*

CSCI-UA-0002

Intro to Computer Programming (No Prior Experience)

Module 6: Functions

Professor Emily Zhao

Section 008	Section 012
T/R 12:30-1:45PM	T/R 4:55-6:10PM

*

Agenda

- Review Schedule
- Midterm Format
- Continue Nested Loops
- Review Ed Questions
- Module 6 Review Part 1

Midterm

Midterm

Date: Thursday, October 26th

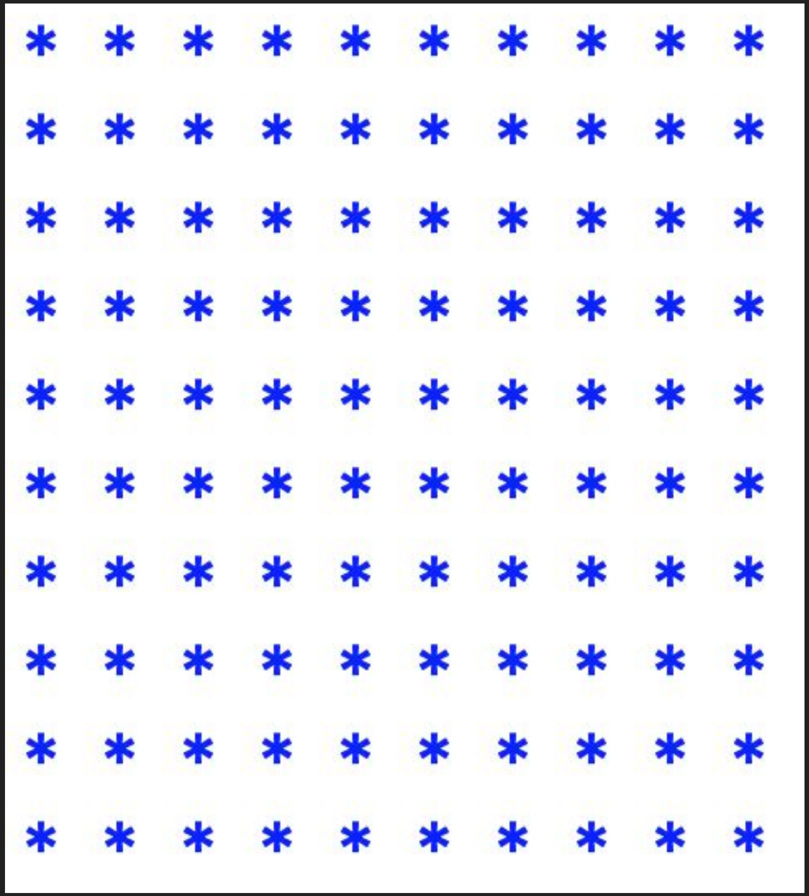
Format: Brightspace Exam

Topics Covered: Modules 1–6

- Brightspace exam w/ Lockdown Browser
- Open note (bring in offline/paper resources)
- Multiple choice
- Fill in the blank/short answer
- Long(er) programming questions

“For” loop review

Grid of Asterisks



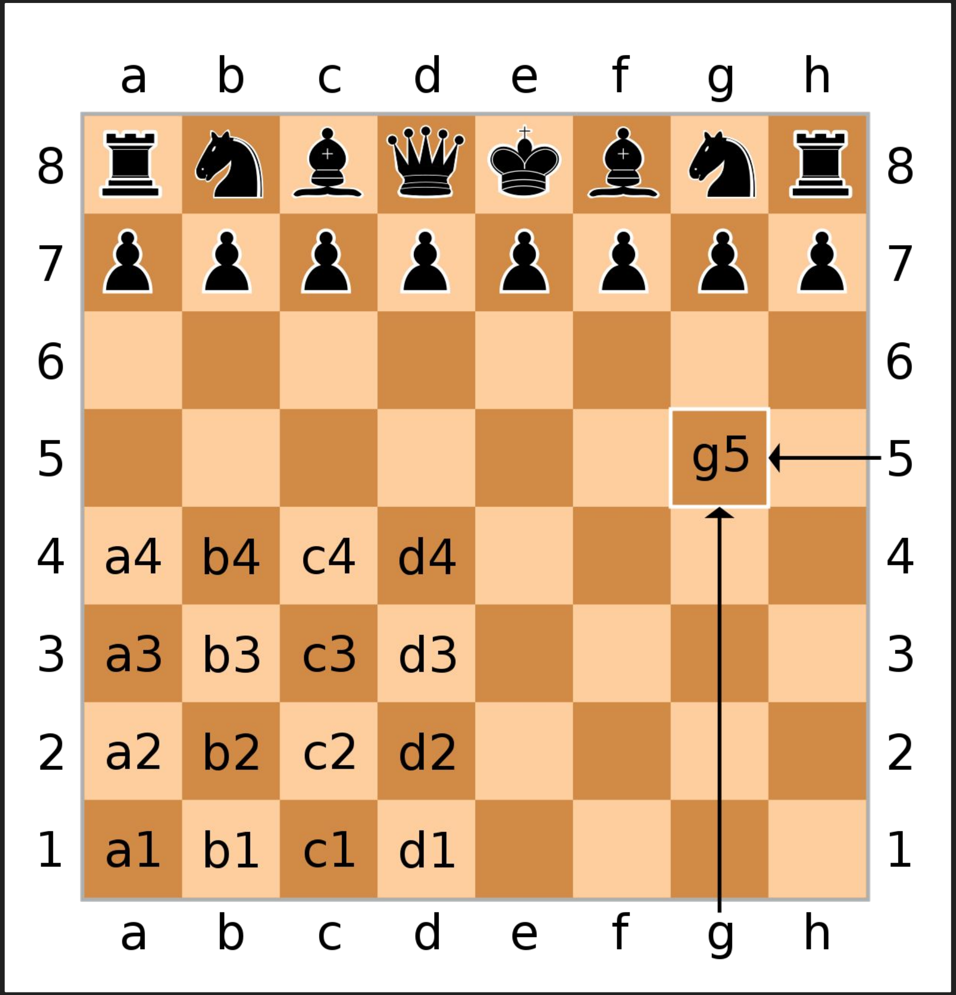
```
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
```

1. Generate a 10x10 grid of asterisks
   a. Try without loops
   b. Try using a while loop
   c. Try (1) for loop
   d. Try (2) for loops

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
```

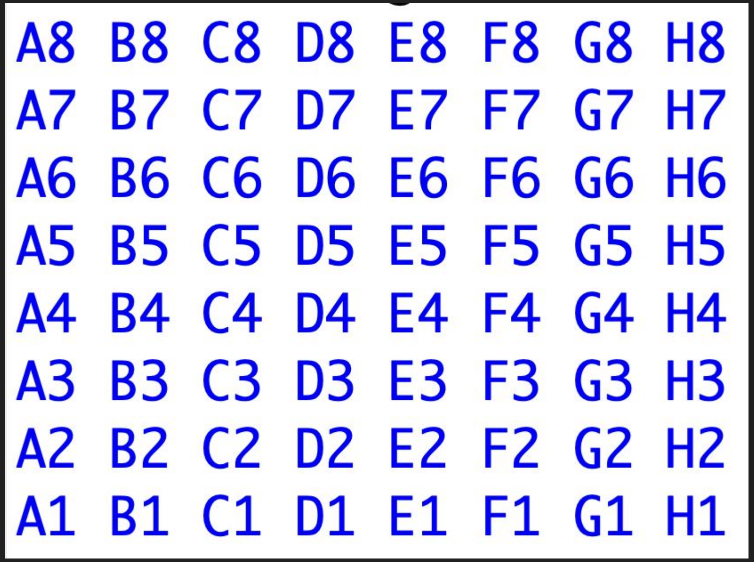
Chessboard



Generate a table of chess coordinates.

How many for loops do you need?

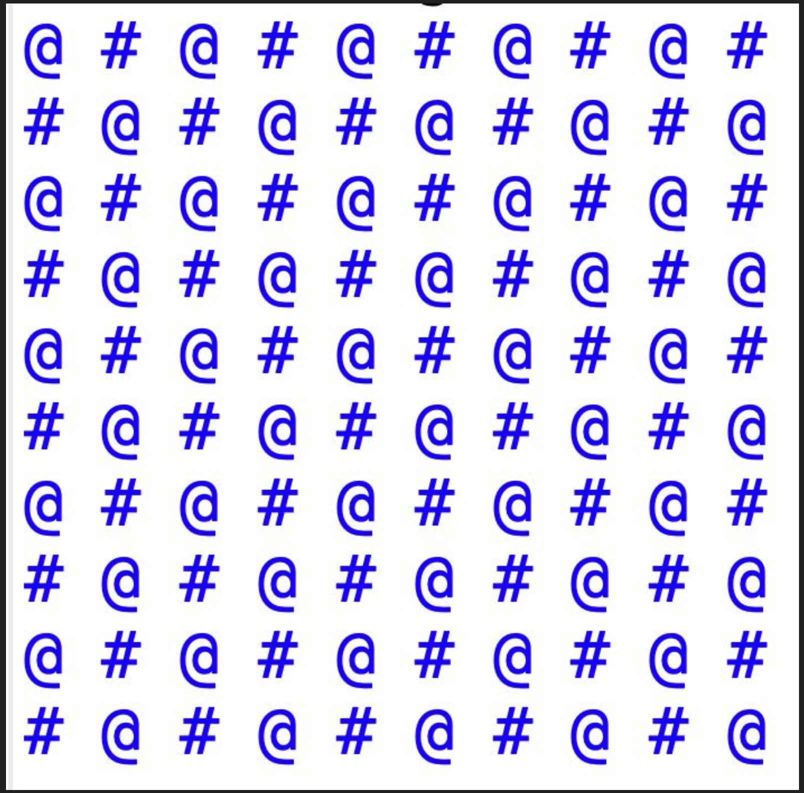
Expected Output:



```
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



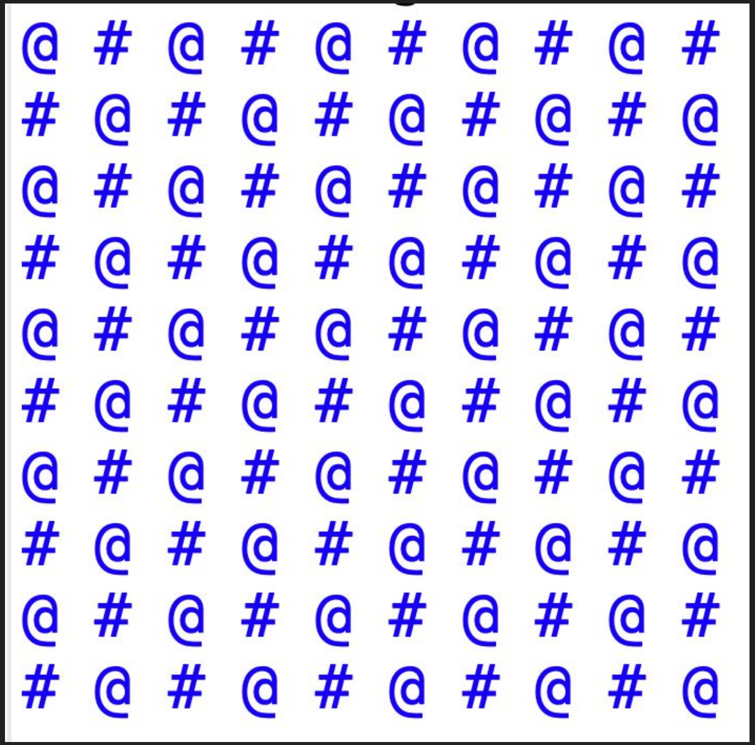
```
@ # @ # @ # @ # @ #
# @ # @ # @ # @ # @
@ # @ # @ # @ # @ #
# @ # @ # @ # @ # @
@ # @ # @ # @ # @ #
# @ # @ # @ # @ # @
@ # @ # @ # @ # @ #
# @ # @ # @ # @ # @
@ # @ # @ # @ # @ #
# @ # @ # @ # @ # @
```

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()
```

Module 6 – Functions

Module 6

- Basic User Defined Functions
- Function Arguments and Variables
- Function Return Values

Your Questions

- → What are ***arguments*** and ***parameters***?
    - → What does it mean to ***pass a value***?
- → What is `return`? How is it different from `break`? How does it differ from `print()`?
- → Quiz questions
- → What is a ***global*** variable?
    - → When do I use it?
    - → How does it differ from a ***local*** variable?

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

Functions

You can think of functions like verbs!

1) They DO things
2) They RETURN things *

The `print()` function

What it does: prints objects to the shell
What is returns: nothing

The `input()` function

What it does: asks the user for input with prompt
What is returns: the user input as a string *

*** If your function returns a value, you must store the value!**

```python
# Definining function
def add(a, b):      # a and b are "parameters"
    c = a + b       # DO: add two nums together
    return c        # RETURN: the sum of the nums

# Calling the function
result = add(3, 5)  # 3 and 5 are "arguments"
                    # they are "passed" to the function
                    # since add returns a value,
                    # we must store it in a variable
```

`return`

Used in **FUNCTIONS** only

- Indicates the end of a function’s execution
- Provides a result (value) to the caller
- Functions can only have **one** `return`

`break`

Used in **LOOPS** only

- Immediately terminates a loop’s execution
- The program continues with the next statement after the loop
- Loops can only have **one** `break`

Defining Functions

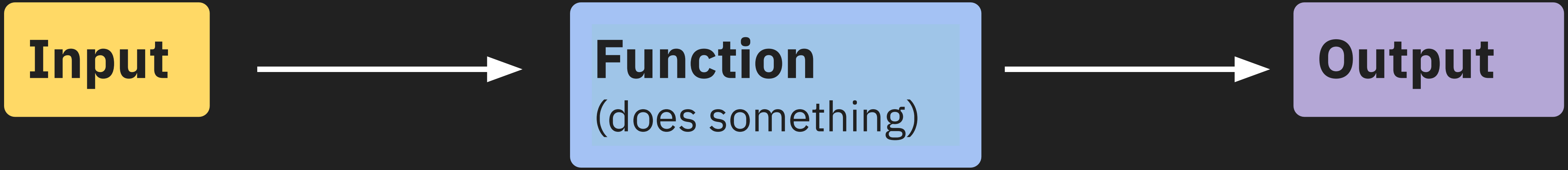
Functions, like variables must be named and created before you can use them

The same naming rules apply for both variables and functions

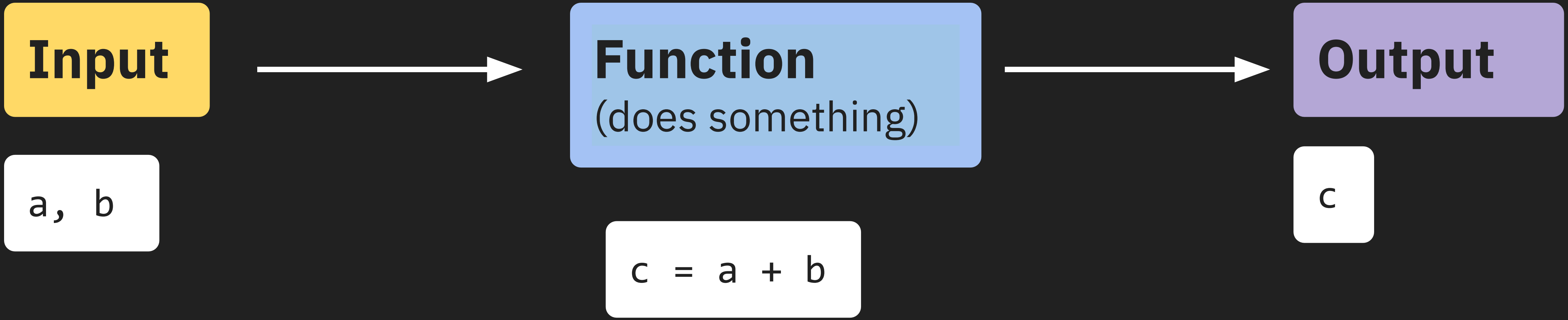
- You can’t use any of Python’s keywords
- No spaces
- The first character must be A-Z or a-z or the “_” character
- After the first character you can use A-Z, a-z, “_” or 0-9
- Uppercase and lowercase characters are distinct

All functions (should) return something!

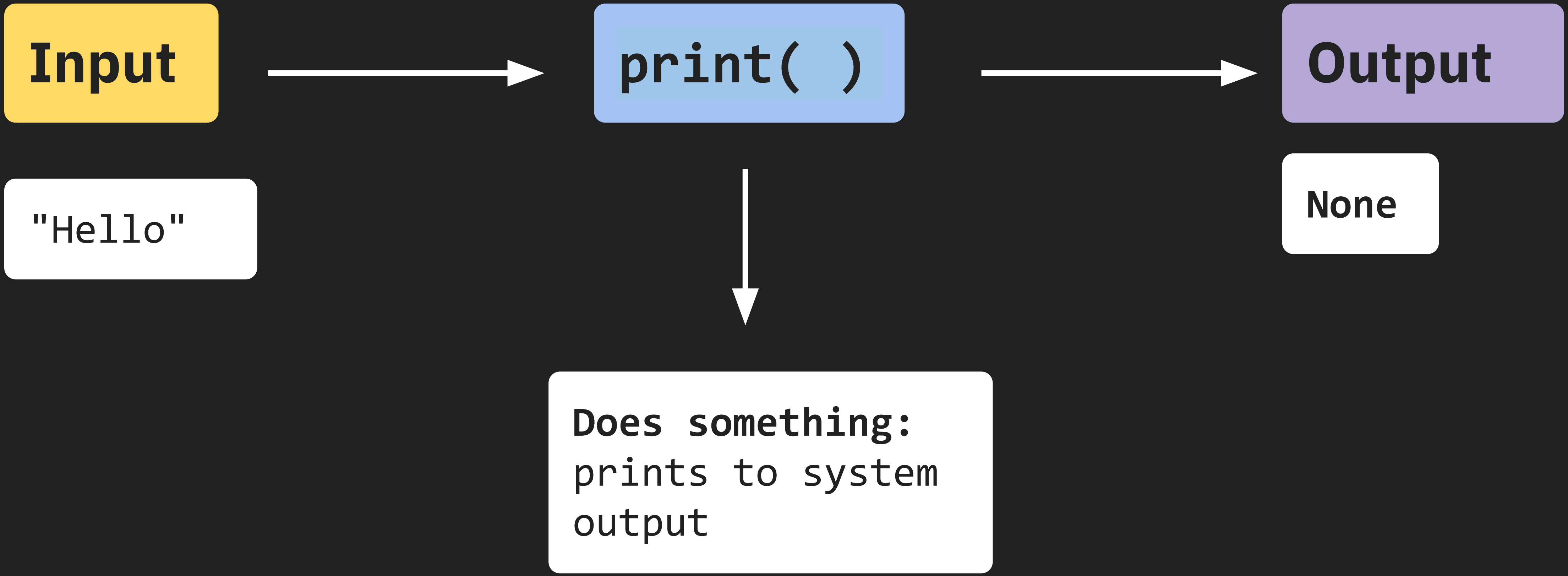
→ Sometimes, that something is **None**


Input
Function
(does something)
Output

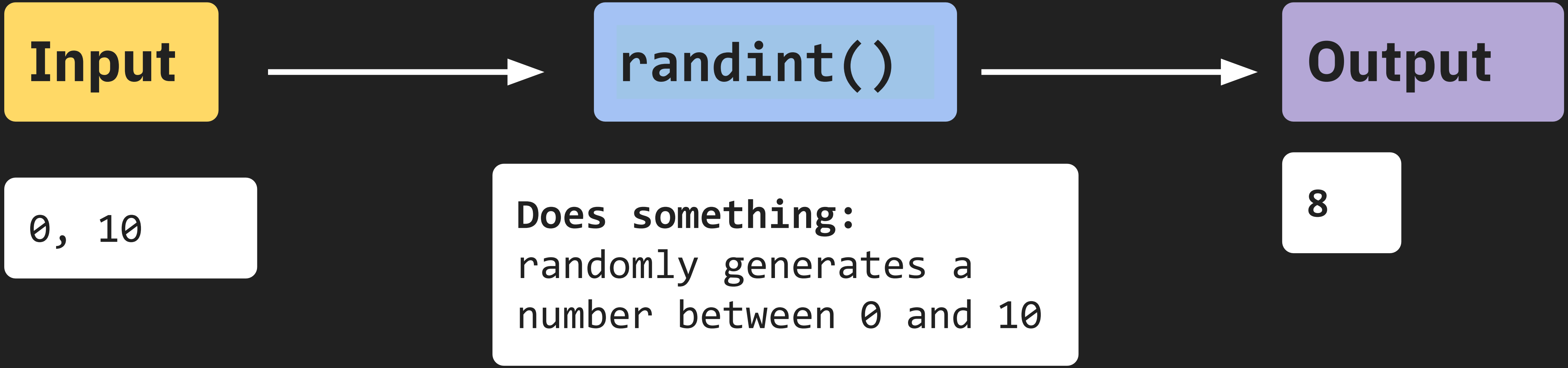
```
def add(a, b):
    c = a + b
    return c
```



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


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

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

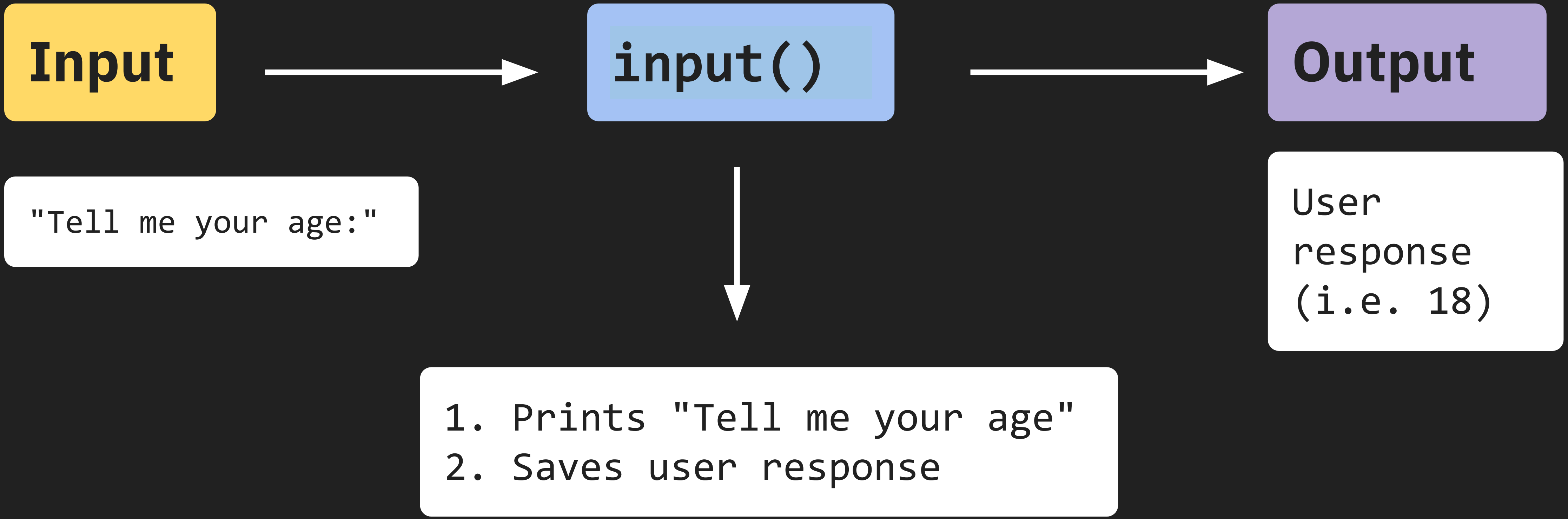


```
0, 10
```

Does something:
randomly generates a number between 0 and 10

```
8
```

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



3 reasons to use functions

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

Flow of Execution with Functions

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

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

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

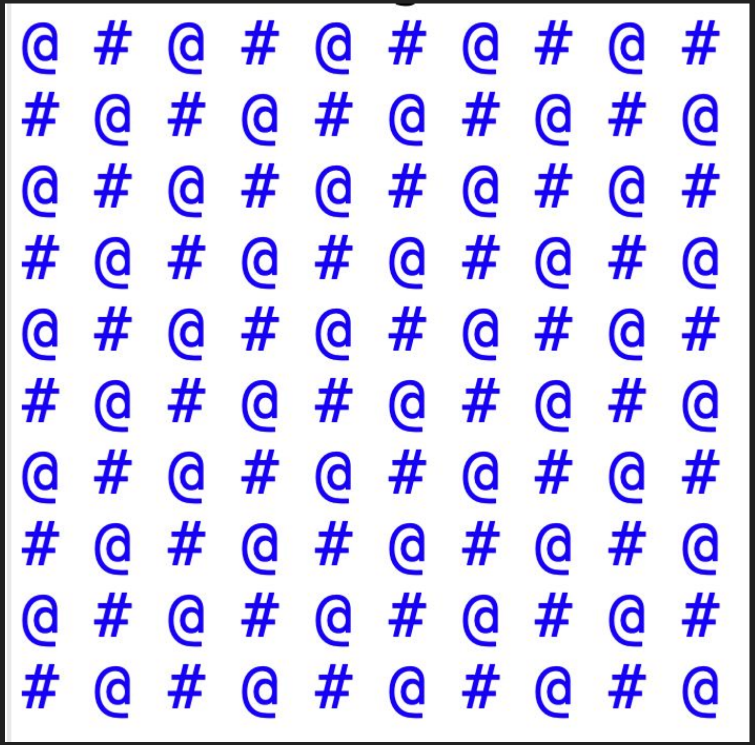
Code

```
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!
```

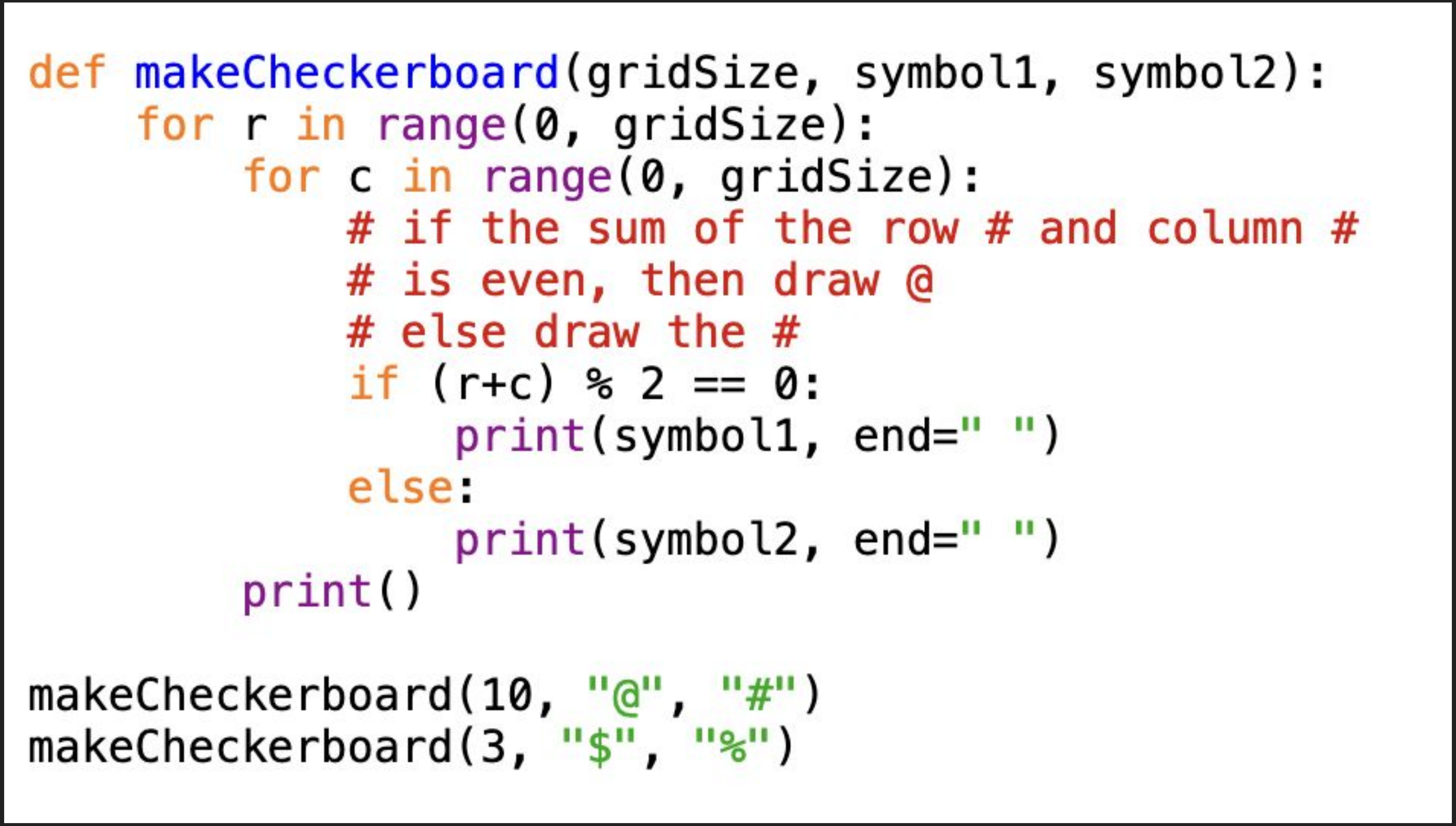
Programming Challenge



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

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 :)

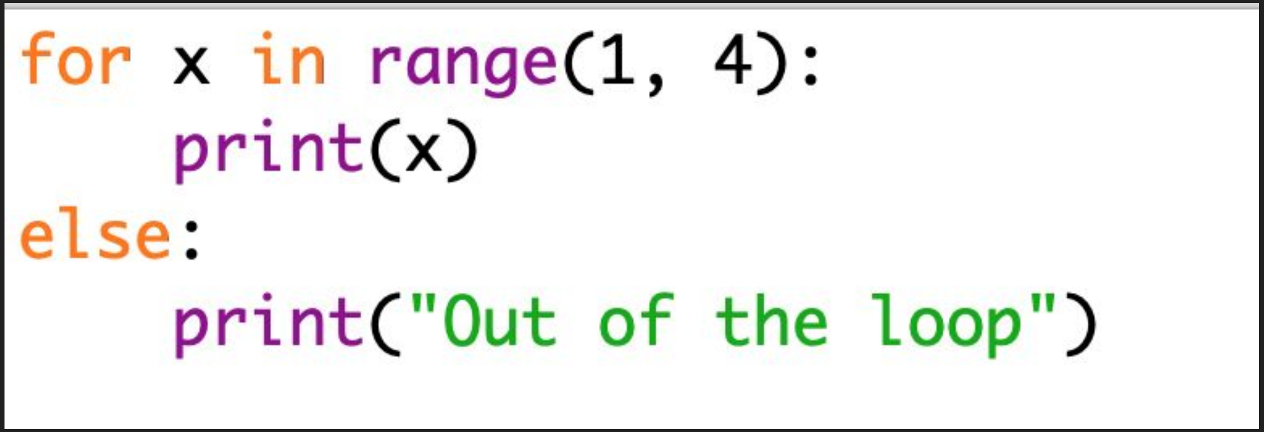
Homework

- Assignment #5 (due next class)

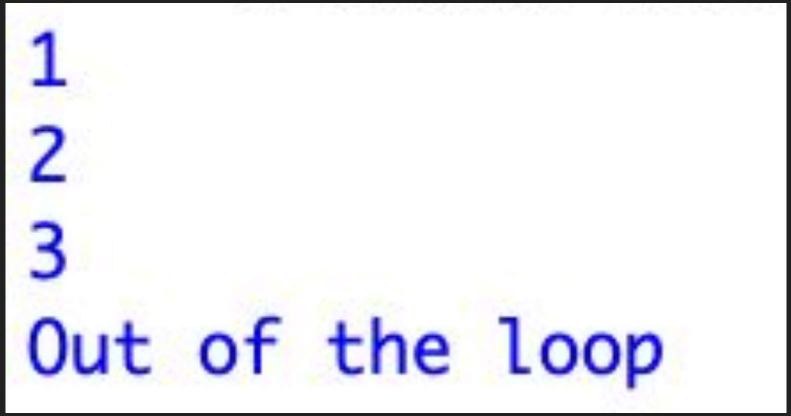
`for / else`

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.

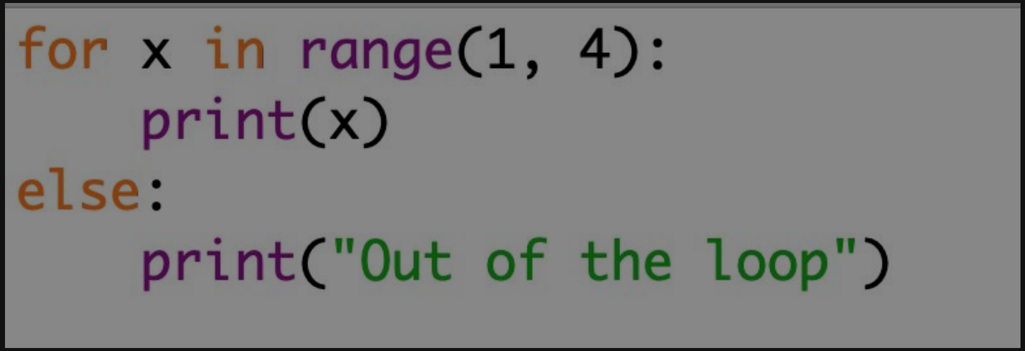


```python
for x in range(1, 4):
    print(x)
else:
    print("Out of the loop")
```

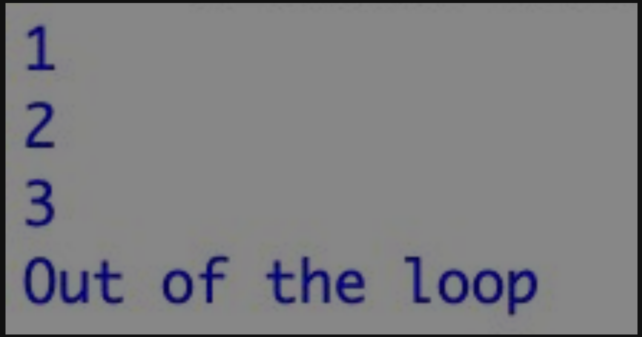


```
1
2
3
Out of the loop
```

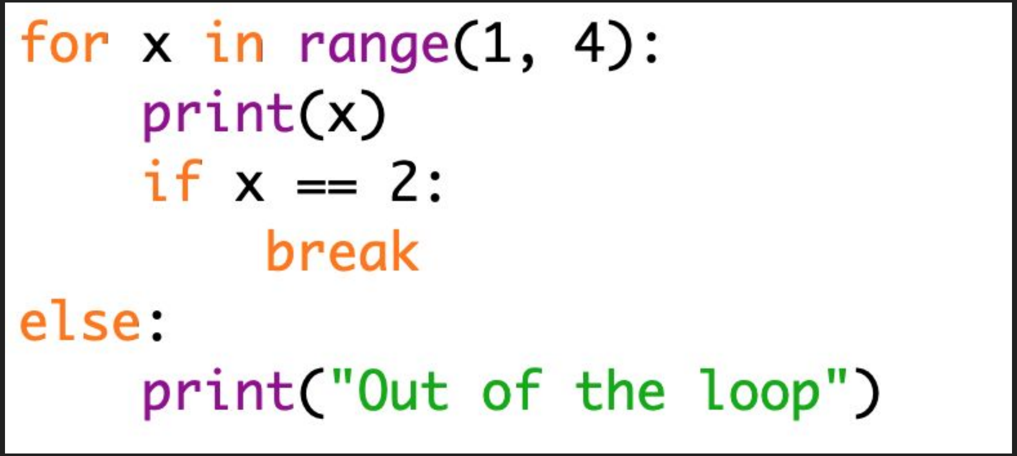
for / else



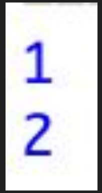
```
for x in range(1, 4):
    print(x)
else:
    print("Out of the loop")
```



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

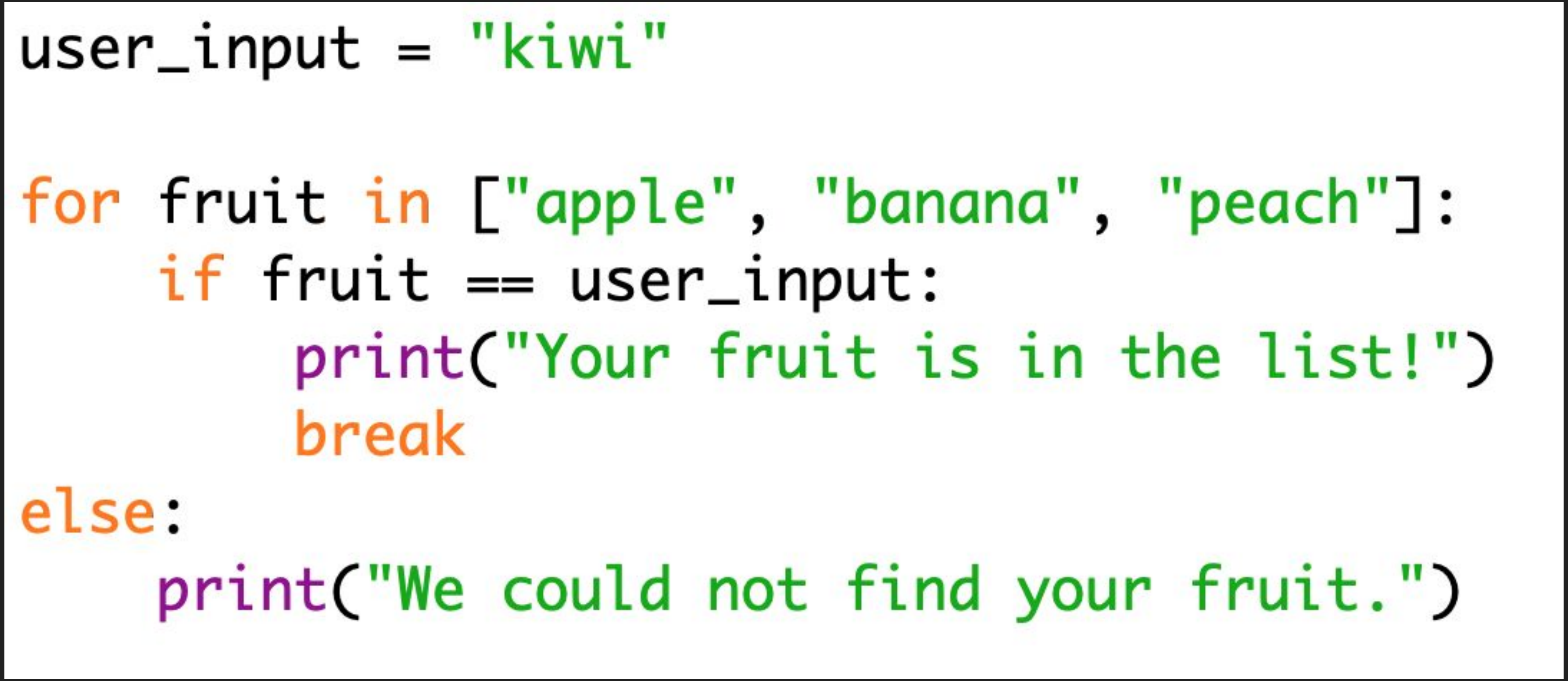


```
for x in range(1, 4):
    print(x)
    if x == 2:
        break
else:
    print("Out of the loop")
```

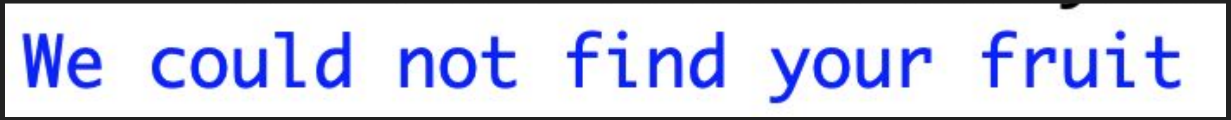


```
1
2
```

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?