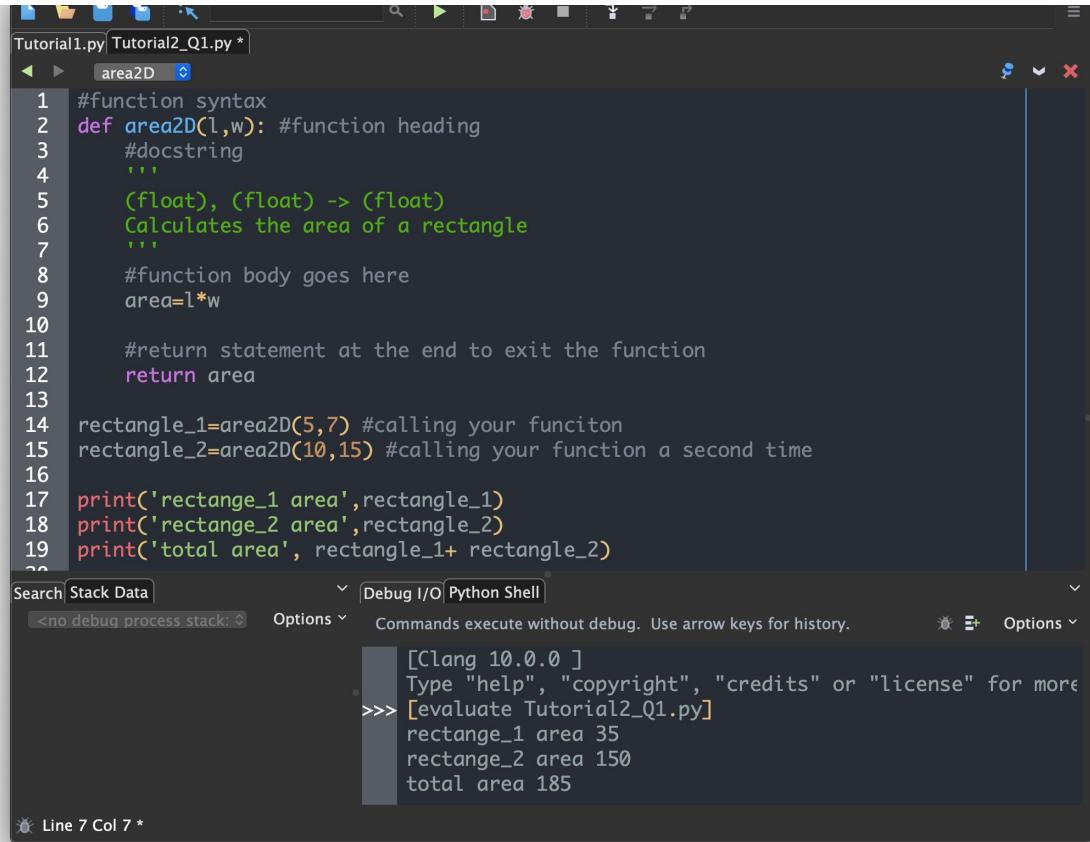
Below the code editor is a toolbar with icons for file operations. The bottom part of the interface is the Python Shell window:

- Search, Stack Data, Debug I/O, Python Shell tabs.
- Stack Data dropdown: <no debug process stack: ▾
- Options dropdown: Commands execute without debug. Use arrow keys for history.
- Python Shell output:

```
Python 3.8.3 (default, Jul 2 2020, 11:26:31)
[Clang 10.0.0]
Type "help", "copyright", "credits" or "license" for more
```
- Input prompt: >>>
- A blue callout box with white text: "What would be printed??"
- Line 16 Col 0 *

When Do We Need to Use return Statements?

What if you wanted
to find the total area
of two rectangles ?



```
Tutorial1.py Tutorial2_Q1.py *
▶ area2D
1 #function syntax
2 def area2D(l,w): #function heading
3     #docstring
4     """
5         (float), (float) -> (float)
6         Calculates the area of a rectangle
7         """
8     #function body goes here
9     area=l*w
10
11    #return statement at the end to exit the function
12    return area
13
14 rectangle_1=area2D(5,7) #calling your funciton
15 rectangle_2=area2D(10,15) #calling your function a second time
16
17 print('rectange_1 area',rectangle_1)
18 print('rectange_2 area',rectangle_2)
19 print('total area', rectangle_1+ rectangle_2)
20

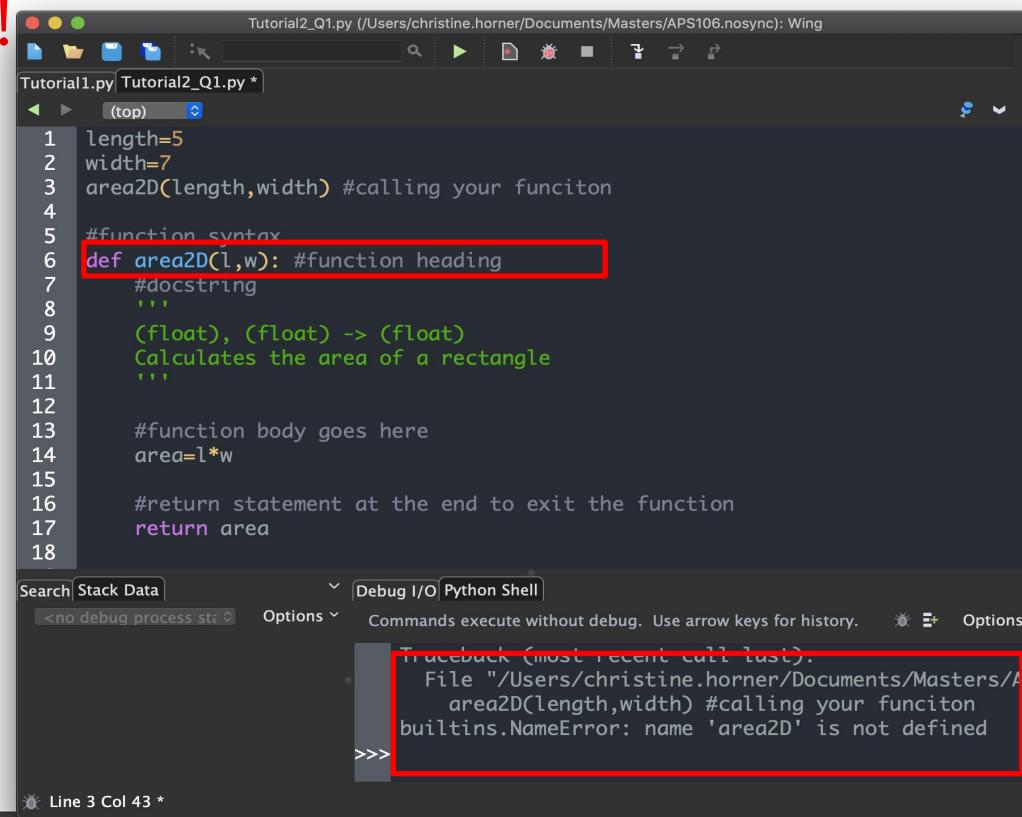
Search Stack Data Debug I/O Python Shell
<no debug process stack: Options Commands execute without debug. Use arrow keys for history. Options
[Clang 10.0.0 ]
Type "help", "copyright", "credits" or "license" for more
>>> [evaluate Tutorial2_Q1.py]
rectange_1 area 35
rectange_2 area 150
total area 185
Line 7 Col 7 *
```

Functions must be defined before being called... or you will get an error !

The error message is telling us that Python has no idea what `area2D` is.

This is a semantic error caused by the programmer calling a function before it was defined.

⇒ Remember that the Python interpreter runs your code line by line. The statement containing the definition of `area2D` had not been reached at the time the statement in line 3 was executed.



The screenshot shows a Python script named `Tutorial2_Q1.py` in a code editor. The code defines a function `area2D` and calls it immediately. A red box highlights the line `def area2D(l,w): #function heading`. The code is as follows:

```
length=5
width=7
area2D(length,width) #calling your funciton
#function syntax
def area2D(l,w): #function heading
    """
    (float), (float) -> (float)
    Calculates the area of a rectangle
    """
    #function body goes here
    area=l*w
    #return statement at the end to exit the function
    return area
```

Below the code editor is a terminal window showing the execution of the script. A red box highlights the error message:

```
Traceback (most recent call last):
  File "/Users/christine.horner/Documents/Masters/APS106/nosync", line 3, in <module>
    area2D(length,width) #calling your funciton
builtins.NameError: name 'area2D' is not defined
```

Line 3 Col 43 *

APS106



Review of Lecture

Built-in Functions

if nothing else, write `#cleancode`

Lecture Content: Built-in Functions!¹

Built-in Functions: functions that come pre-packaged with a Python implementation

`abs(-23)`

`int(4.2)`

`abs(56.24)`

`float(8)`

`type('hello')`

`id(4)`

- Other built in functions can be found here:
<https://docs.python.org/3.8/library/functions.html>
- Use `help()` to look up the documentation for a function, e.g.,
`help(abs)`

Lecture Content: Functions²

`input([prompt])`

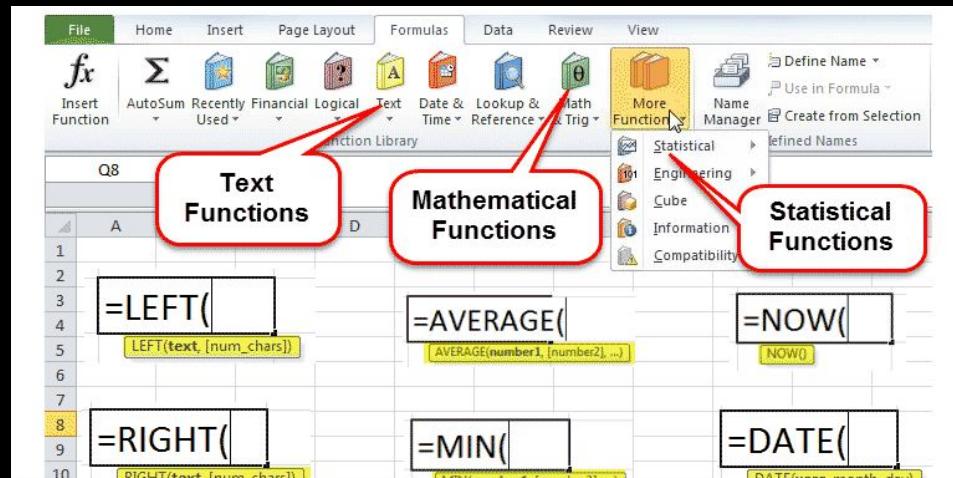
- If the *prompt* argument is present, it is written to standard output. The function then reads the line entered by the user in the command line/shell, converts it to a string (stripping a trailing newline), and returns that.
 - Depending on how we want to process the value entered by the user in our code, we may need to explicitly convert the string returned by `input()` using typecasting

`print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)`

- Print *objects* to the text stream *file*, separated by *sep* and followed by *end*.
 - *sep*, *end*, *file*, and *flush*, if present, must be given as keyword arguments.

Examples of functions you might have used...

- Excel!



- Think of your favourite apps on your phone

APS106



Review of Lecture

Importing and Using Modules

if nothing else, write `#cleancode`

Python Modules

- Various statements, including function and class definitions (which will be introduced in later lectures) can be stored in separate Python files, called **modules**.
- To get access to the functions defined in a module, you need to import the module.
 - Typical form of importing a module:

```
import module_name
```

- To access a function within a module:

```
module_name.function_name
```

Python's Built-in modules Modules

- Built-in modules we use in this course: **math** and **csv**.
- Example: **import math**

```
print(math.abs(-4))
```

- There are many other cool, open source Python libraries, e.g., NumPy, Pandas, scikit-learn, but we won't cover them in this course and you are not allowed to use them in lab assignments or exams!

APS106



Practice Problems

if nothing else, write `#cleancode`



What is printed when this code is run?

- ⓘ Start presenting to display the poll results on this slide.

Review Practice Problem 1

```
# Function practice problem 1
def my_func():
    x = 1
    print (x)
    x = x + 1
    return x

x = 4
print(x)
x = my_func()
print(x)
```

What is printed when this code is run?

4,5,5

4,1,4

4,1,2

4,1,1

None of the above



What is the mistake in this code?

- ① Start presenting to display the poll results on this slide.

Review Practice Problem 2

```
(bottom) ▾  
#function syntax  
def area2D (l,w): #function heading  
    #docstring  
    ...  
    (float) -> (float)  
    Calculates the area of a rectangle  
    ...  
  
    # function body goes here  
    area=l*w  
  
    # return statement at the end to exit the function  
    return area  
  
  
l=input('enter length: ') #input from user  
w=input('enter width: ') #input from user  
  
answer=area2D(l,w) #calling your function  
  
print(answer) #output displayed on shell
```

What is the mistake in this code?

Coding Problem 1

Write a function to convert an angle expressed in degrees to radians.

HINT: You can import modules. Are there any built-in functions from the `math` module that can help ?

Things to consider before you start writing your code:

1. What would be the parameters of your function?
2. What should the function output?

Coding Problem 2

Write a function with 3 inputs: the minimum speed limit and maximum speed limit of that street, as well as your current speed

Return a number to the user telling them whether they are obeying speed limit rules (number itself), and if not, which speed limit they are breaking (minimum or maximum)

Hint:

There are functions that take in values and return their minimum and maximum

slido



Any questions?

- ⓘ Start presenting to display the poll results on this slide.