ITP 115 – Programming in Python

Functions

Review

Outline

- Write your own functions
- Accept values into your functions through parameters
- Return information from your functions through return values
- Work with global variables and constants

Functions

- Go off and perform a task and then return control to your program
- Allow you to break up your code into manageable, bite-sized chunks
- Programs with functions can be easier to create and work with

Why Use Functions?

- Reuse code!
  - Write once, use multiple time
- Better code!
  - Fewer errors
- Easier to read code!
  - Code is "self-explanatory"
  - Remember **`print()`**? Let's see what it actually does

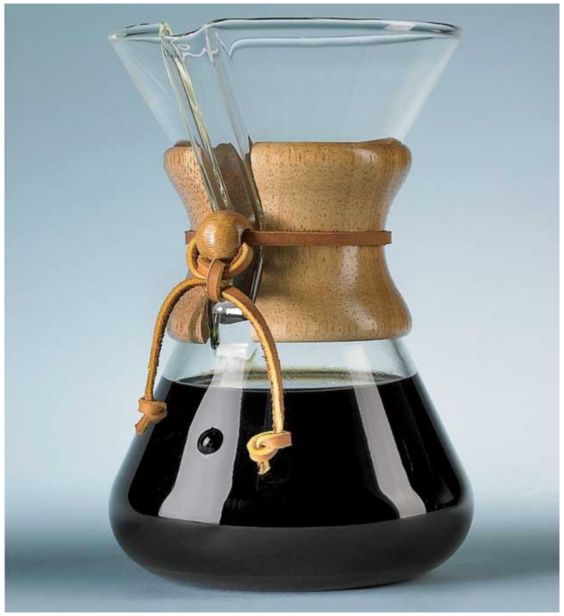
Two Steps to Using Functions

- Function Definition
  - What the function DOES (in theory)
  - This is steps that you want to happen
  - Like a recipe

- Function Call
  - Actually USING the function (reality)

Function Definitions

- Recipe for Making Coffee
  - Grind beans
  - Heat water
  - Put water and grounds in pot
  - Brew coffee
  - Pour into cup



Function Calls

- Execute recipe (function call)

Defining a Function

- Use the word **def**, followed by a
  function name *(same rules as variables)*
  parentheses
  colon
  indented block

```
def functionName ():
   statements(s)
```

Defining a Function

- Examples

```
# define a function called spam
def spam():
   print("spam, spam, spam")

# define a function called showWeather
def showWeather():
   weather = int(input("What is the temperature?"))
   if temp > 80:
      print("It seems hot!")
   else:
      print("I bet it's cold")
```

Calling a Function

- To call a function, use the name of the function followed parentheses

```
functionName()
```

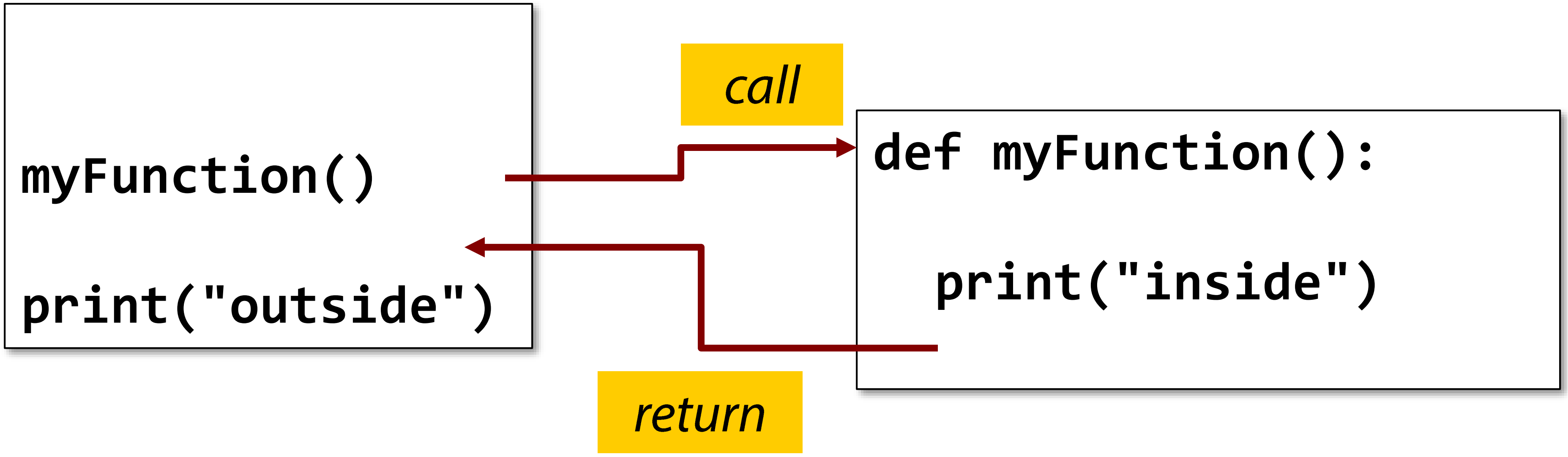
- Must define the function **before** you call it

- Example

```
spam()
spam()
showWeather()
```

```
spam, spam, spam
spam, spam, spam
What is the temperature? 90
It seems hot!
```

Flow of Control with Functions



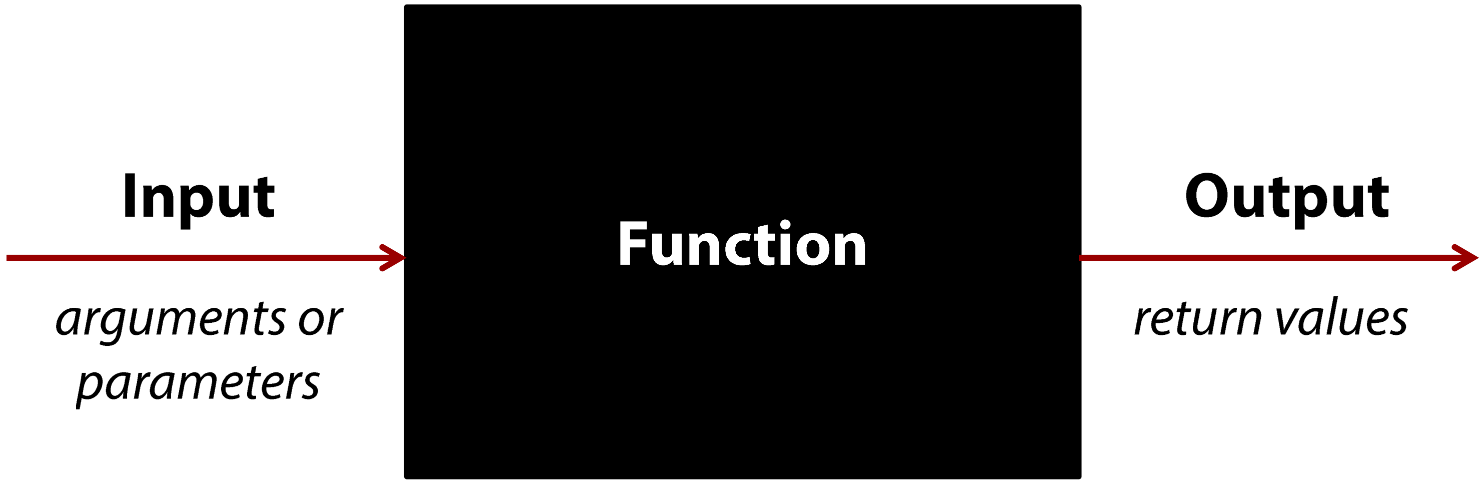
```
myFunction()

print("outside")
```

```
def myFunction():

    print("inside")
```

Functions with Input and Output



Using Parameters

- Can add parameters when you define your function
  - Multiple parameters need to be separated by commas

```
def functionName(parameter1, parameter2):
    statement(s)
```

function definition

- Call the function with the same number of arguments as the function has parameters

```
functionName(argument1, argument2)
```

function call

Parameters Example

```
# define a function with a parameter
def display(message):
    print(message)

# call the function with a parameter
display("Hi Mom")
```

```
Hi Mom
```

```
# define a function with a parameter
def displaySum(num1, num2):
    print(num1 + num2)

# call the function with a parameter
displaySum(4, 8)
```

```
12
```

Using Positional Parameters

- Most common way to pass arguments to functions
- Parameters get their values based on the position of the values sent
- The 1st parameter gets the 1st value sent, the 2nd parameter gets the 2nd value sent, etc.

Using Positional Parameters

```
# positional parameters
def birthday(name, age):
  print("Happy Birthday " + name + "! You are " + str(age))

# call birthday
birthday("Chuck", 25)
birthday("Sheila", 45)
```

```
Happy Birthday Chuck!  You are 25
Happy Birthday Sheila!  You are 45
```

Using Default Parameter Values

- You can assign default values to your parameters
  - Parameters get assigned these values if no value is passed to them
- Ex: **`print`** function
  - There is a default value given to the parameter **`end`**
  - When you say **`end="  "`**, we override the default value
- Note: once you assign default values to a parameter in a list, you have to assign default values to all the parameters listed after it

Using Default Parameter Values

```
# default parameters
def birthday(name = "Cooper", age = 1):
 print("Happy Birthday " + name + "! You are " + str(age))

# call birthday
birthday()
birthday("Tracy", 39))
birthday(name = "Carter")
birthday(age = 6)
birthday(name = "Carter", age = 6)
```

```
Happy Birthday Cooper! You are 1
Happy Birthday Tracy! You are 39
Happy Birthday Carter! You are 1
Happy Birthday Cooper! You are 6
Happy Birthday Carter! You are 6
```

Keyword Arguments

- Assign values to specific parameters, regardless of order

- Use the actual parameter names from the function header to link a value to a parameter

Keyword Arguments

```
# positional parameters with keyword arguments
def birthday(name, age):
 print("Happy Birthday " + name + "! You are " + str(age))

# call birthday
birthday(name = "Evan", age = 7)
birthday(age = 4, name = "Quinn")
```

```
Happy Birthday Evan!  You are 7.
Happy Birthday Quinn!  You are 4.
```

Using Return Values

- When you make a function call, the function can also return a value *(think "give back a value")*

- Return values can be stored in variable
  - Ex: `len()` function returns get the length of a sequence

    ```
    wordLength = len("Gibraltar")
    ```

- To return value, use `return` followed by the value you want to return

Using Return Values

- Function definition

```
def functionName (parameters):
    statement(s)
    return value
```

- Function call

```
var = functionName (argument)
or
print(functionName (argument))
```

Using Return Values

```
# define a function that has a return value
def doubler(x):
    return x*2

# call a function that has a return value
num = doubler(2)
print(num)
print(doubler(2.2))
print(doubler("Hi"))
```

```
4
4.4
HiHi
```

Multiple Return Values

- A function can return multiple values
  - This is not allowed by most programming languages
- List all the values to return separated by commas
- Make sure to have enough variables to catch all the return values of a function

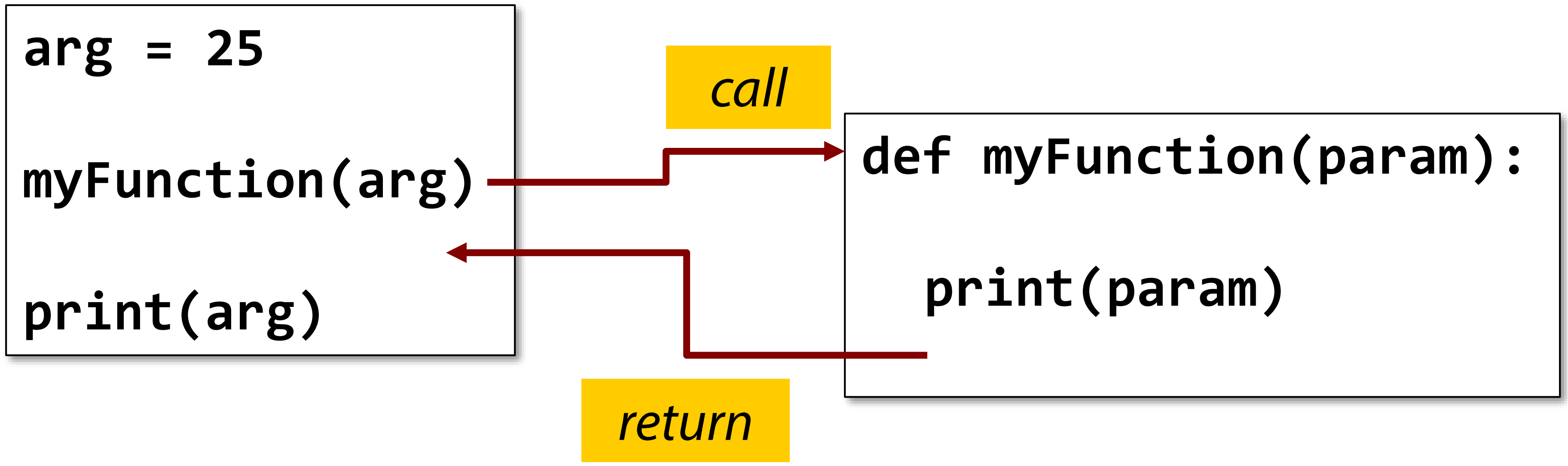
Multiple Return Values

```
# define a function with multiple return values
def times2_power2 (x):
    return x*2, x**2

# call a function with multiple return values
num = 5
numX2, numPower2 = times2_power2 (num)
print(num, "X 2 =", numX2)
print(num, "^ 2 =", numPower2)
```

```
5 X 2 = 10
5 ^ 2 = 25
```

Flow of Control with Functions



```
arg = 25

myFunction(arg)

print(arg)
```

```
def myFunction(param):

   print(param)
```

- *End lecture*

`main()` Function

- `main()` is often used as the starting point of a larger program
  - From now on in assignments, `main()` will contain your "main" program
- In Python, the word `main()` has no special meaning
  - But it is a programming convention to call this ***starting function*** `main()`

`main()` Function

- All your code which used to be *aligned left* in your file will now be in your `main()` function

- `main()` will call other functions as needed

`main()` Function

- Takes no arguments and returns no values
- Functions can be defined in any order as long as a function call to `main()` is called at the end of the file

Example

```
def main():
    number = int(input("Enter a number: "))
    result = square(number)
    print("The square of", number, "is", result)

def square(x):
    return x * x

main()
```

In addition to defining **`main()`***, we still must call it at the end of the file*

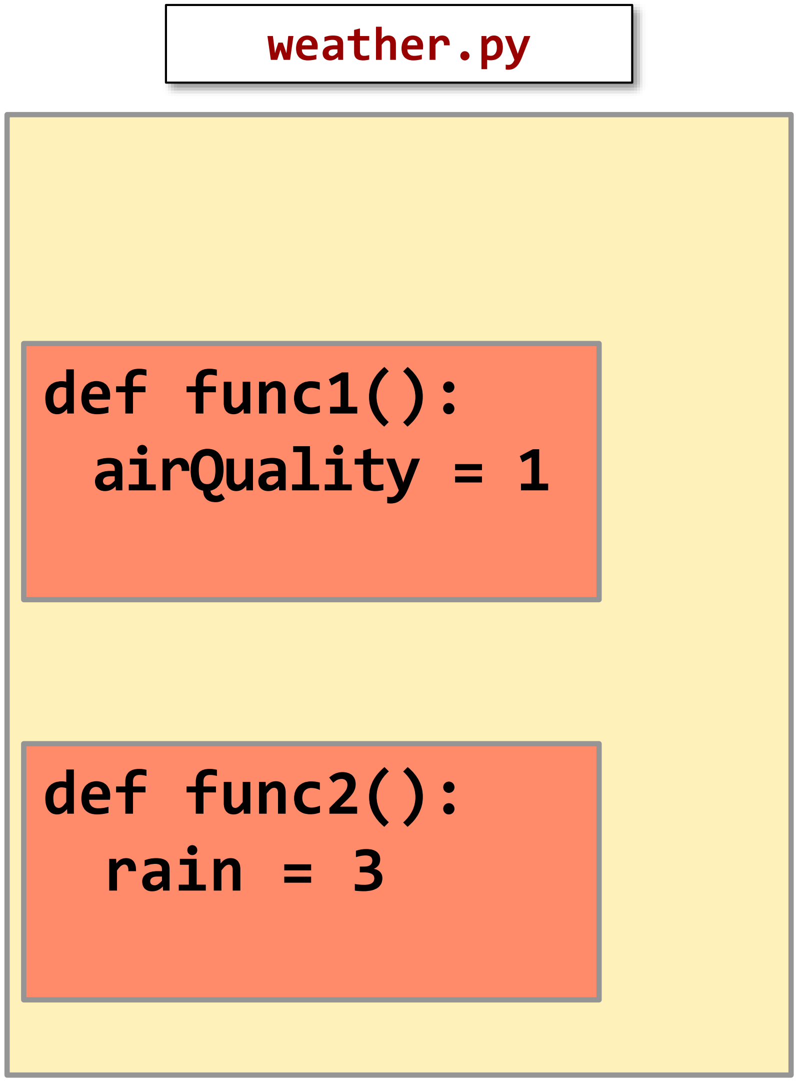
Namespaces

- **Namespaces** (also called *scopes*) represent different areas of your program that are separate from each other

- Each function you define has its own namespace
  - A function can't access a variable in another function

- Think of **namespaces** as a table that lists all the variables (and other things) that it contains

Namespaces



weather.py

```
def func1():
  airQuality = 1
```

```
def func2():
  rain = 3
```

- **`airQuality`** is a local variable
  - Can be accessed ONLY from **`func1()`**
- **`Rain`** is a local variable
  - Can be accessed ONLY from **`func2()`**

Aside: Constants

- A **constant** is a variable that can not change
- Constants can be useful to ensure some important data never changes
  - Ex: Sales tax rate or speed of light
- Style: constants are **ALL_CAPS_WITH_UNDERSCORES**
  - Ex. **`SALES_TAX_RATE`** or **`SPEED_OF_LIGHT`**

Global Constants

- Global constants are **constants** created in the global namespace
  – This means on the far left of the file

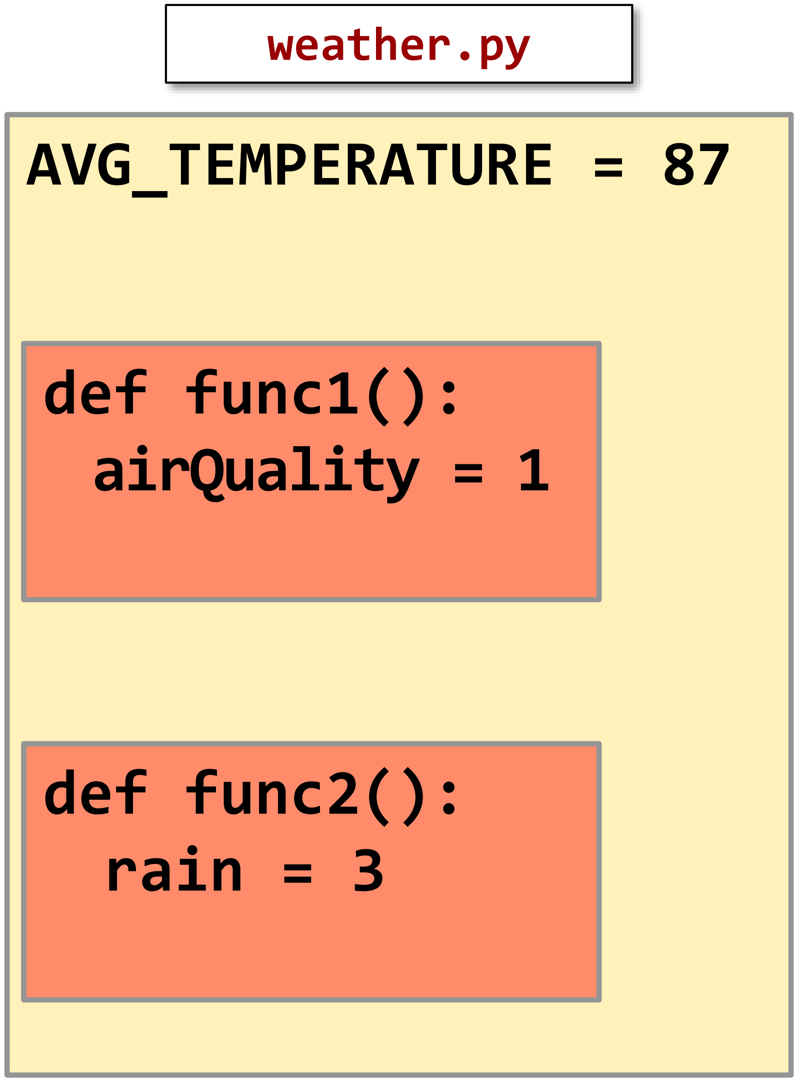
- Global constants can be access from everywhere in your program (e.g. inside functions)

- Global constants can not change their values once they are assigned

Aside: Global Constants

- Technically, Python doesn't have true global constants
  - It has global variables that we AGREE not to change
- It is possible to change a global variable from inside a function if you use the keyword **global** *(ch 10)*
- However, in our class we won't do this and will treat these as **constants**

Namespaces



```
AVG_TEMPERATURE = 87

def func1():
  airQuality = 1

def func2():
  rain = 3
```

- **`AVG_TEMPERATURE`** is a global constant
  - Can be accessed from within any function
- **`airQuality`** is a local variable
  - Can be accessed ONLY from **`func1()`**
- **`rain`** is a local variable
  - Can be accessed ONLY from **`func2()`**