# ITP 115 – Programming in Python

## Functions

# 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
- Easier to read code!
  - Code is "self-explanatory"
  - Remember **`print()`**? Let's see what it actually does
- Better code!
  - Fewer errors

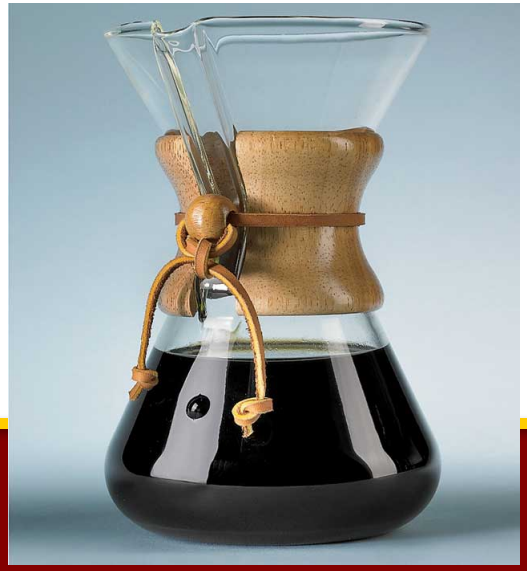
# Two Steps to Using Functions

- Function Definition
  - What the function DOES (in theory)
  - This is the 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():
    statement(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 weather > 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 by parentheses

```
functionName()
```

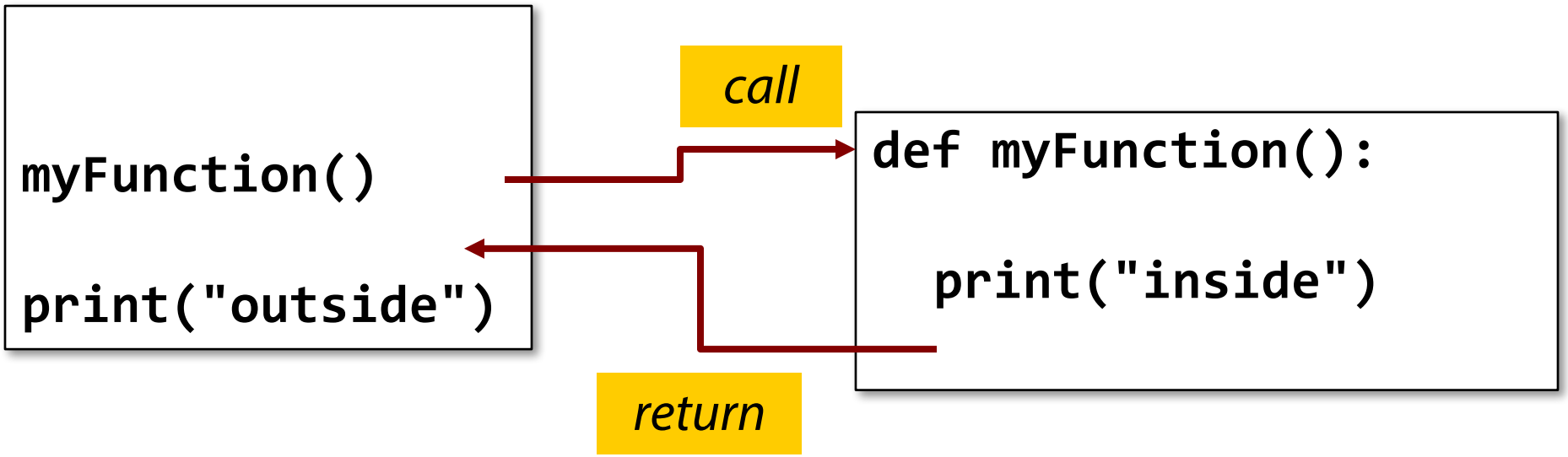
- Must define the function **before** you call it

- Example

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

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

# Flow of Control with Functions



# 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()`

# Namespaces

weather.py

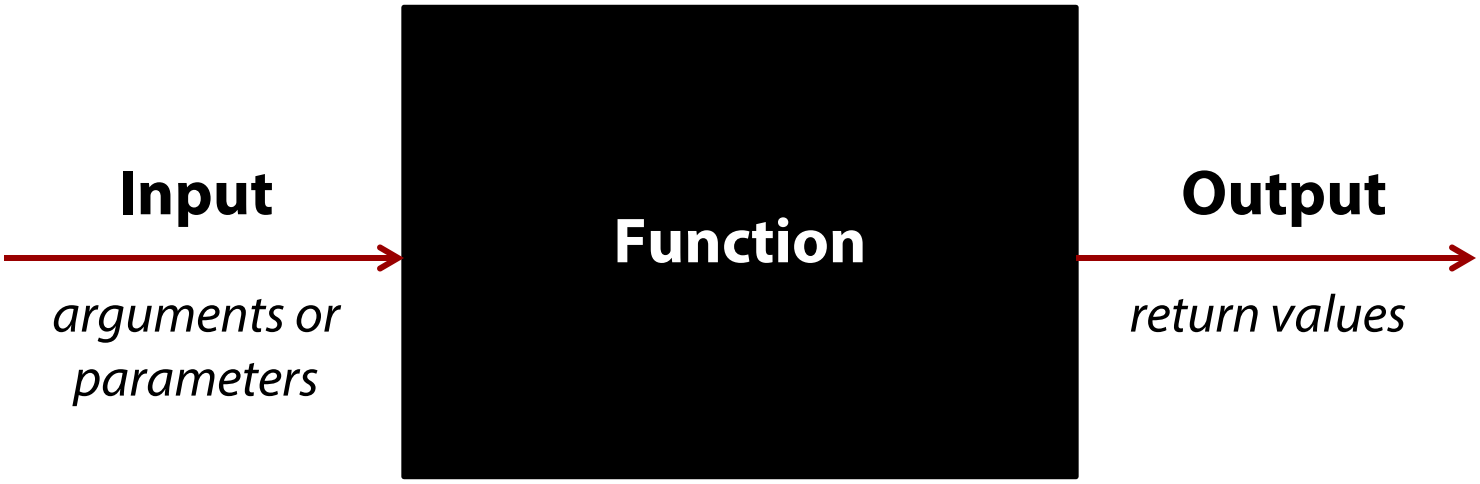
```
def func1():
  airQuality = 1
  print(rain)

def func2():
   rain = 3
   print(airQuality)

def main():
   func1()    #error!
   func2()    #error!
```

- Calling/running either of these functions will result in **errors**
- `func1()`
  - `rain` was not defined inside this function
- `func2()`
  - `airQuality` was not defined inside this function

# Namespaces

- Namespaces are also referred to as **scope**.
- Think of it as the area of code where variables can be used.
- Variables defined inside of a function can only be used inside of that function.

# Functions with Input and Output



# Input Parameters and Arguments

- Parameters are the way we provide input to a function

- Examples

```
random.randrange(3, 10)
```

  - The **randrange** function generates a random number between the first parameter (3) and up-to-but-including the second parameter (10)

```
print("hello world")
```

  - The **print** function take a parameter and displays the parameter on the screen

# 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")
```

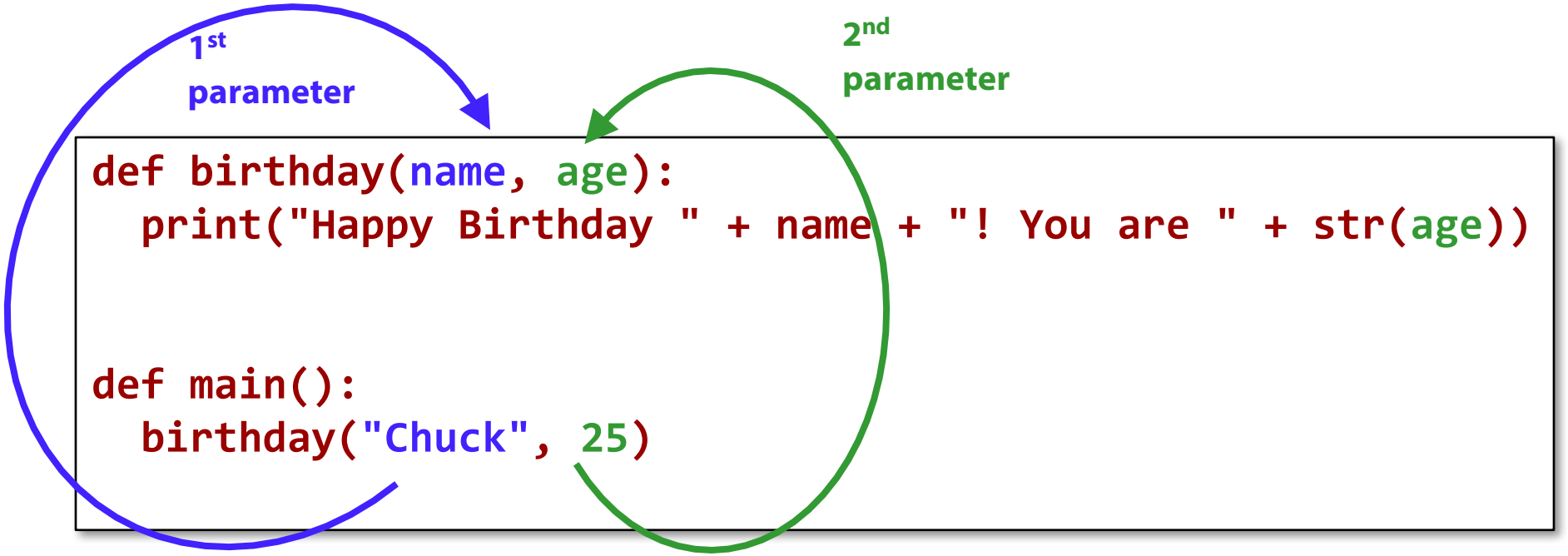
```
Hi Mom
```

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

# call the function with a parameter
displayTwice("Hello")
```

```
HelloHello
```

# Positional Parameters

- Multiple parameters get their values based on the position of the values sent in the function call
- 1st parameter gets 1st value, 2nd parameter gets 2nd value, etc.



# Position Parameter Examples

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

birthday("Chuck", 25)
birthday("Sheila", 45)
birthday("Kevin", "Briana", 35)  # third call
```

```
Happy Birthday Chuck!  You are 25
Happy Birthday Sheila!  You are 45
# third call would produce an error
```

```
def displaySum(num1, num2):
    print(num1 + num2)

displaySum(4, 8)
```

```
12
```

# Parameter Types

- What happen we call the function, passing a different variable type than it expects?

```
# define the function
def displaySum(num1, num2):
    print(num1 + num2)

# call the function
displaySum("dog", "cat")
```

**string** parameters, not **ints**

```
dogcat
```

# Parameter Types cont.

- Passing a parameter that is a different type than what your function is expecting will not always work…

```
# define the function
def displaySum(num1, num2):
    print(num1 + num2)

# call the function
displaySum("dog", 9876)
```

```
Error
```

- **`displaySum()`** was expecting two numbers to add
- Passing a string parameter caused the program to crash

- *End lecture*

# Review of Functions

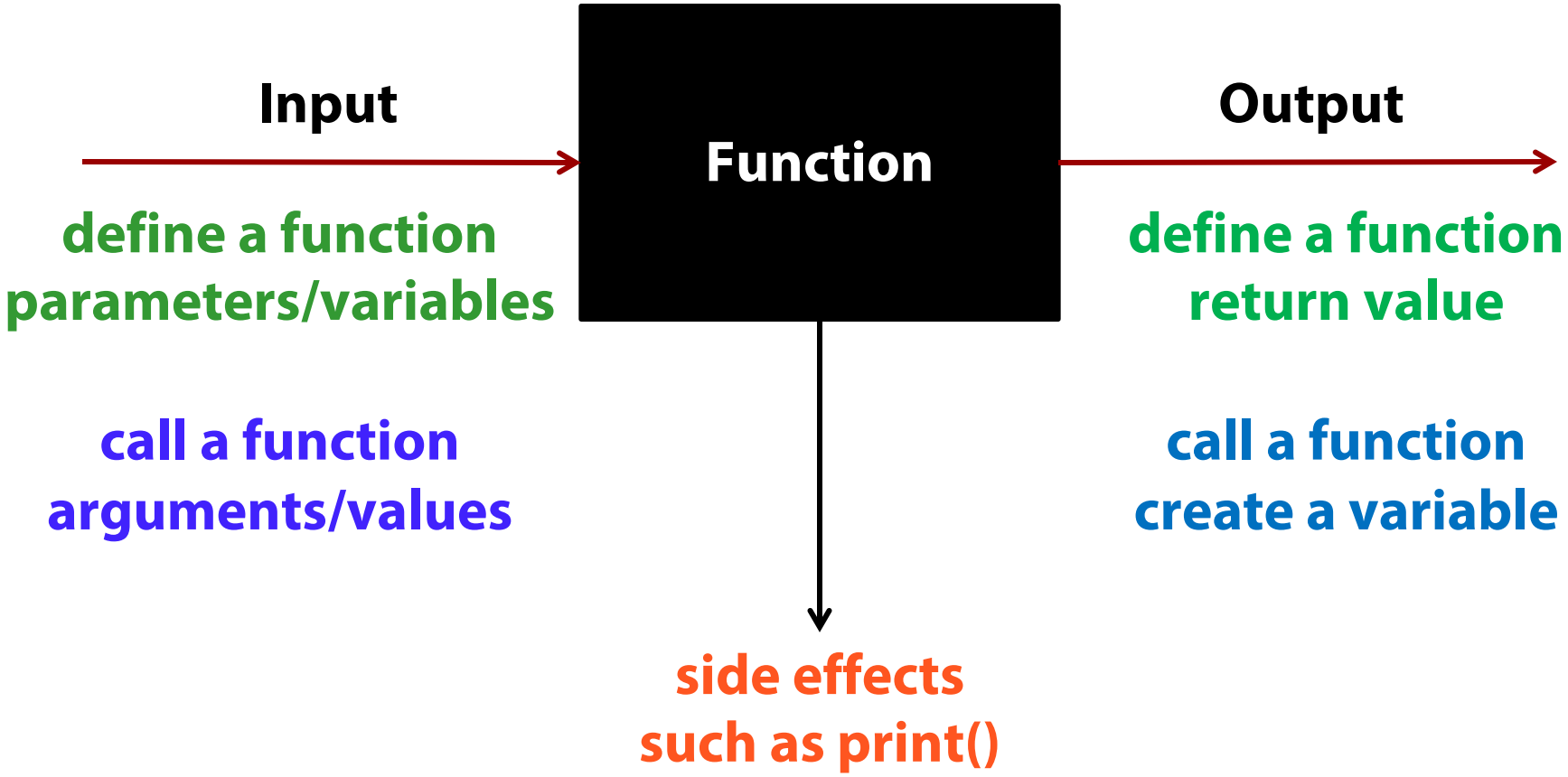
- Go off and perform a task and then return control to your program
- Allow you to break up your code into manageable, bite-sized chunks
- Programs with functions can be easier to create and work with

# Two Steps to Using Functions

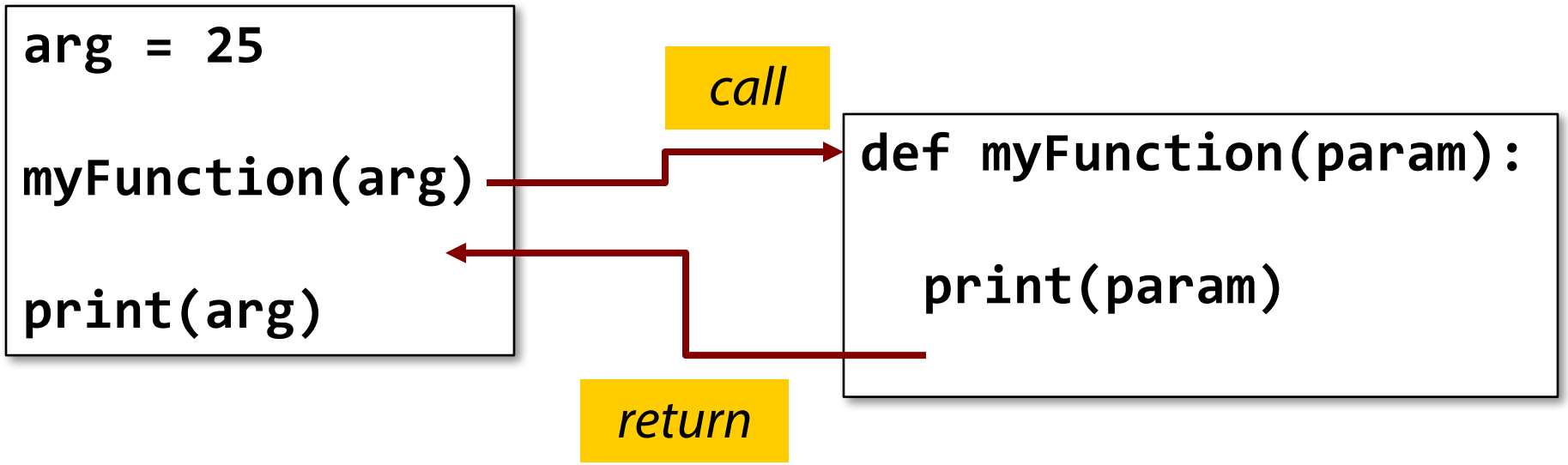
- Function Definition
  - What the function DOES (in theory)
  - This is the steps that you want to happen
  - Like a recipe

- Function Call
  - Actually USING the function (reality)

# Functions with Input and Output



# Flow of Control with Functions



# Input Parameters and Arguments

- Parameters are the way we provide input to a function

- Examples

```
random.randrange(3, 10)
```

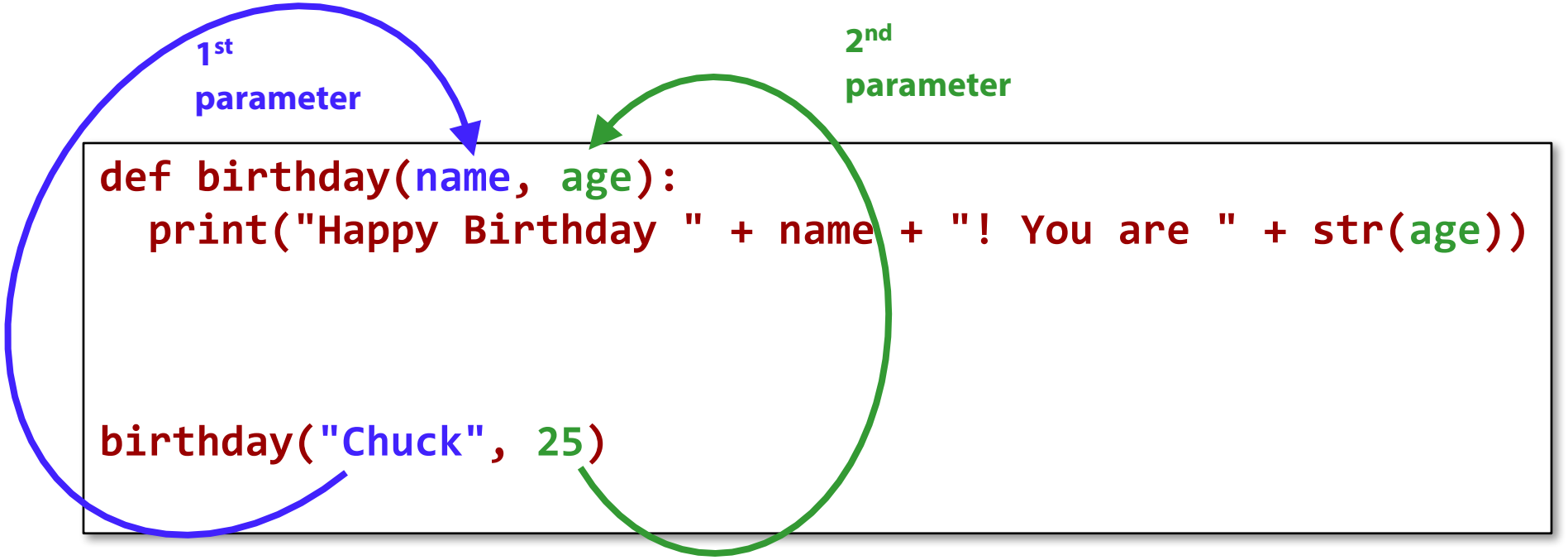
  - The **randrange** function generates a random number between the first parameter (3) and up-to-but-including the second parameter (10)

```
print("hello world")
```

  - The **print** function take a parameter and displays the parameter on the screen

# Positional Parameters

- Multiple parameters get their values based on the position of the values sent in the function call
- 1st parameter gets 1st value, 2nd parameter gets 2nd value, etc.



# 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 = 11):
 print("Happy Birthday " + name + "! You are", age)

# call birthday
birthday("Tracy", 48))
birthday("Carter")
birthday()
```

```
Happy Birthday Tracy! You are 48
Happy Birthday Carter! You are 11
Happy Birthday Cooper! You are 11
```

# 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 a 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
```

# main()

- We have actually been using functions since the first day of class!

```
def main():
    print("Hello World")
    print("Python is awesome")
main()
```

# `main()` Function

- **`main()`** is often used as the starting point of a larger program
- **`main()`** 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

- As we add more functions, `main()` will call other functions as needed
- 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 square(x):
    return x * x

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

main()
```

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