Module 06

Functions

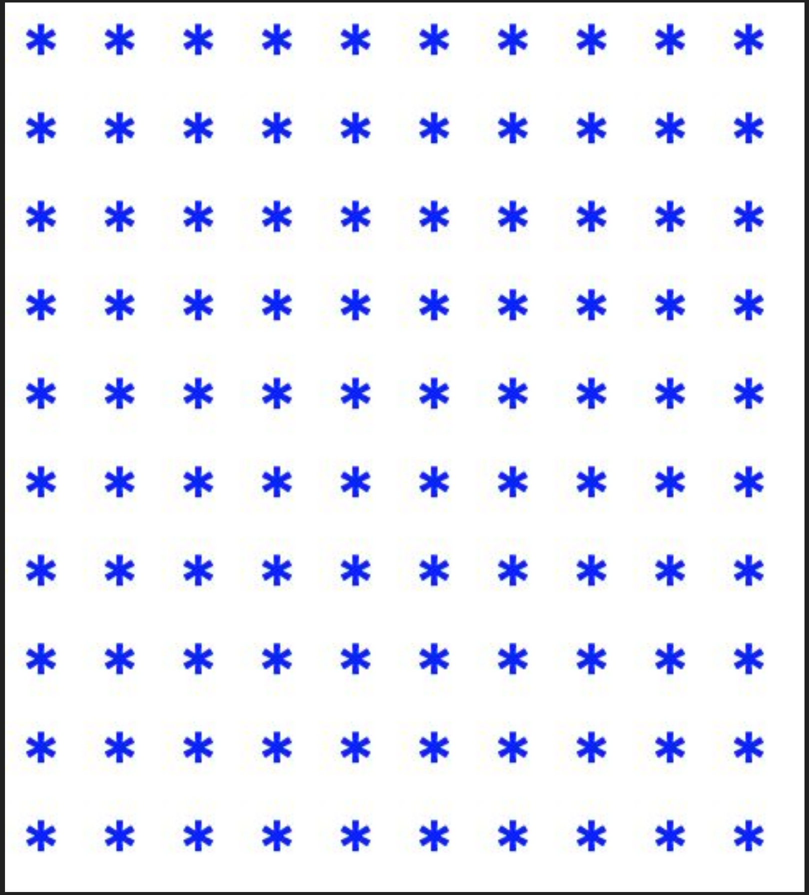
Emily Zhao
CSCI–UA.0002

Agenda

- Review For Loops
- Review Module 6 + Quiz 6
- Practice Problems

“For” loop review

Grid of Asterisks



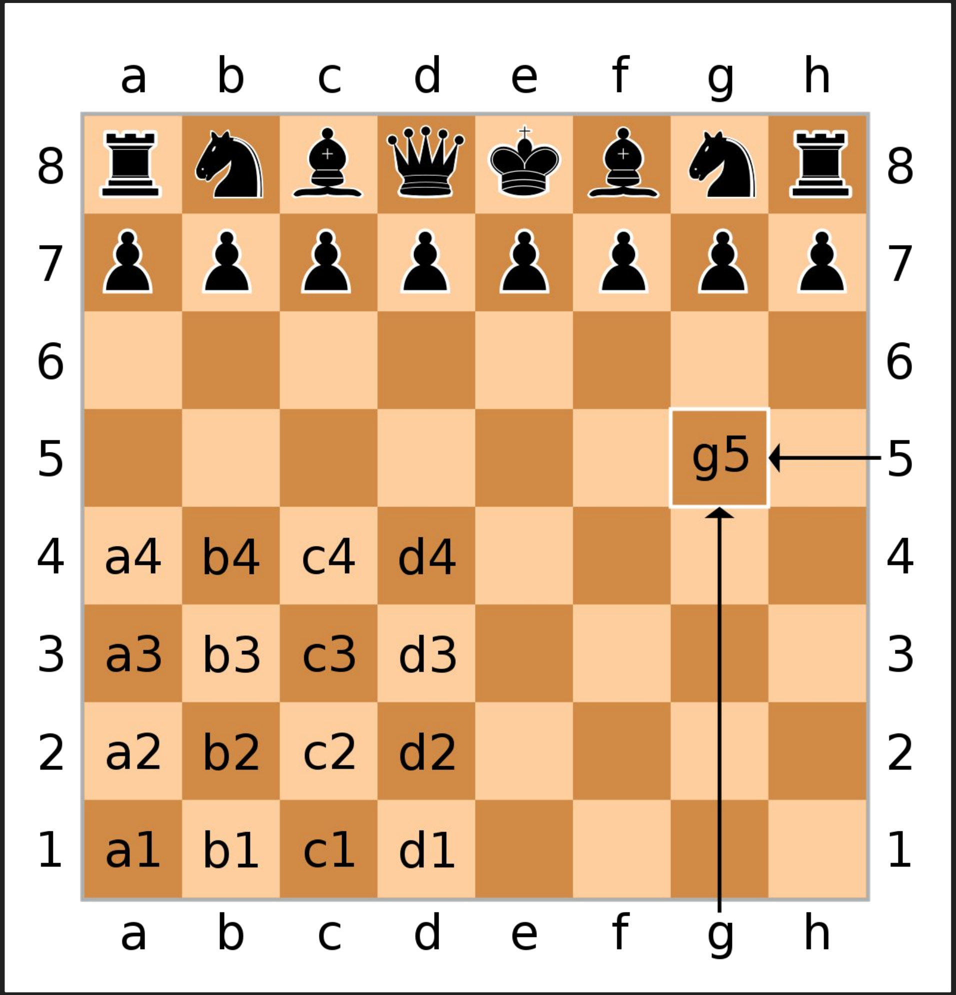
```
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
```

1. Generate a 10x10 grid of asterisks using **for loop(s)**

2. Change your code so that it can generate an any number by any number grid

```python
rows = 10
cols = 10

# single for loop
for r in range(rows):
    print("* " * cols)

# nested for loop
for r in range(rows):
    # prints 10 rows of 10 *s
    for c in range(cols):
        # prints * * * * * * * * * *
        print("*", end=" ")
    print() # prints new line at the end of each row
```

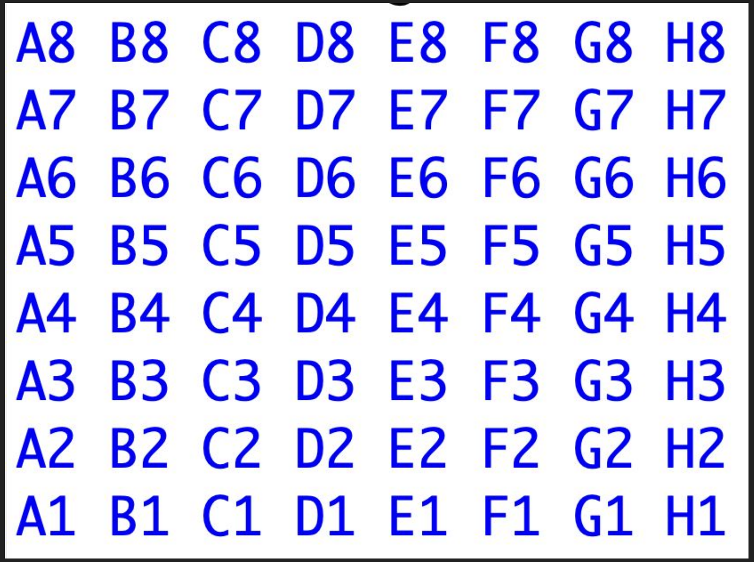
Chessboard



Generate a table of chess coordinates.

How many for loops do you need?

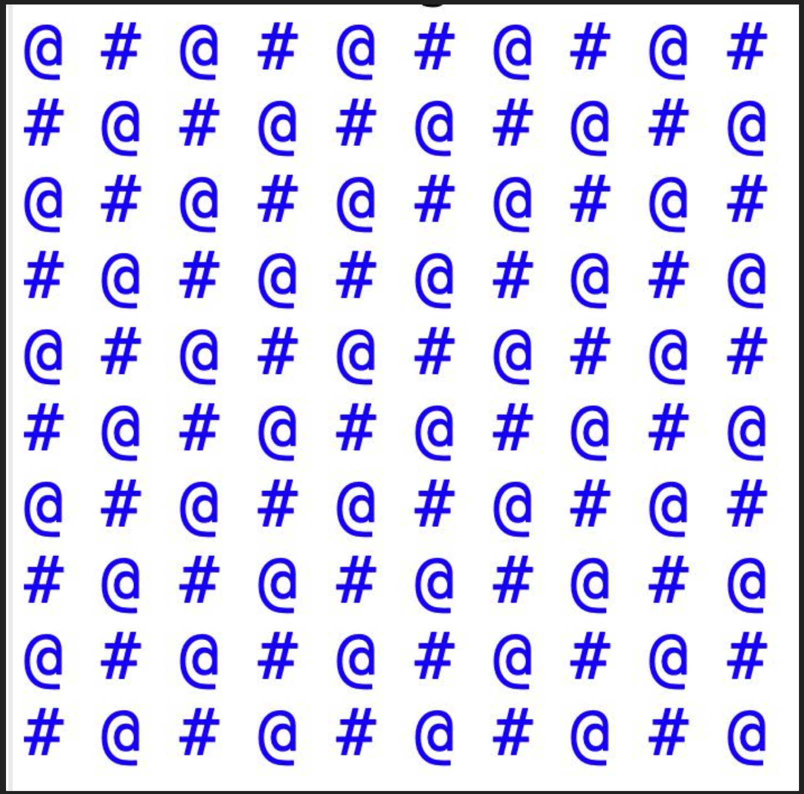
Expected Output:



```
A8 B8 C8 D8 E8 F8 G8 H8
A7 B7 C7 D7 E7 F7 G7 H7
A6 B6 C6 D6 E6 F6 G6 H6
A5 B5 C5 D5 E5 F5 G5 H5
A4 B4 C4 D4 E4 F4 G4 H4
A3 B3 C3 D3 E3 F3 G3 H3
A2 B2 C2 D2 E2 F2 G2 H2
A1 B1 C1 D1 E1 F1 G1 H1
```

```python
# chessboard

letters = "ABCDEFGH"

for char in letters:
    for n in range(8, 0, -1):
        # print A8, A7, etc...
        print(char + str(n), end=" ")
    print() # new line after each row
```

Checkerboard – Challenge



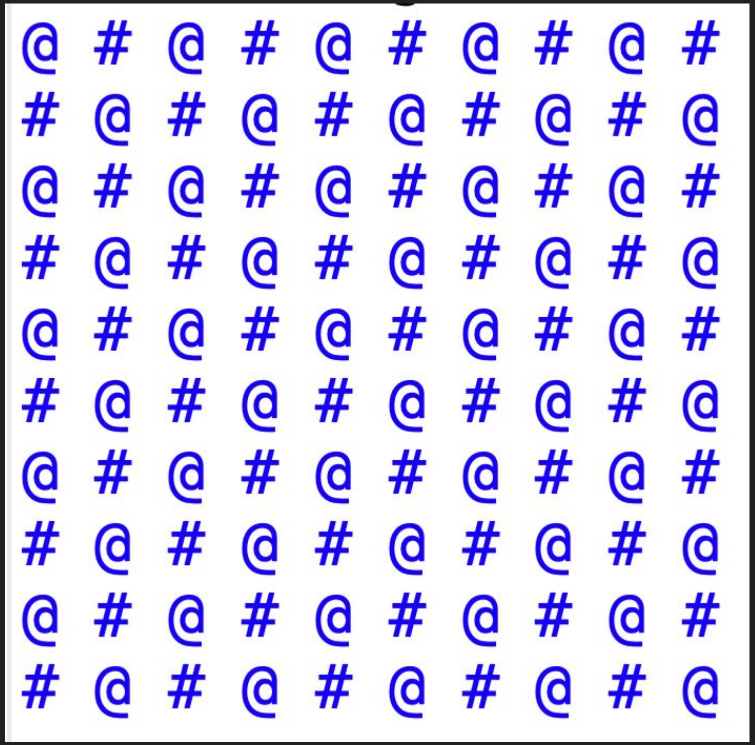
```
@ # @ # @ # @ # @ #
# @ # @ # @ # @ # @
@ # @ # @ # @ # @ #
# @ # @ # @ # @ # @
@ # @ # @ # @ # @ #
# @ # @ # @ # @ # @
@ # @ # @ # @ # @ #
# @ # @ # @ # @ # @
@ # @ # @ # @ # @ #
# @ # @ # @ # @ # @
```

Make a 10x10 checkerboard grid with alternating symbols.

Careful: Does your code work if you want to make an odd# x odd# grid?

Hint: Is there a relationship between the row and column numbers and what symbol is drawn?

Checkerboard – Thought Process

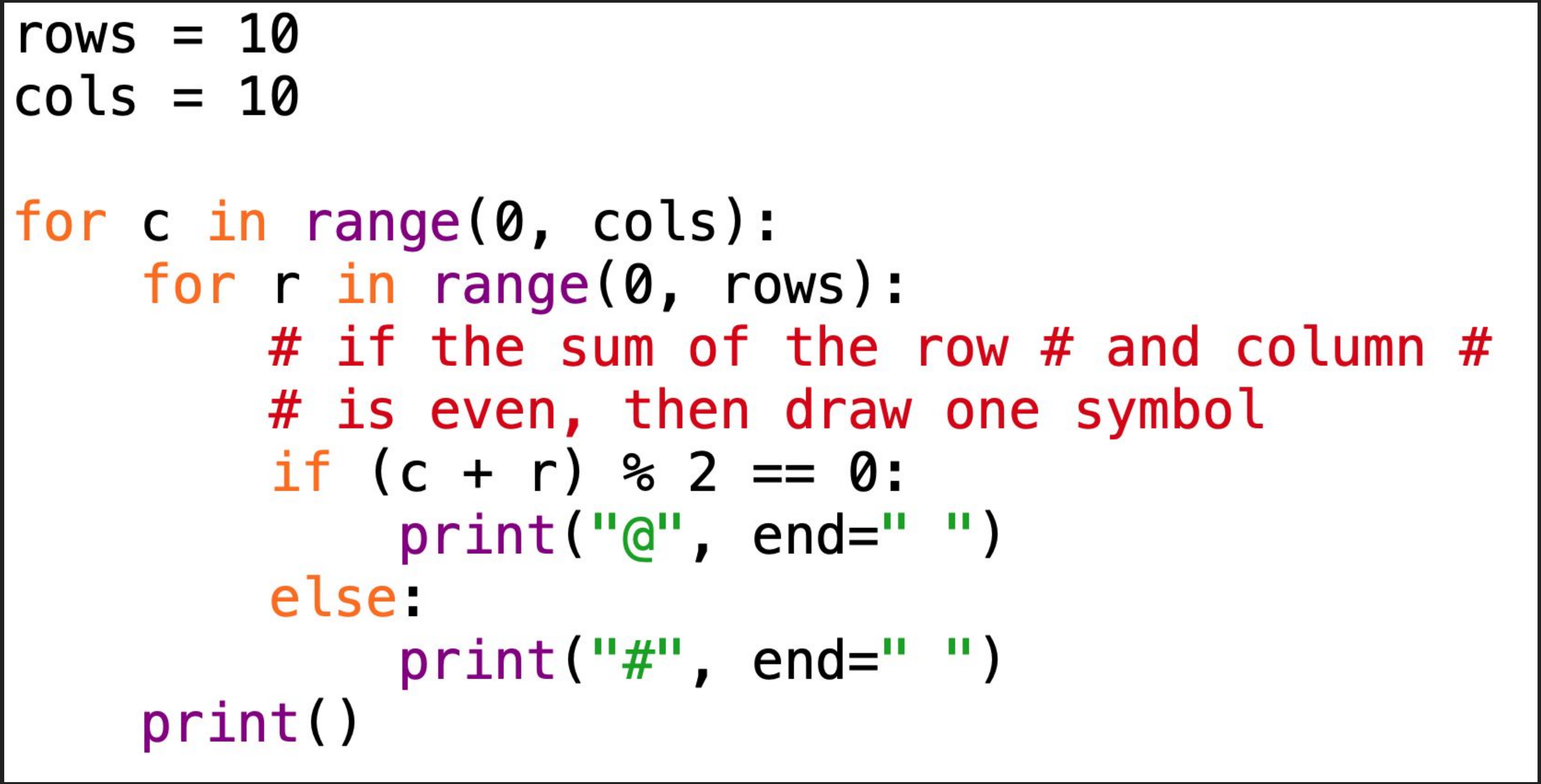


```
@ # @ # @ # @ # @ #
# @ # @ # @ # @ # @
@ # @ # @ # @ # @ #
# @ # @ # @ # @ # @
@ # @ # @ # @ # @ #
# @ # @ # @ # @ # @
@ # @ # @ # @ # @ #
# @ # @ # @ # @ # @
@ # @ # @ # @ # @ #
# @ # @ # @ # @ # @
```

What’s the pattern?

@:
row 0: col 0, col 2, col 4, col 6…
row 1: col 1, col 3, col 5, col 7…

When row# and col# are both even or when row# and col# are both odd

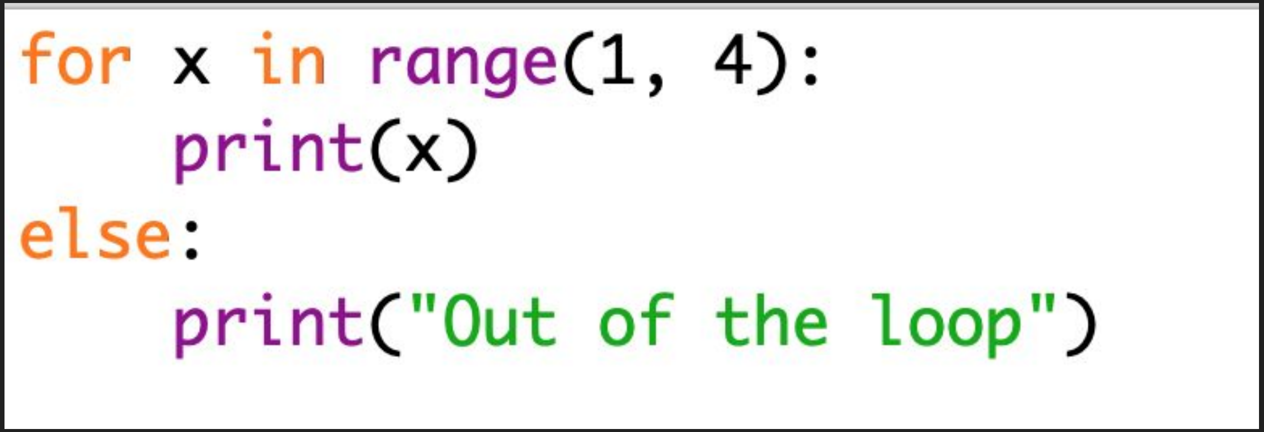


```python
rows = 10
cols = 10

for c in range(0, cols):
    for r in range(0, rows):
        # if the sum of the row # and column #
        # is even, then draw one symbol
        if (c + r) % 2 == 0:
            print("@", end=" ")
        else:
            print("#", end=" ")
    print()
```

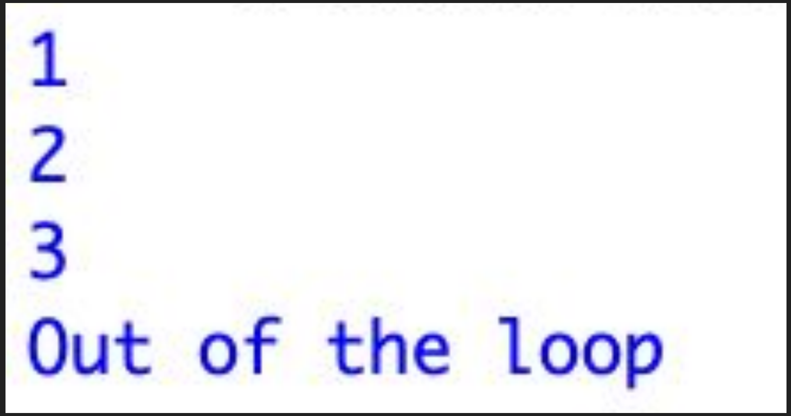
`for / else`

for / else

- **for** loops also have an **else** clause
- The else clause executes after the loop completes normally.
- This means that the loop did not encounter a break statement.

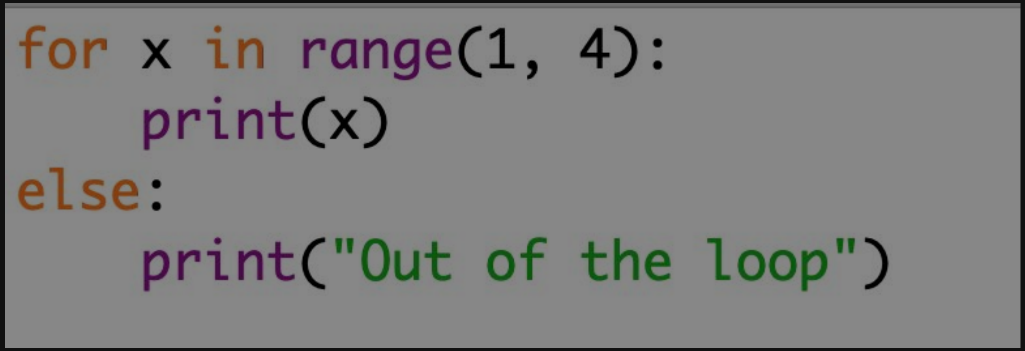


```python
for x in range(1, 4):
    print(x)
else:
    print("Out of the loop")
```

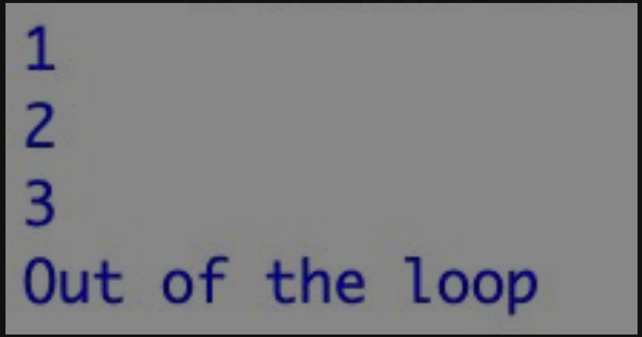


```
1
2
3
Out of the loop
```

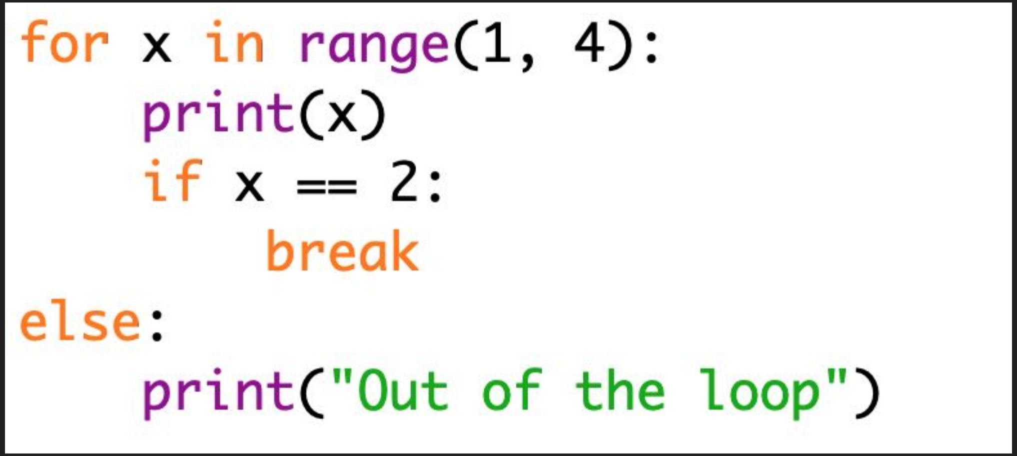
for / else



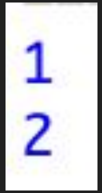
```python
for x in range(1, 4):
    print(x)
else:
    print("Out of the loop")
```



```
1
2
3
Out of the loop
```

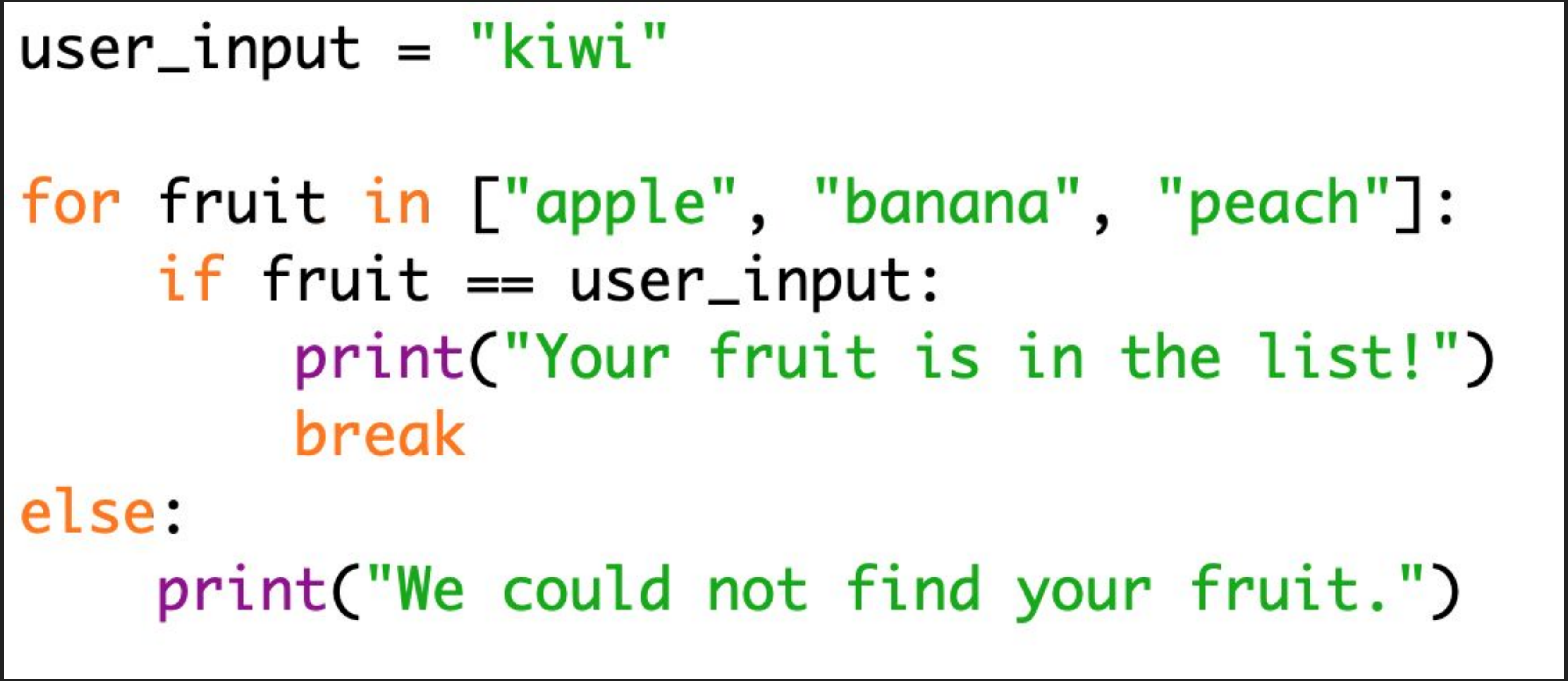


```python
for x in range(1, 4):
    print(x)
    if x == 2:
        break
else:
    print("Out of the loop")
```

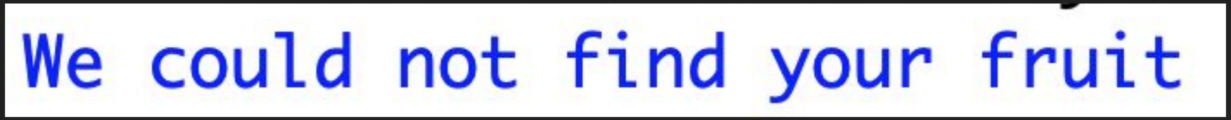


```
1
2
```

for / else



```python
user_input = "kiwi"

for fruit in ["apple", "banana", "peach"]:
    if fruit == user_input:
        print("Your fruit is in the list!")
        break
else:
    print("We could not find your fruit.")
```



```
We could not find your fruit
```

Can you rewrite your prime number finder using for/else?

```python
# Program to check if a number is prime or not

num = 407

# To take input from the user
#num = int(input("Enter a number: "))

# prime numbers are greater than 1
if num > 1:
   # check for factors
   for i in range(2,num):
       if (num % i) == 0:
           print(num,"is not a prime number")
           print(i,"times",num//i,"is",num)
           break
   else:
       print(num,"is a prime number")
```

Functions

Functions

- A function is a group of statements that exist within a program for the purpose of performing a specific task
- Since the beginning of the semester we have been using a number of Python’s built-in functions, including:
  - print()
  - range()
  - len()
  - random.randint()
  - ... etc

3 reasons to use functions

1. Organize your code
2. Reuse your code
3. Collaborate with others

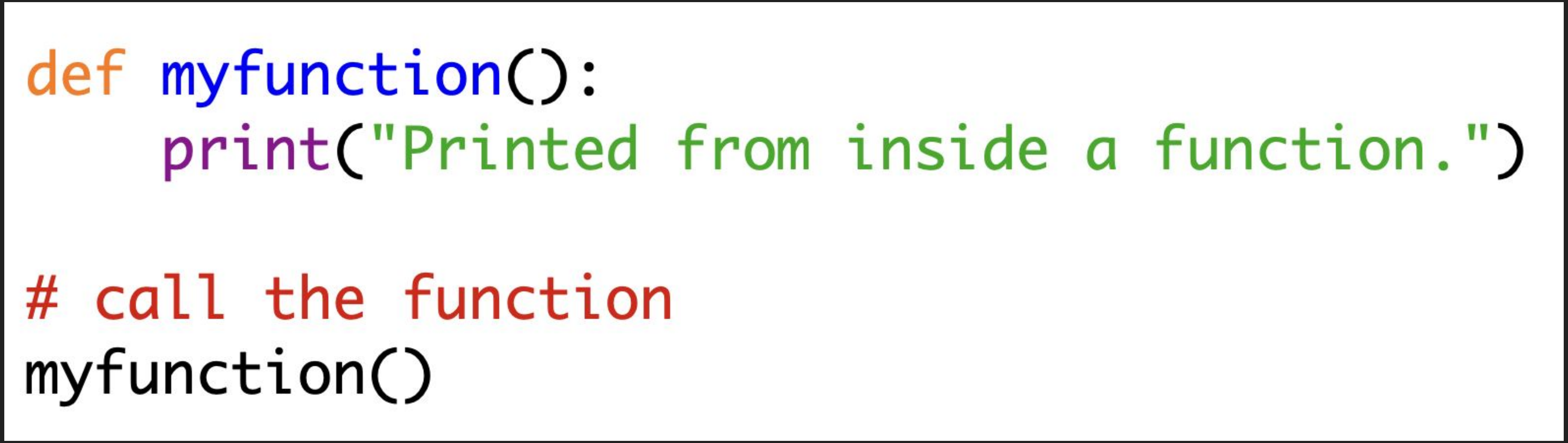
Defining Functions

Functions, like variables must be named and created before you can use them

The same naming rules apply for both variables and functions

- You can’t use any of Python’s keywords
- No spaces
- The first character must be A-Z or a-z or the “_” character
- After the first character you can use A-Z, a-z, “_” or 0-9
- Uppercase and lowercase characters are distinct

Defining functions



```
def myfunction():
    print("Printed from inside a function.")

# call the function
myfunction()
```

```
> Printed from inside a function
```

Some notes on functions

- When you run a function you say that you “call” it

- Once a function has completed, Python will return back to the line directly after the initial function call

- Functions must be defined before they can be used. In Python we generally place all of our functions at the beginning of our programs.

Flow of Execution with Functions

Flow of Execution

Code

```
def hello():
   print(“Hi there!”)
   print(“I’m a function!”)

print(“Good afternoon”)
print(“Welcome to class”)

hello()

print(“And now we’re done”)
```

Output

Flow of Execution

Code

Output

```
def hello():
    print(“Hi there!”)
    print(“I’m a function!”)

print(“Good afternoon”)
print(“Welcome to class”)

hello()

print(“And now we’re done”)
```

Flow of Execution

Code

```
def hello():
    print(“Hi there!”)
    print(“I’m a function!”)

print(“Good afternoon”)
print(“Welcome to class”)

hello()

print(“And now we’re done”)
```

Output

```
Good afternoon
```

Flow of Execution

Code

```
def hello():
    print(“Hi there!”)
    print(“I’m a function!”)

print(“Good afternoon”)
print(“Welcome to class”)

hello()

print(“And now we’re done”)
```

Output

```
Good afternoon
Welcome to class
```

Flow of Execution

Code

```
def hello():
    print(“Hi there!”)
    print(“I’m a function!”)

print(“Good afternoon”)
print(“Welcome to class”)

hello()

print(“And now we’re done”)
```

Output

```
Good afternoon
Welcome to class
```

Flow of Execution

Code

```
def hello():
    print(“Hi there!”)
    print(“I’m a function!”)

print(“Good afternoon”)
print(“Welcome to class”)

hello()

print(“And now we’re done”)
```

Output

```
Good afternoon
Welcome to class
```

Flow of Execution

Code

```
def hello():
    print(“Hi there!”)
    print(“I’m a function!”)

print(“Good afternoon”)
print(“Welcome to class”)

hello()

print(“And now we’re done”)
```

Output

```
Good afternoon
Welcome to class
Hi there!
```

Flow of Execution

Code

```
def hello():
    print(“Hi there!”)
    print(“I’m a function!”)

print(“Good afternoon”)
print(“Welcome to class”)

hello()

print(“And now we’re done”)
```

Output

```
Good afternoon
Welcome to class
Hi there!
I’m a function!
```

Flow of Execution

Code

```
def hello():
    print(“Hi there!”)
    print(“I’m a function!”)

print(“Good afternoon”)
print(“Welcome to class”)

hello()

print(“And now we’re done”)
```

Output

```
Good afternoon
Welcome to class
Hi there!
I’m a function!
And now we’re done
```

Multiple Functions

Multiple functions

Code

```
def hello():
   print(“Hello there!”)

def goodbye():
   print(“See ya!”)

hello()
goodbye()
```

Output

Multiple functions

Code

```
def hello():
   print(“Hello there!”)

def goodbye():
   print(“See ya!”)

hello()
goodbye()
```

Output

```
Hello there!
See ya!
```

Multiple functions

Code

```
def _message():
   print(“The password is ‘foo’”)

def main():
   print(“I have a message for you”)
   _message()
   print(“Goodbye!”)

main()
```

Output

Multiple functions

Code

```
def _message():
   print(“The password is ‘foo’”)

def main():
   print(“I have a message for you”)
   _message()
   print(“Goodbye!”)

main()
```

Output

```
I have a message for you
The password is ‘foo’
Goodbye!
```

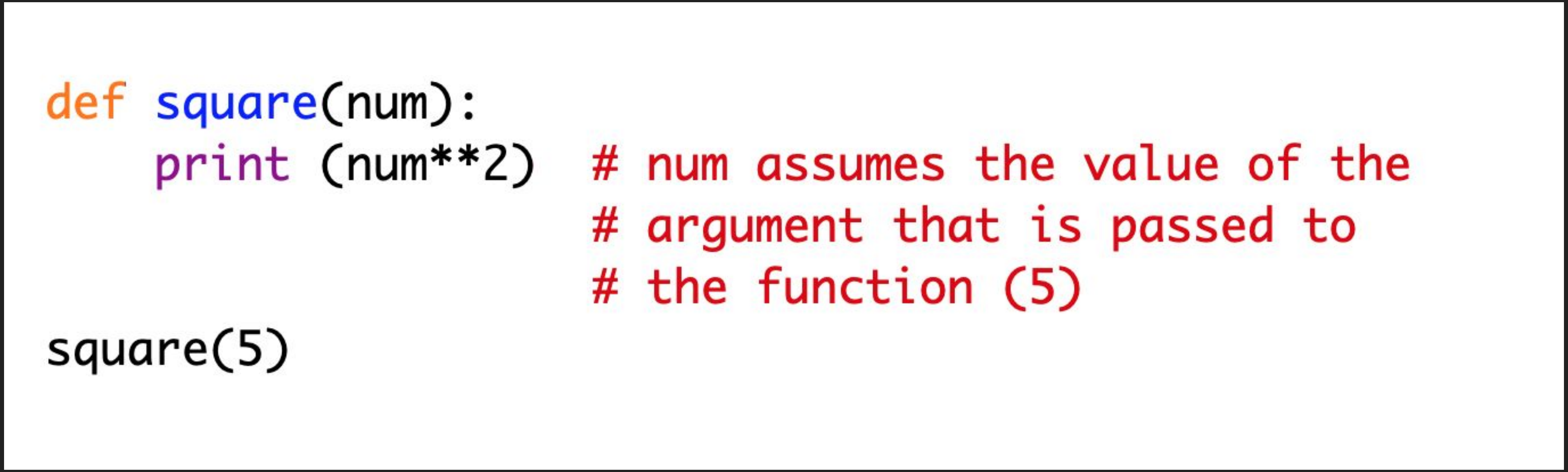
Passing Arguments to a Function

Passing Arguments to a Function

- Sometimes it’s useful to not only call a function but also send it one or more pieces of data as an argument
- This process is identical to what you’ve been doing with the built-in functions we have studied so far

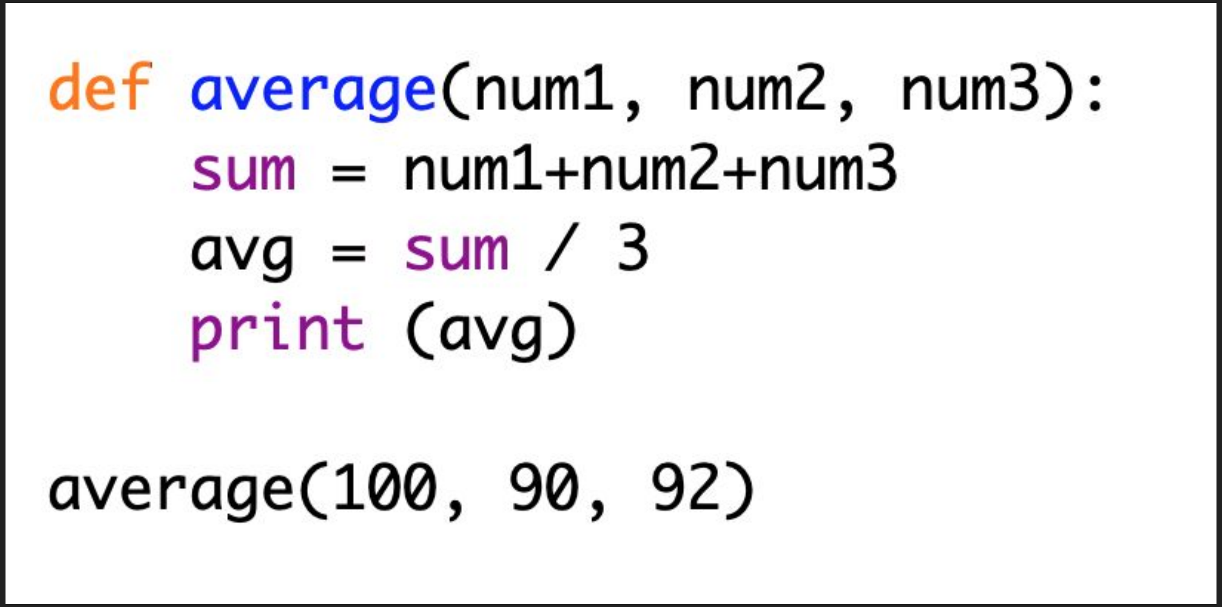
```
x = random.randint(1,5)    # send 2 integers
y = len('Emily')           # send 1 string
```

Passing Arguments to a Function



```python
def square(num):
    print (num**2)  # num assumes the value of the
                    # argument that is passed to
                    # the function (5)
square(5)
```

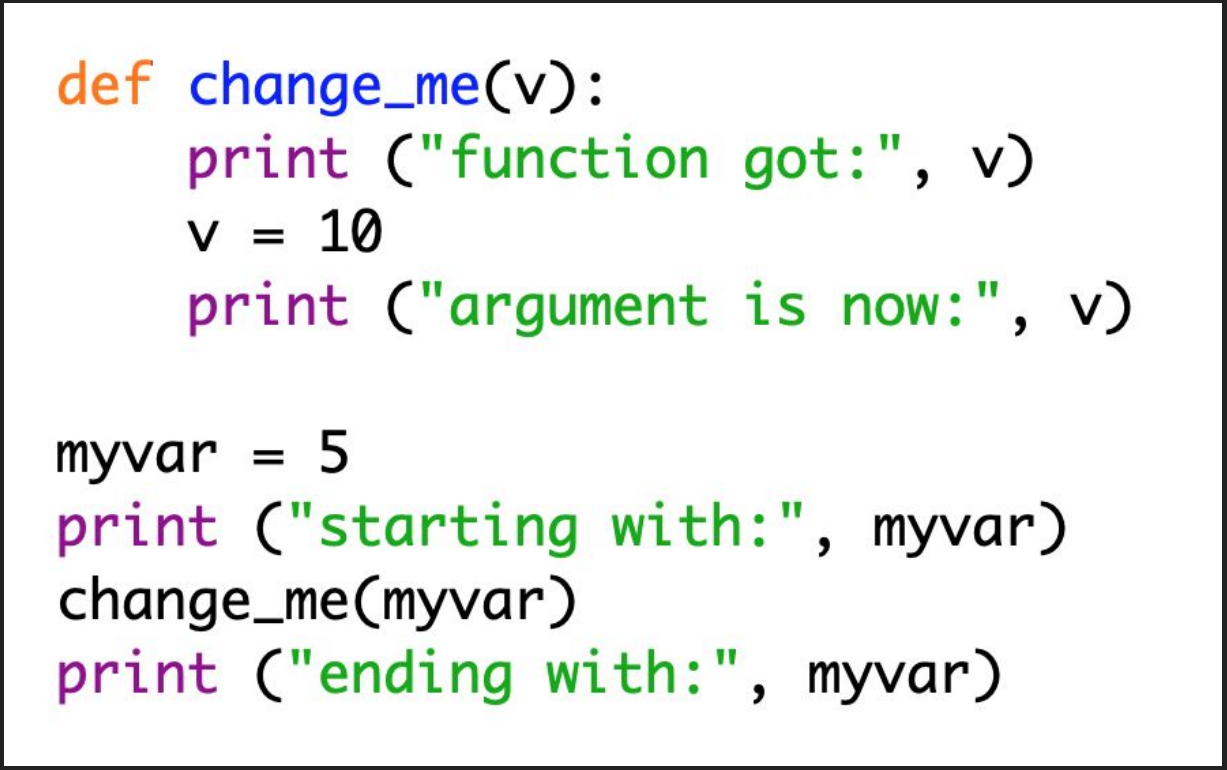
Passing Multiple Arguments to a Function



```python
def average(num1, num2, num3):
    sum = num1+num2+num3
    avg = sum / 3
    print (avg)

average(100, 90, 92)
```

Argument Mechanics

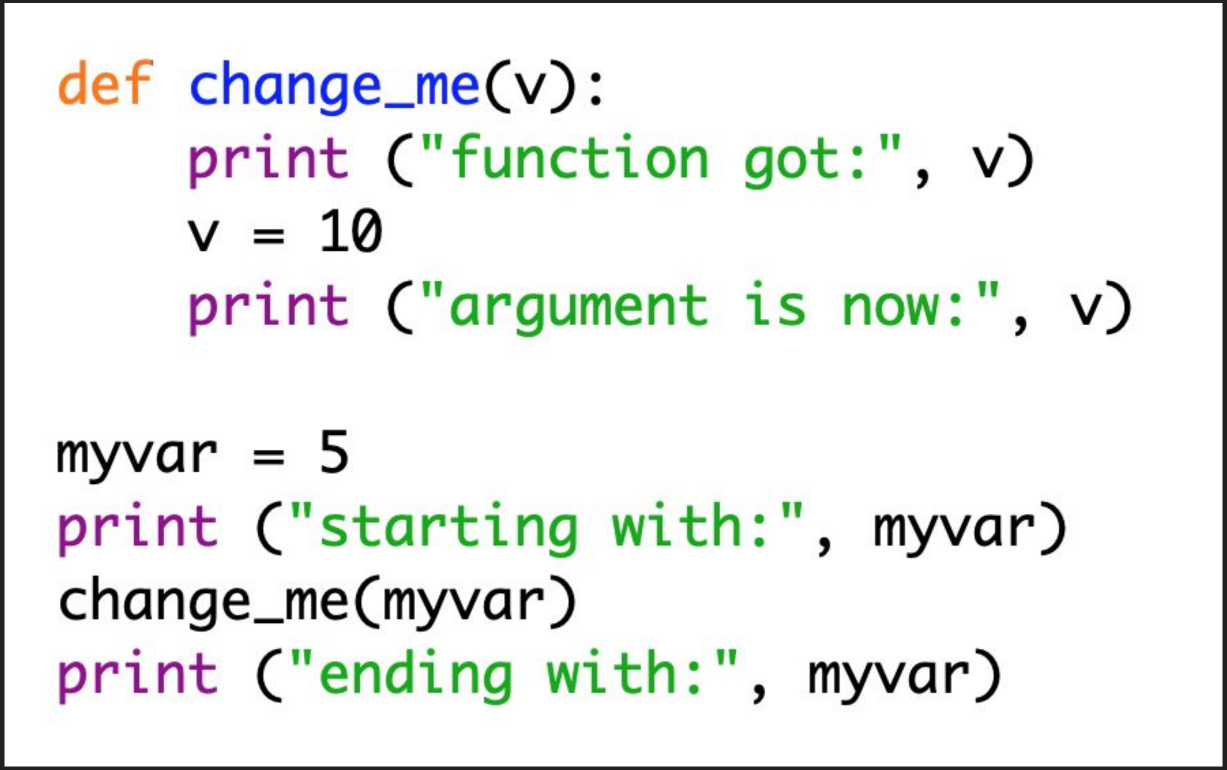
When we pass an argument to a function in Python we are actually passing it’s “value” into the function, and not an actual variable



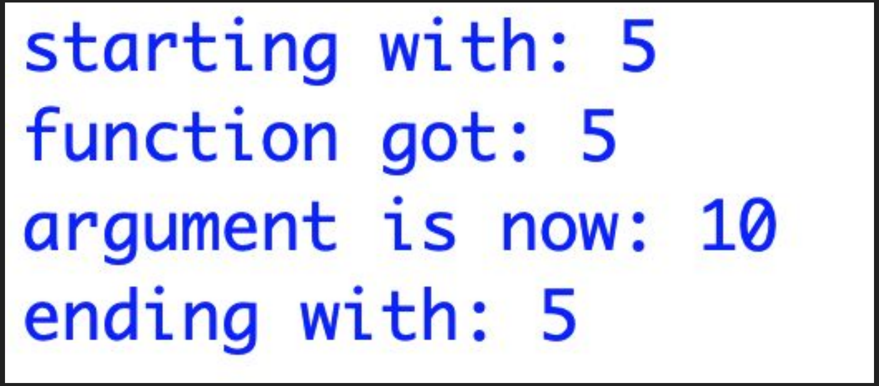
```
def change_me(v):
    print ("function got:", v)
    v = 10
    print ("argument is now:", v)

myvar = 5
print ("starting with:", myvar)
change_me(myvar)
print ("ending with:", myvar)
```

Argument Mechanics

When we pass an argument to a function in Python we are actually passing it’s “value” into the function, and not an actual variable

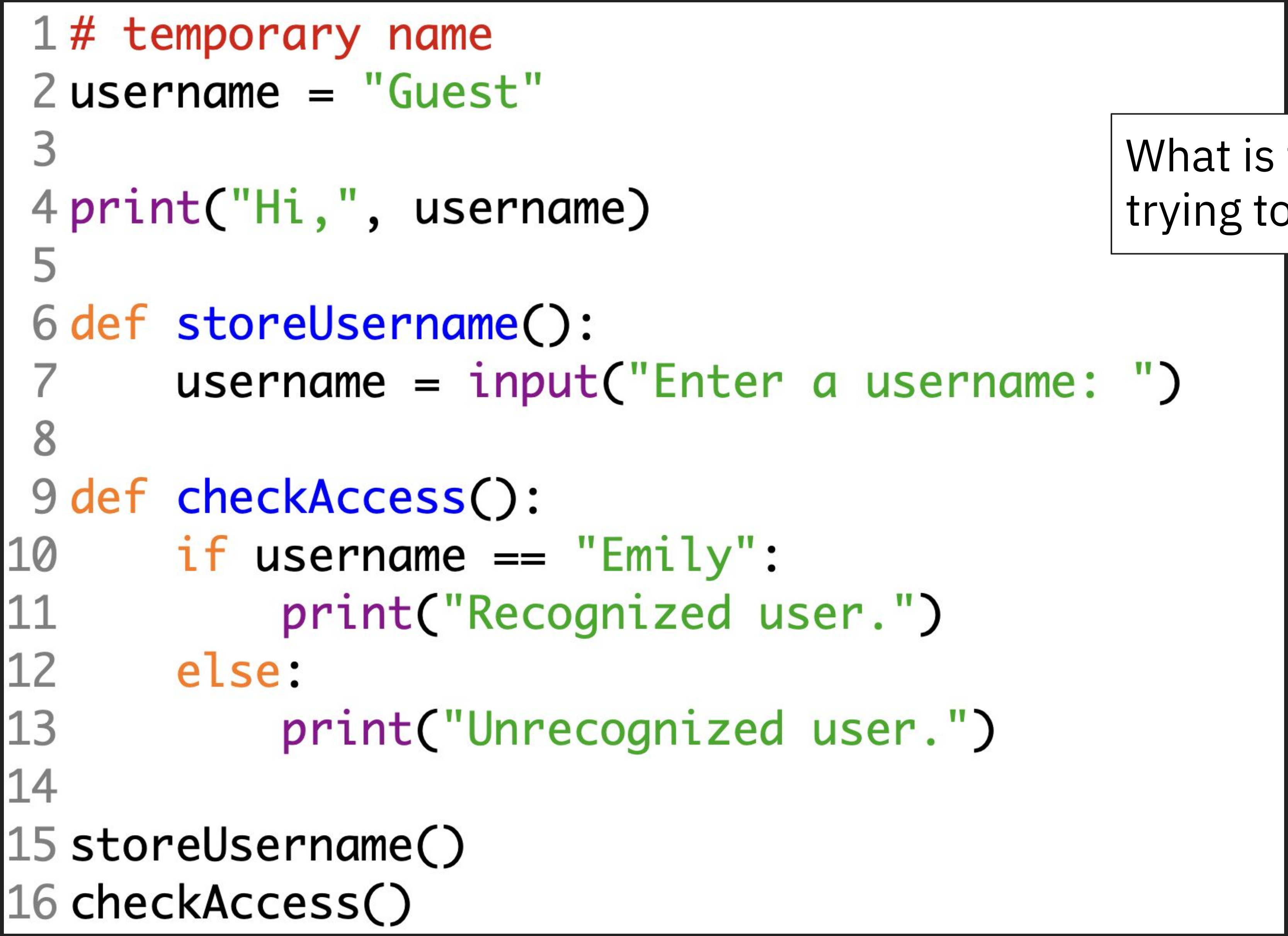


```
def change_me(v):
    print ("function got:", v)
    v = 10
    print ("argument is now:", v)

myvar = 5
print ("starting with:", myvar)
change_me(myvar)
print ("ending with:", myvar)
```



```
starting with: 5
function got: 5
argument is now: 10
ending with: 5
```

Local vs Global Variables

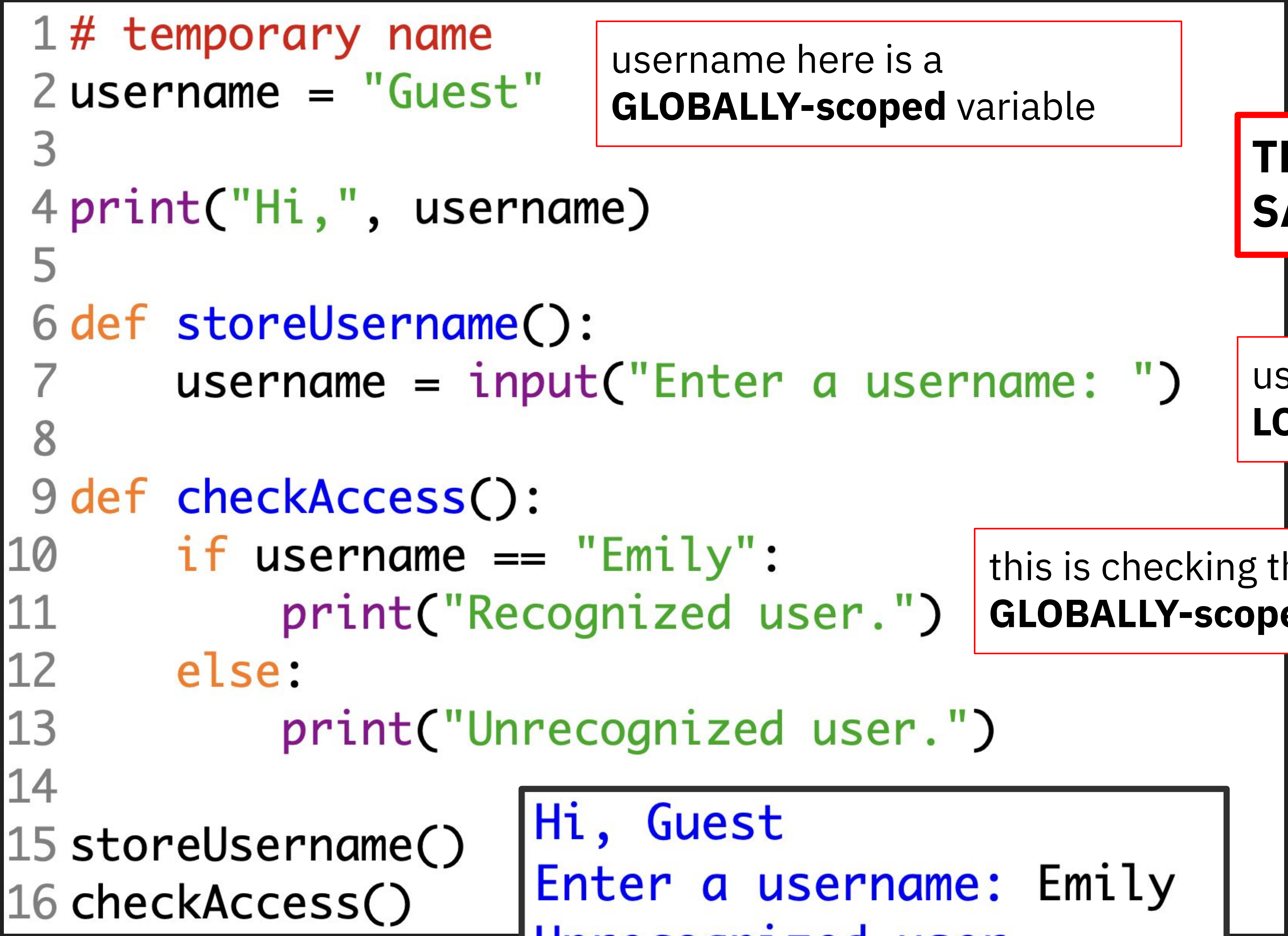


```python
# temporary name
username = "Guest"

print("Hi,", username)

def storeUsername():
    username = input("Enter a username: ")

def checkAccess():
    if username == "Emily":
        print("Recognized user.")
    else:
        print("Unrecognized user.")

storeUsername()
checkAccess()
```

What is this code trying to accomplish?

```python
# temporary name
username = "Guest"

print("Hi,", username)

def storeUsername():
    username = input("Enter a username: ")

def checkAccess():
    if username == "Emily":
        print("Recognized user.")
    else:
        print("Unrecognized user.")

storeUsername()
checkAccess()
```

```
Hi, Guest
Enter a username: Emily
Unrecognized user.
```



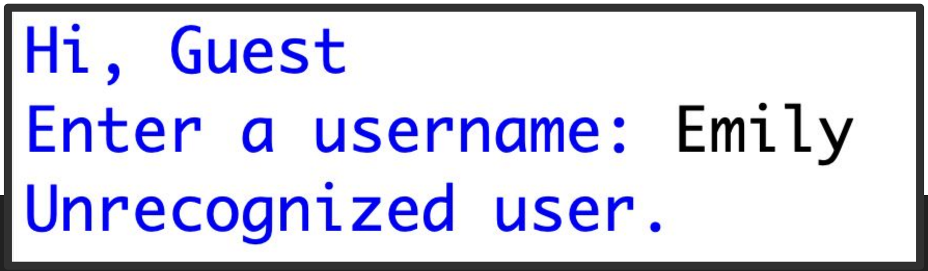
```python
# temporary name
username = "Guest"

print("Hi,", username)

def storeUsername():
    username = input("Enter a username:  ")

def checkAccess():
    if username == "Emily":
        print("Recognized user.")
    else:
        print("Unrecognized user.")

storeUsername()
checkAccess()
```

username here is a **GLOBALLY-scoped** variable

THEY ARE NOT THE SAME!

username here is a **LOCALLY-scoped** variable

this is checking the **GLOBALLY-scoped** variable

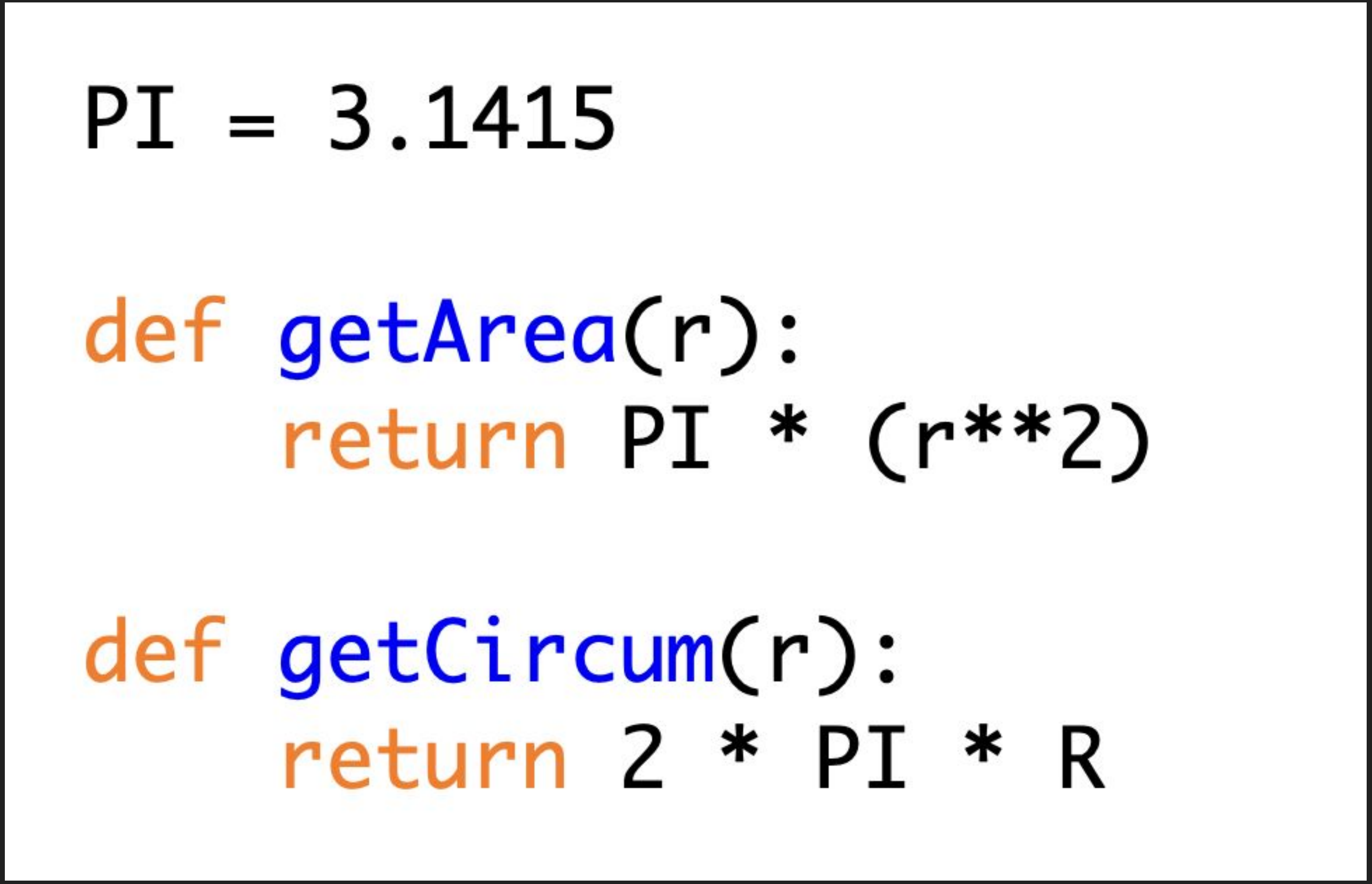


```
Hi, Guest
Enter a username: Emily
Unrecognized user.
```

Local vs Global Variables: Best Practices

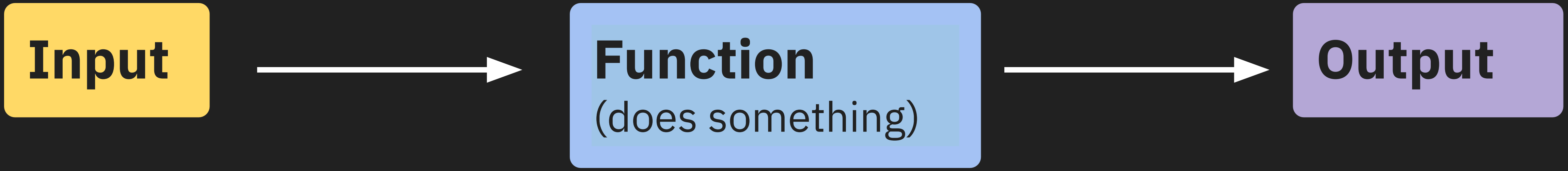
- Keep variables as local as possible
- It makes your code more readable and easier to debug
- The best use cases for global variables are constants (variables that rarely change but appear in multiple functions)

Local vs Global Variables: Best Practices

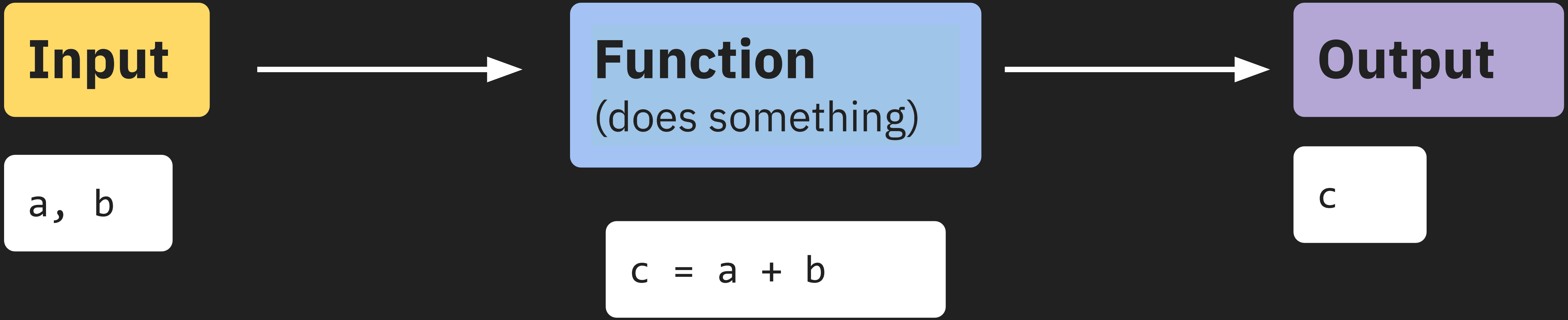


```
PI = 3.1415

def getArea(r):
    return PI * (r**2)

def getCircum(r):
    return 2 * PI * R
```

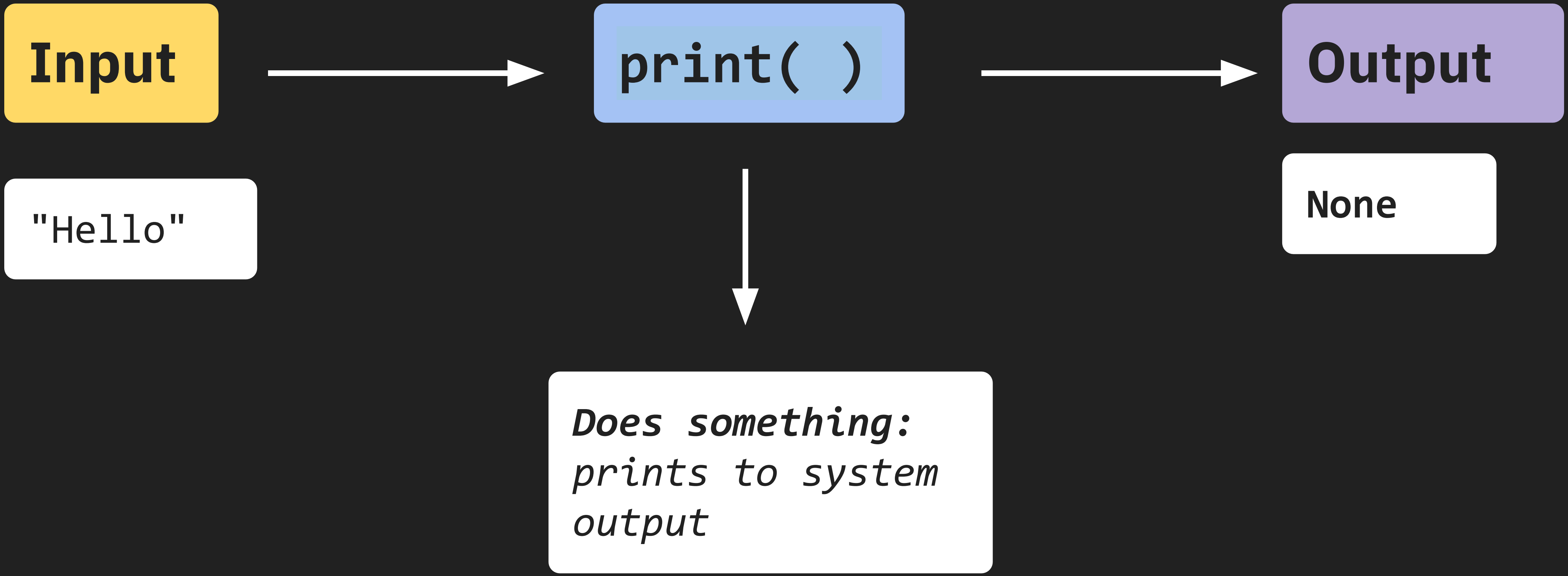
All functions return something


Input
Function
(does something)
Output

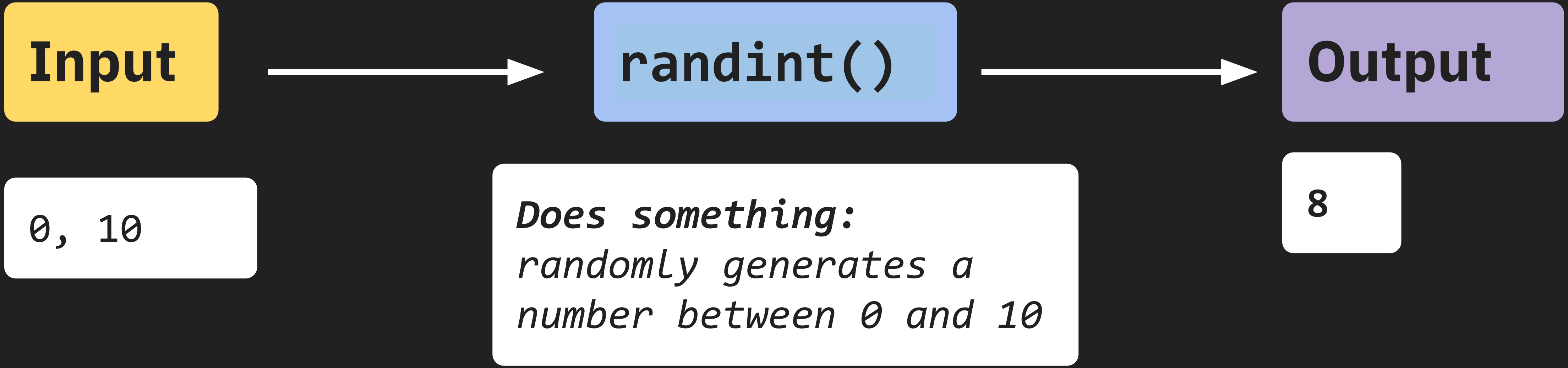
```
def add(a, b):
    c = a + b
    return c
```



```
print("Hello")
```

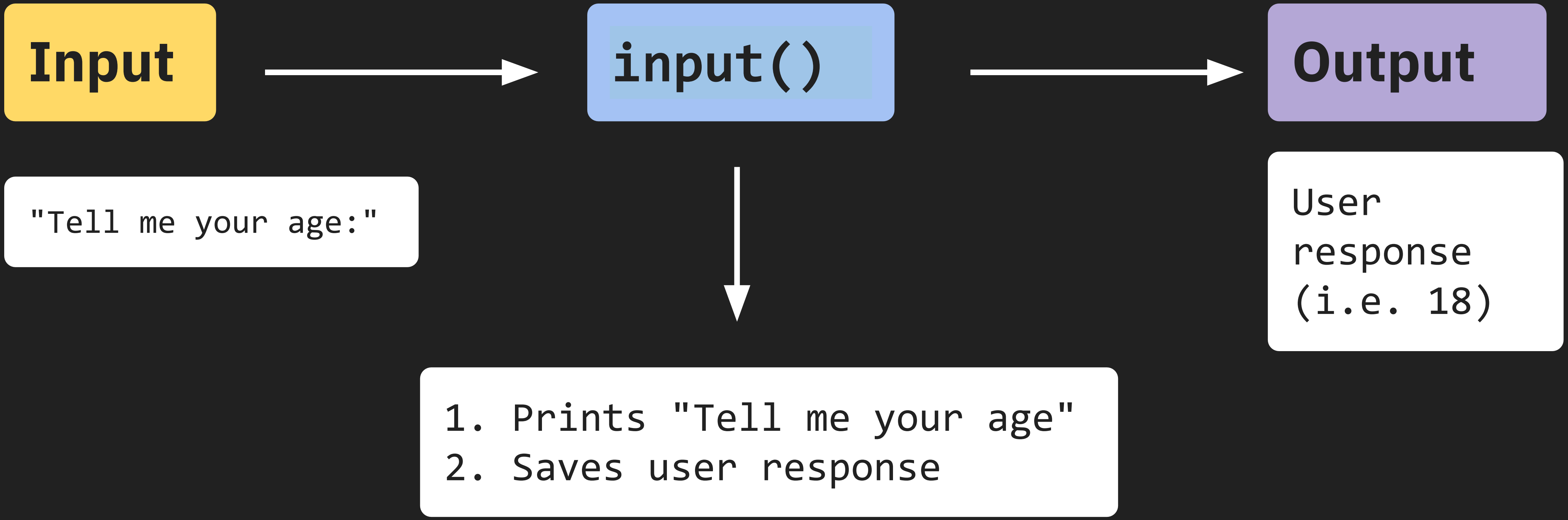

Input
"Hello"
print( )
Output
None
Does something:
prints to system output

```
random.randint(0,10)
```



Does something:
randomly generates a number between 0 and 10

```
input("Tell me your age: ")
```



"Tell me your age:"

1. Prints "Tell me your age"
2. Saves user response

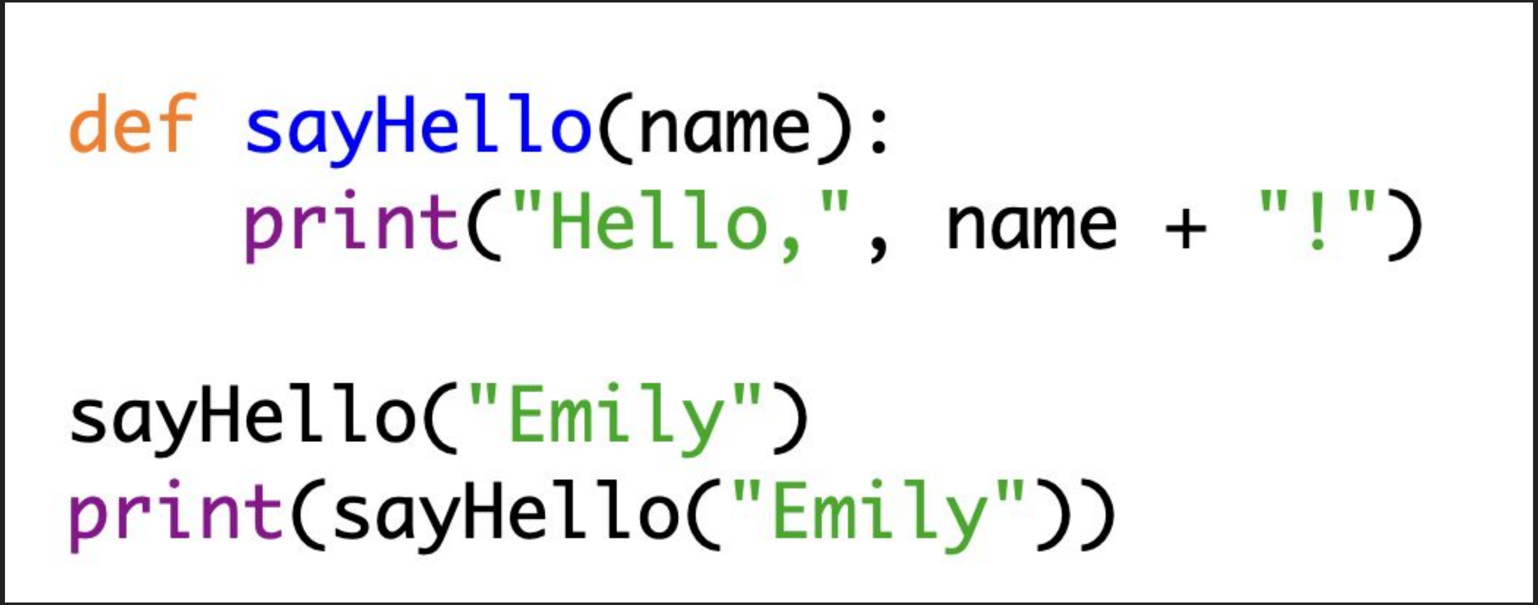
User response (i.e. 18)

```
def translateRight(x, y):
    x += 1
    return x, y

x_coord, y_coord = translateRight(1, 5)

# x_coord -> 2
# y_coord -> 5
```

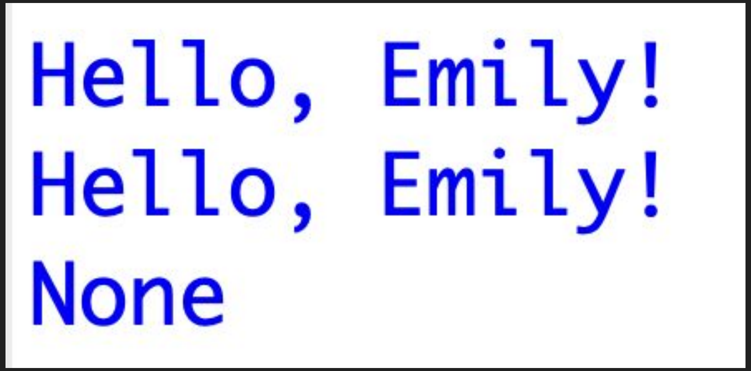
When you run a function, you usually want to capture the output.

You can do that by assigning what is returned to a variable, or multiple variables.

Value Returning Functions

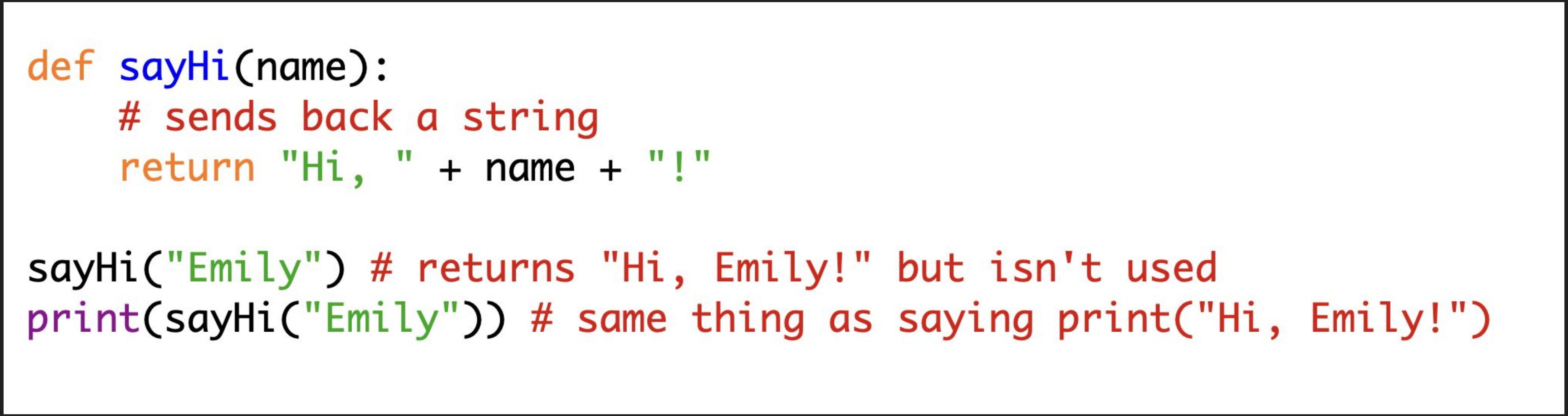


```
def sayHello(name):
    print("Hello,", name + "!")

sayHello("Emily")
print(sayHello("Emily"))
```



```
Hello, Emily!
Hello, Emily!
None
```

Value Returning Functions

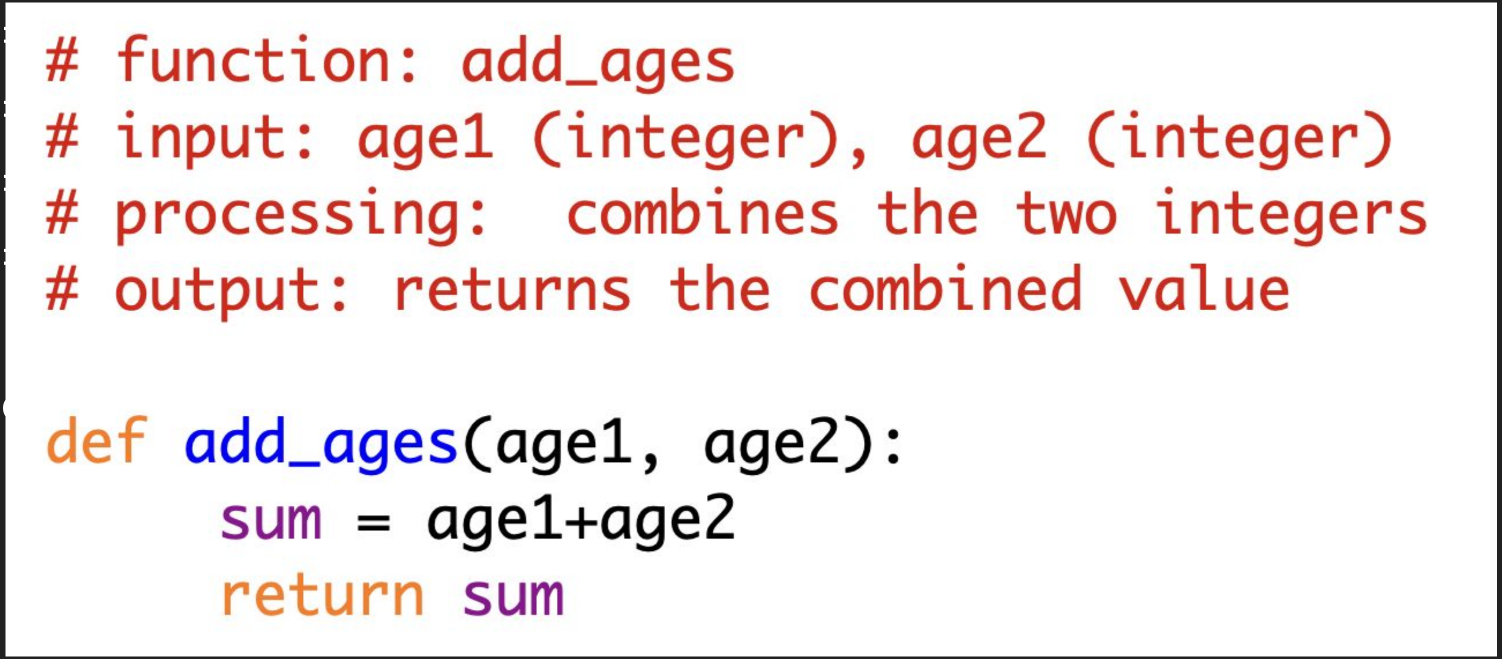


```python
def sayHi(name):
    # sends back a string
    return "Hi, " + name + "!"

sayHi("Emily") # returns "Hi, Emily!" but isn't used
print(sayHi("Emily")) # same thing as saying print("Hi, Emily!")
```

IPO Notation

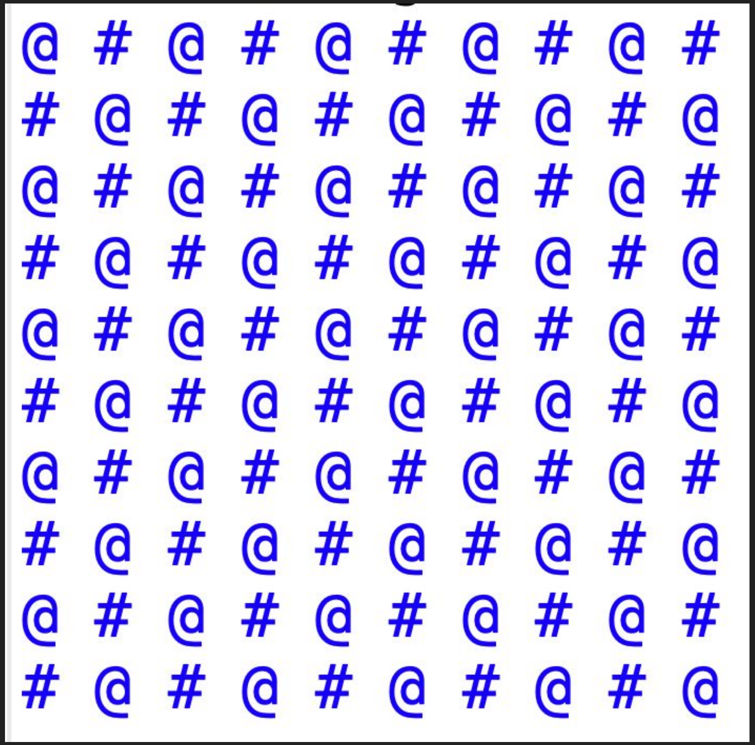
- As you start writing more advanced functions you should think about documenting them based on their Input, Processing and Output (IPO)

- Example:



```
# function: add_ages
# input: age1 (integer), age2 (integer)
# processing:  combines the two integers
# output: returns the combined value

def add_ages(age1, age2):
        sum = age1+age2
        return sum
```

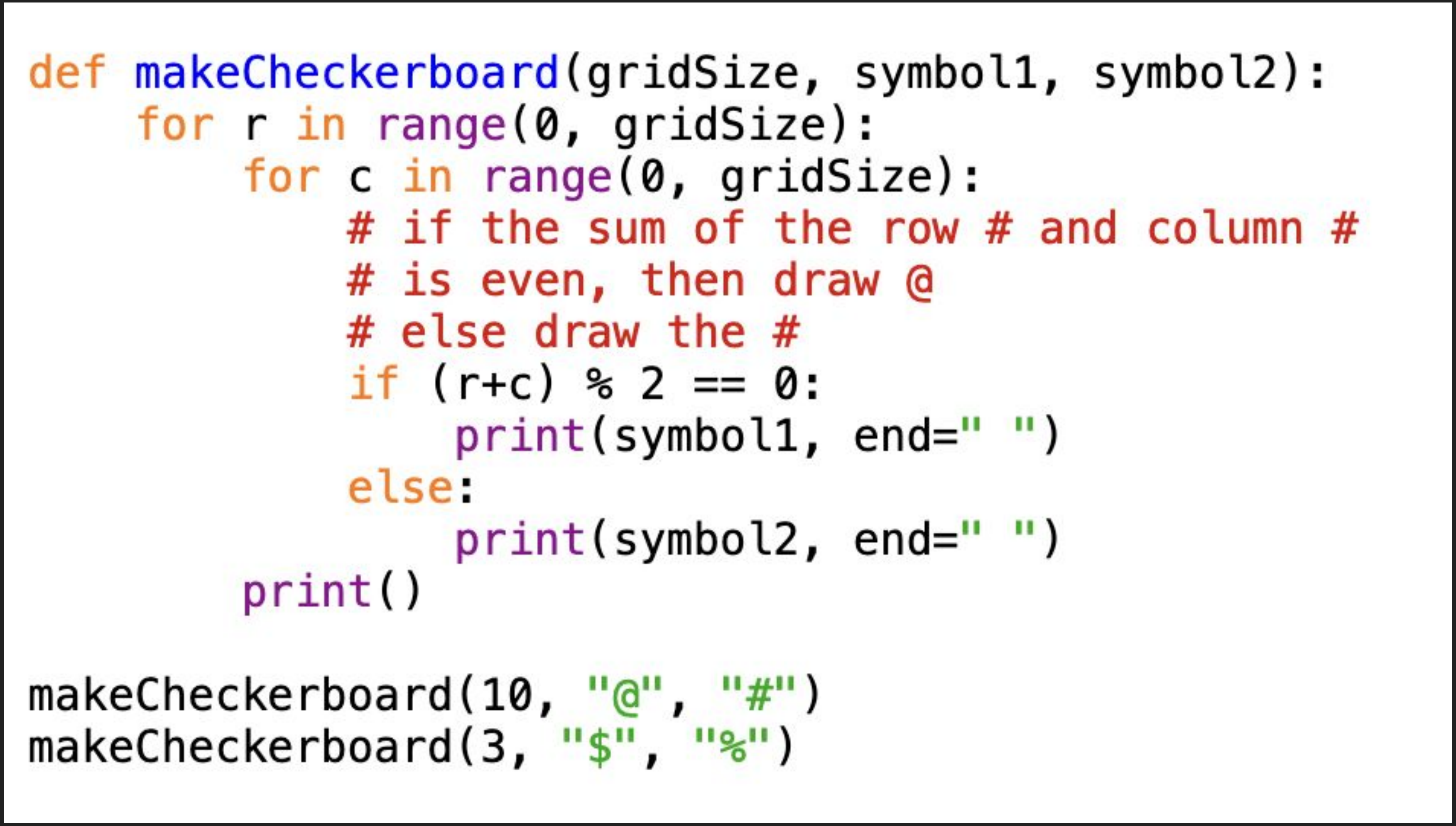
Programming Challenge



```
@ # @ # @ # @ # @ #
# @ # @ # @ # @ # @
@ # @ # @ # @ # @ #
# @ # @ # @ # @ # @
@ # @ # @ # @ # @ #
# @ # @ # @ # @ # @
@ # @ # @ # @ # @ #
# @ # @ # @ # @ # @
@ # @ # @ # @ # @ #
# @ # @ # @ # @ # @
```

Convert our earlier checkerboard code into a function that accepts three parameters – grid size, first character, second character

Solution 1: the function returns nothing and just prints to Shell



```python
def makeCheckerboard(gridSize, symbol1, symbol2):
    for r in range(0, gridSize):
        for c in range(0, gridSize):
            # if the sum of the row # and column #
            # is even, then draw @
            # else draw the #
            if (r+c) % 2 == 0:
                print(symbol1, end=" ")
            else:
                print(symbol2, end=" ")
        print()

makeCheckerboard(10, "@", "#")
makeCheckerboard(3, "$", "%")
```

Solution 2: the function returns an output string

```python
def makeCheckerboard(rows, cols, symbol1, symbol2):
    # create an output string
    output = ""
    for r in range(rows):
        for c in range(cols):
            if (c + r) % 2 == 0:
                output += symbol1 + " "
            else:
                output += symbol2 + " "
        output += "\n"
    return output


# call the function
print(makeCheckerboard(5, 6, "@", "#"))
print(makeCheckerboard(2, 10, ":)", ":("))
```

I prefer this solution!

I like it when functions return things :)

Homework

— Assignment #5 (due next class)