● RECORDING IN-PROGRESS

Day 5: Functions

Friday 7/16/21

● RECORDING IN-PROGRESS

Agenda

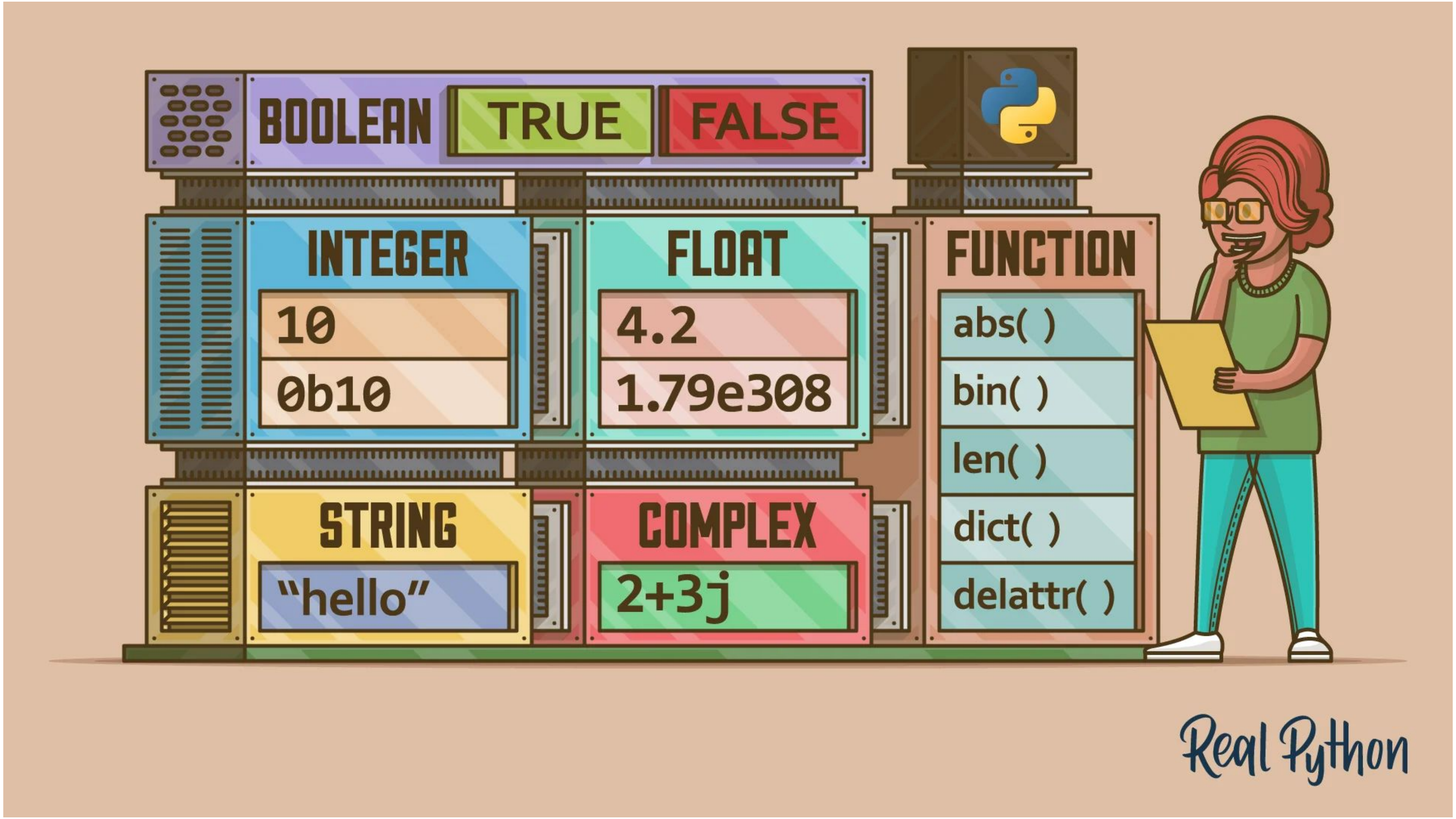
- Daily check-in
- Recap from yesterday
- Functions
  - Defining functions
  - Parameters and arguments to functions
  - Returning values from functions
- Examples
- Live coding - task!

Daily check-in

Check in

- Questions? Thoughts?
- Announcements
  - Next Monday our Tech Trek (Google) will start at **1pm EST** -not- 12:30pm EST
    - I'll post an announcement on Canvas as a reminder
  - Our session with Google will be over Google Meets, you don't need to download it or anything, but I would double check that you're able to join beforehand: https://meet.google.com/
  - I will give us the link to the meet room on Monday
- Lab 4
  - Questions first?
  - Go over solution
- Questions?

RECAP


BOOLEAN
TRUE
FALSE
INTEGER
10
0b10
FLOAT
4.2
1.79e308
FUNCTION
abs( )
bin( )
len( )
dict( )
delattr( )
STRING
"hello"
COMPLEX
2+3j
Real Python

What we’ve learned so far

- Types (int, str, float, bool, lists, etc.)
- Controlling the flow of our programs using if/elif/else
- Logic statements using >, <, ==, !=, etc.
- How to manipulate Lists
- Repeating code iteratively using loops (while loops, for loops)

Things we should (quickly) cover for today’s lab

- Working with 2D arrays / lists
    - What is a 2D array / list?

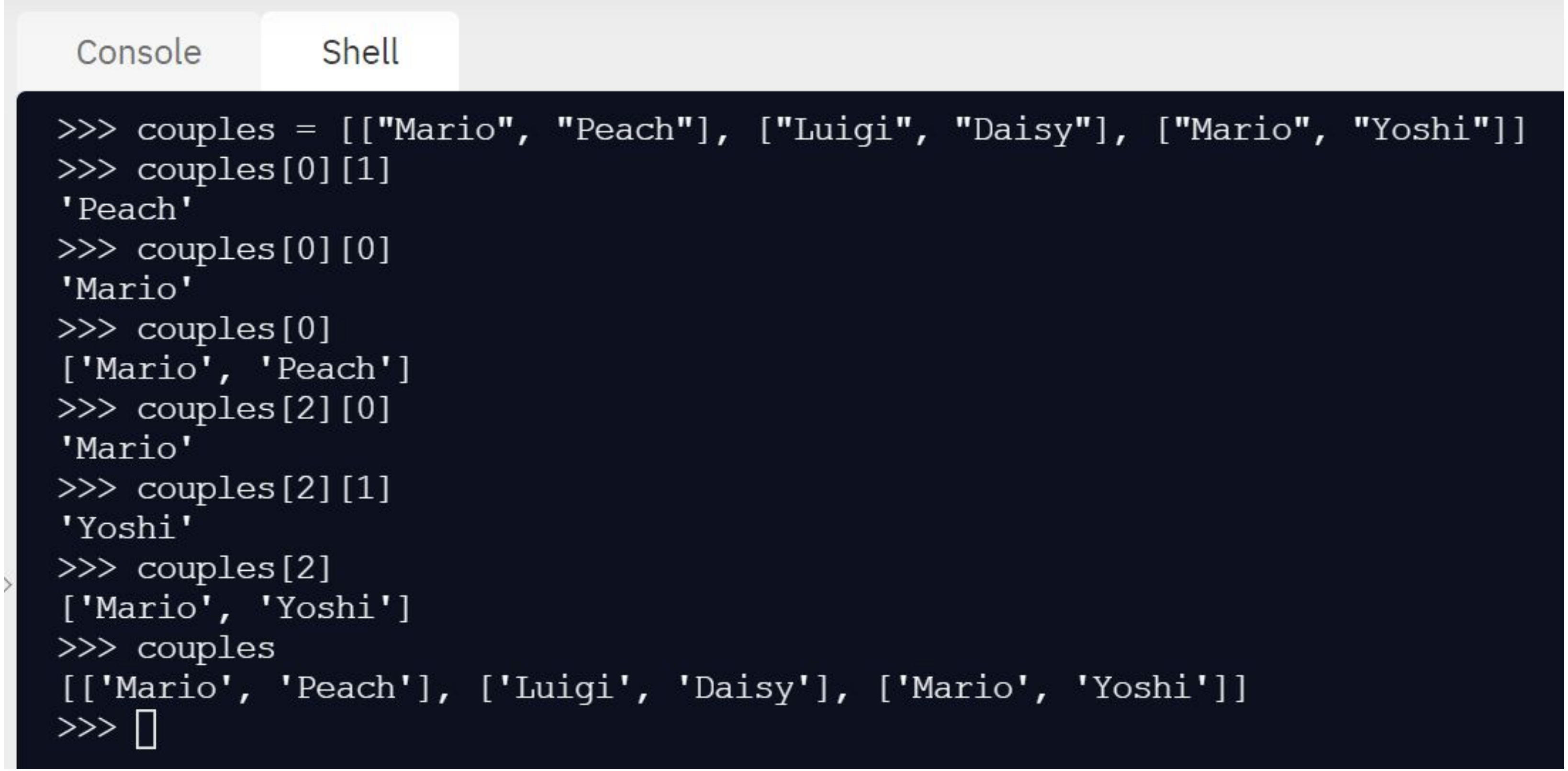
Things we should (quickly) cover for today’s lab

Lists of lists!

```
couples = [[“Mario”, “Peach”], [“Luigi”, “Daisy”], [“Mario, “Yoshi”]]
```

- How many *elements* (or, items) does **couples** have?
    - As in, if I did **len**(couples), what would be returned?
- How would I access the element “Peach”?
- How would I replace the first “Mario” with “Luigi” instead?
    - couples[0][0] = “Luigi”

Things we should (quickly) cover for today’s lab



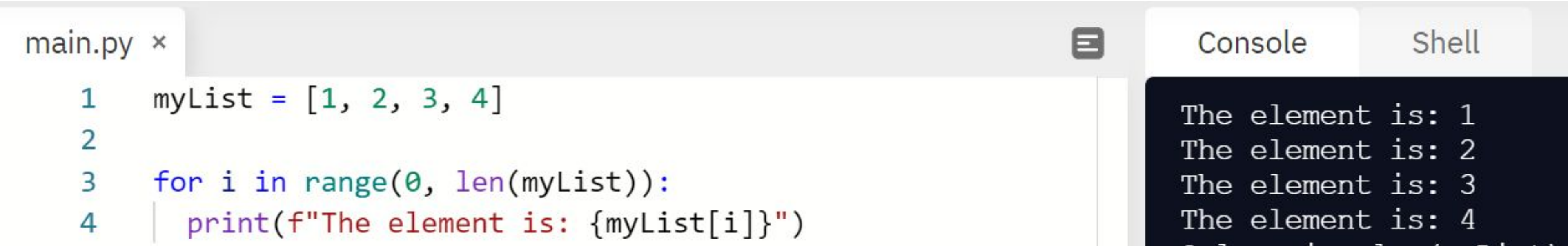
```
>>> couples = [["Mario", "Peach"], ["Luigi", "Daisy"], ["Mario", "Yoshi"]]
>>> couples[0][1]
'Peach'
>>> couples[0][0]
'Mario'
>>> couples[0]
['Mario', 'Peach']
>>> couples[2][0]
'Mario'
>>> couples[2][1]
'Yoshi'
>>> couples[2]
['Mario', 'Yoshi']
>>> couples
[['Mario', 'Peach'], ['Luigi', 'Daisy'], ['Mario', 'Yoshi']]
>>> 
```

Things we should (quickly) cover for today’s lab

- Another way to write **for** loops
- Let’s say I’m given the list: [1, 2, 3, 4, 5]
- How could I wrote a for loop that iterates over that list and prints each element by using the element’s **index** in the list?

Things we should (quickly) cover for today’s lab

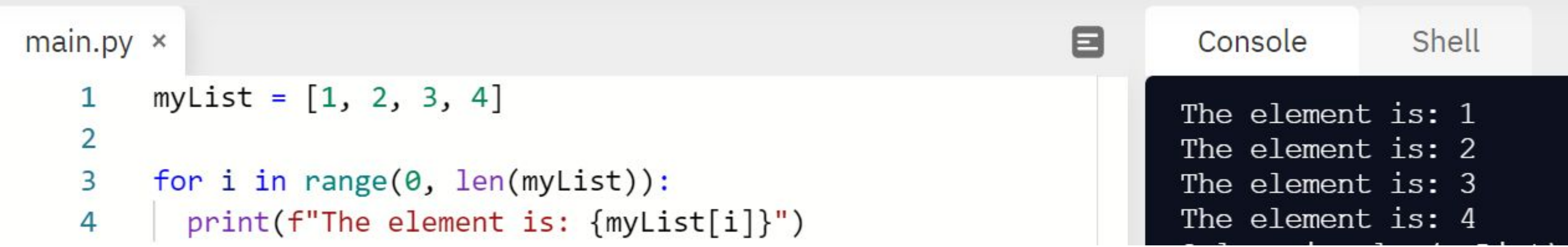
- Another way to write **for** loops
- Let’s say I’m given the list: [1, 2, 3, 4, 5]
- How could I wrote a for loop that iterates over that list and prints each element by using the element’s **index** in the list?



```python
myList = [1, 2, 3, 4]

for i in range(0, len(myList)):
  print(f"The element is: {myList[i]}")
```

```
The element is: 1
The element is: 2
The element is: 3
The element is: 4
```

Things we should (quickly) cover for today's lab



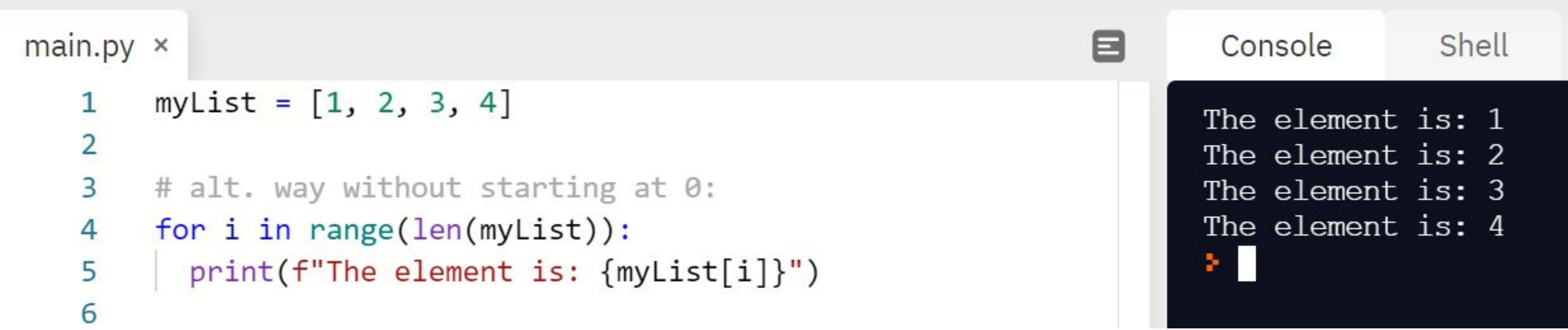
```python
myList = [1, 2, 3, 4]

for i in range(0, len(myList)):
  print(f"The element is: {myList[i]}")
```

```
The element is: 1
The element is: 2
The element is: 3
The element is: 4
```

With the way we've written the code above, we're essentially saying: for each **i** in the range between 0 and the size of **myList**, do <this>

* **range**(start, stop) provides a sequence of numbers starting at **start** (inclusive) and ending at **stop** (exclusive). This is more similar to what you might've seen writing loops in other languages: *for (i = 0; i < myList.size(); i++) { .... }*

Things we should (quickly) cover for today’s lab

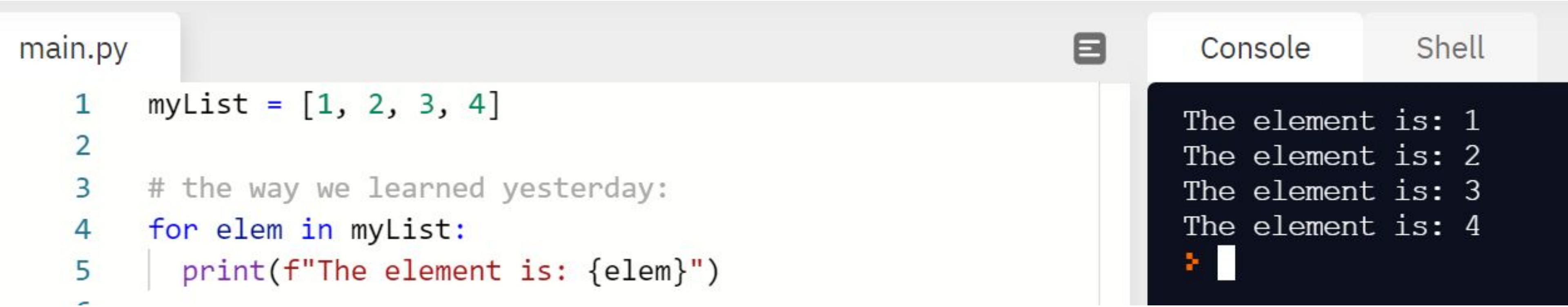


```python
myList = [1, 2, 3, 4]

# alt. way without starting at 0:
for i in range(len(myList)):
  print(f"The element is: {myList[i]}")

```

```
The element is: 1
The element is: 2
The element is: 3
The element is: 4
> 
```

As a heads up, **range**(…) expects either 1 or 2 arguments (the values between the parentheses). The first value is where to START counting, and the last value is where to STOP counting (*exclusive*). So, given **range**(x, y), we would get back a list that ranges from [x, y) in math notation. But we could also pass in only 1 value, like above, and Python will automatically start counting at 0.

Things we should (quickly) cover for today’s lab



```python
myList = [1, 2, 3, 4]

# the way we learned yesterday:
for elem in myList:
    print(f"The element is: {elem}")
```

```
The element is: 1
The element is: 2
The element is: 3
The element is: 4
```

This is the way we learned yesterday. We’re looping over an individual element in a collection (here, we’re looping over myList) then inside of the code, we have direct access to that specific element.

New topic: Functions

Functions

- Built-in functions that come with Python
  - input( ) = a function that allows user input
  - print( ) = a function that prints to the console
  - len( ) = a function that returns the length (size) of something
  - And so on: https://www.w3schools.com/python/python_ref_functions.asp
- What are some benefits of functions?

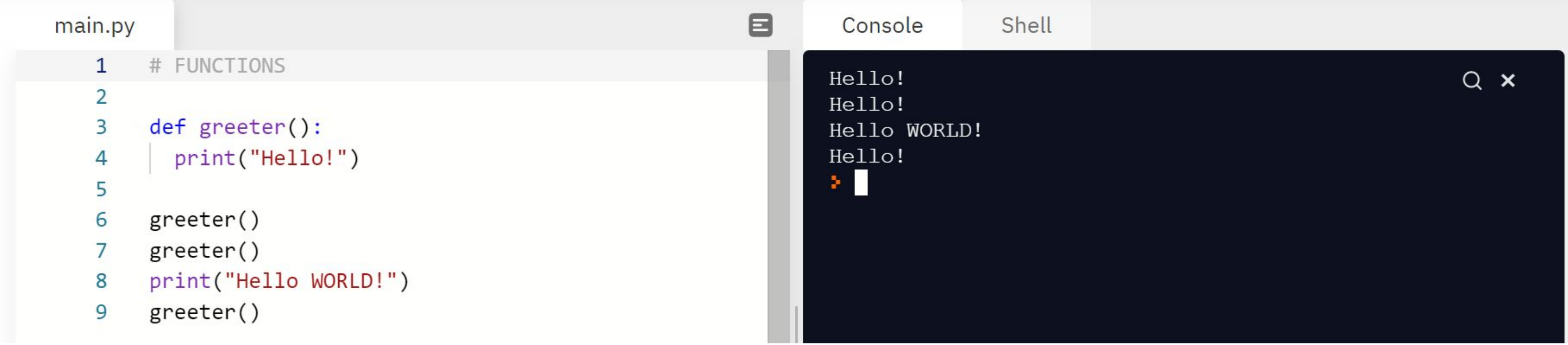
Functions

- Built-in functions that come with Python
  - input( ) = a function that allows user input
  - print( ) = a function that prints to the console
  - len( ) = a function that returns the length (size) of something
  - And so on: https://www.w3schools.com/python/python_ref_functions.asp
- What are some benefits of functions?
  - Reusable & convenient - we can repeatedly use a function without re-writing the code
  - Concise - we can shorten our overall code by using functions
  - Customizable - we can write our own functions ourselves
  - (and plenty other reasons too)

Functions

What is a function?

A function is a chunk of Python code that you give a name. The purpose of any function is to execute that same code any time you write (or, “call”) the function’s name in your program.



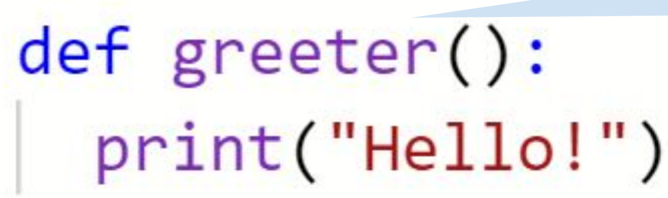
```python
# FUNCTIONS

def greeter():
  print("Hello!")

greeter()
greeter()
print("Hello WORLD!")
greeter()
```

```
Hello!
Hello!
Hello WORLD!
Hello!
>
```

Writing Functions

For Python to know that you’re creating a function, you must **define** your function using the keyword **def**. (*How does this differ from C/C++/Java, etc?*)

(1) Begin your function with the keyword **def**



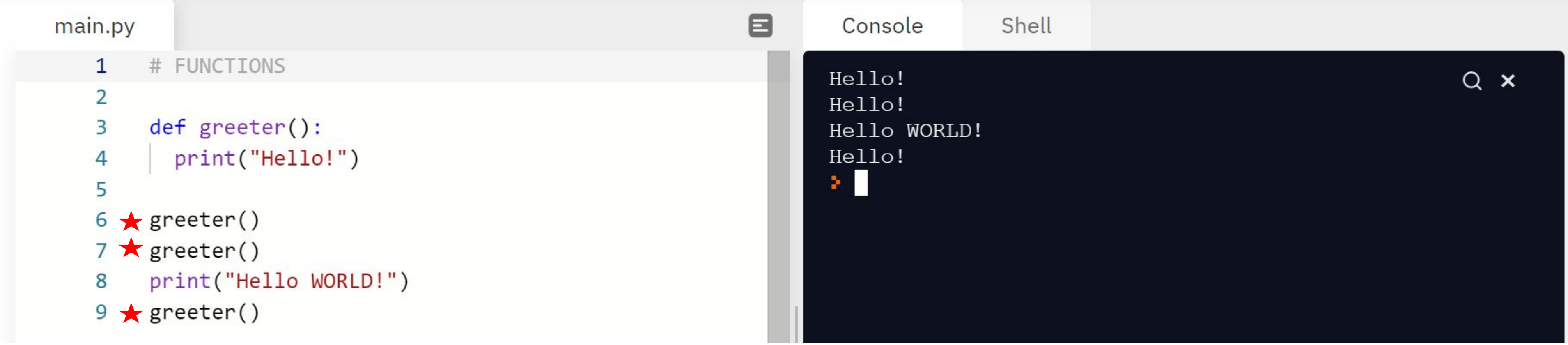
```
def greeter():
    print("Hello!")
```

(2) Give your function a name (don’t name it a built-in func. name)

(3) Indent any of the code that belongs to that function

Functions

The code you write in functions will **only execute when you call the function.**

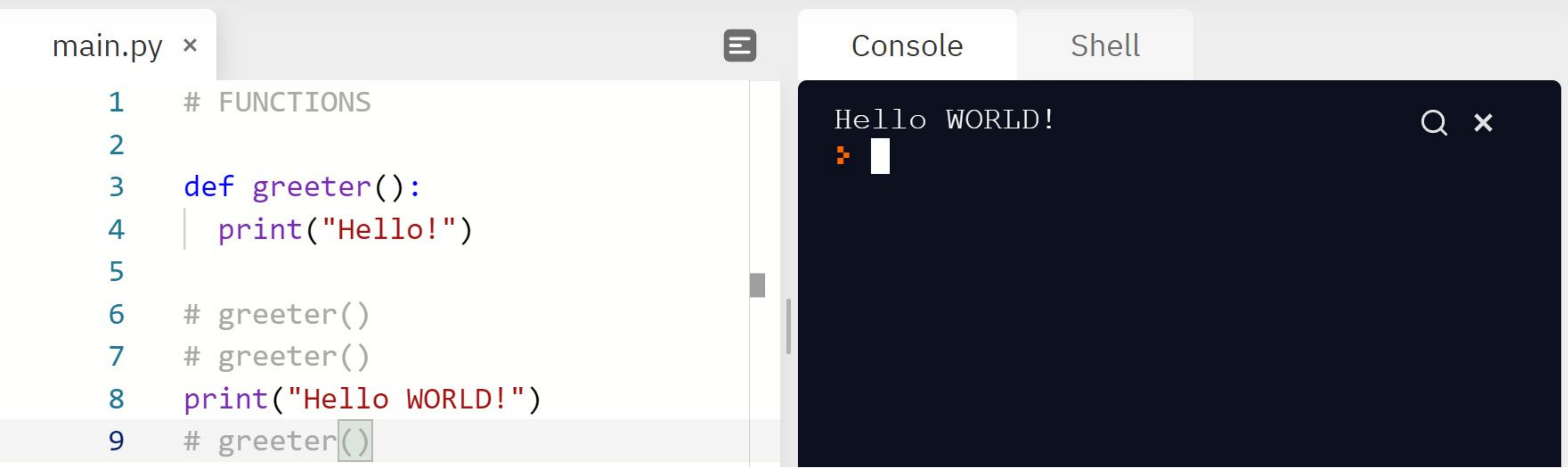


```
  # FUNCTIONS

  def greeter():
      print("Hello!")

★ greeter()
★ greeter()
  print("Hello WORLD!")
★ greeter()
```

```
Hello!
Hello!
Hello WORLD!
Hello!
>
```

Functions

The code you write in functions will **only execute when you call the function.**

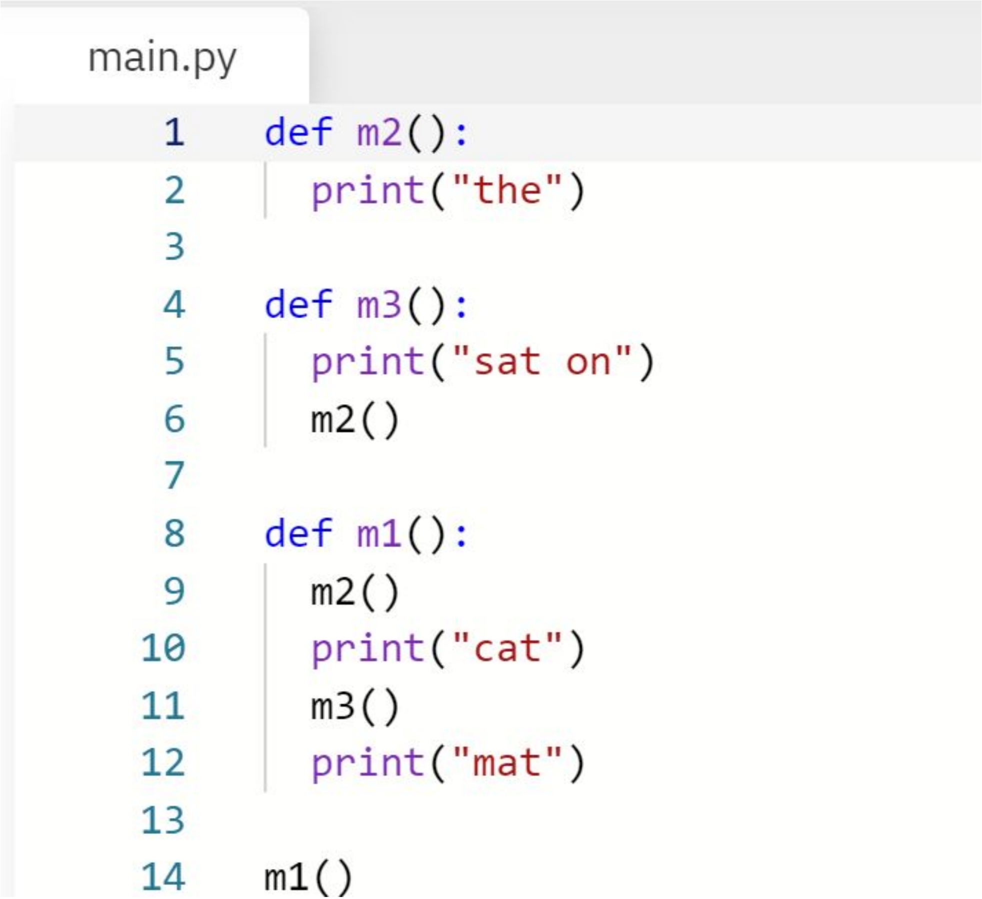


```python
# FUNCTIONS

def greeter():
  print("Hello!")

# greeter()
# greeter()
print("Hello WORLD!")
# greeter()
```

```
Hello WORLD!
```

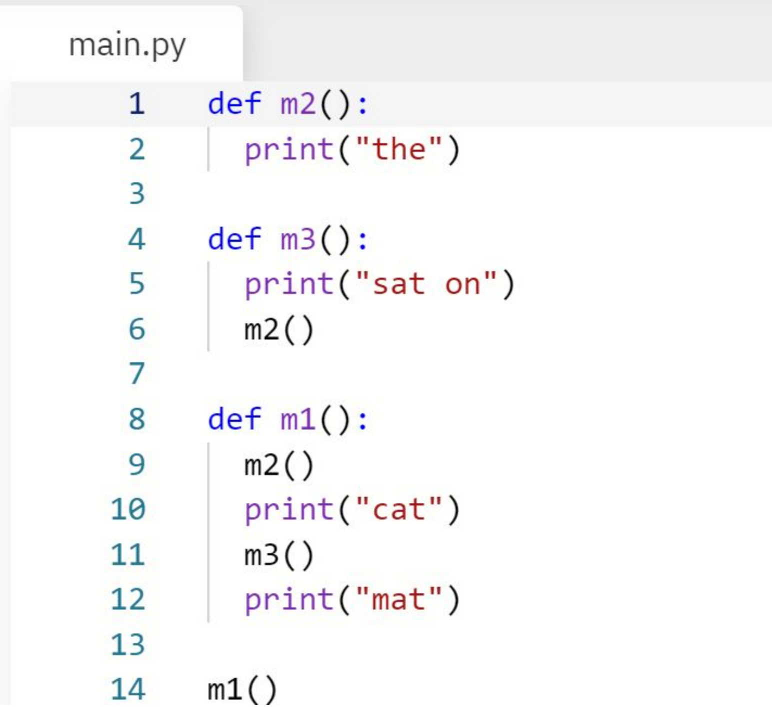
Functions

What will this program display when it runs?

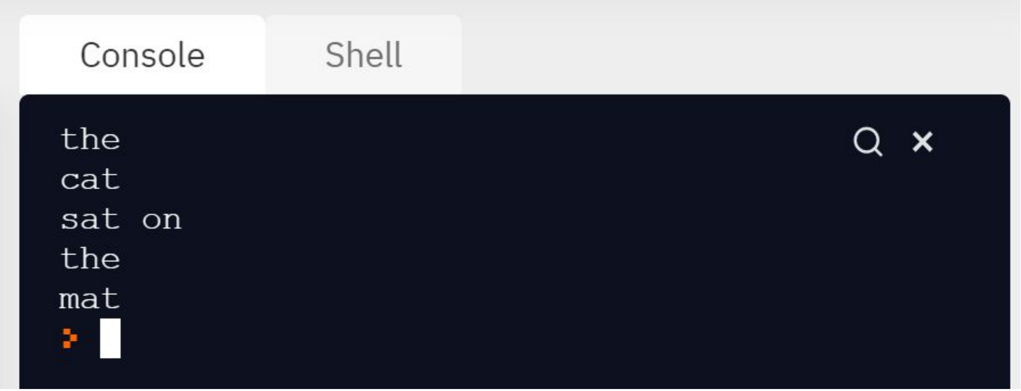


```python
def m2():
    print("the")

def m3():
    print("sat on")
    m2()

def m1():
    m2()
    print("cat")
    m3()
    print("mat")

m1()
```

Functions



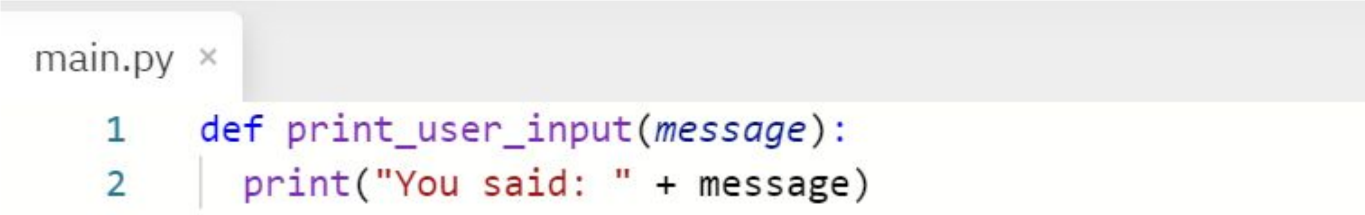
```python
def m2():
    print("the")

def m3():
    print("sat on")
    m2()

def m1():
    m2()
    print("cat")
    m3()
    print("mat")

m1()
```



```
the
cat
sat on
the
mat
>
```

Giving information to functions

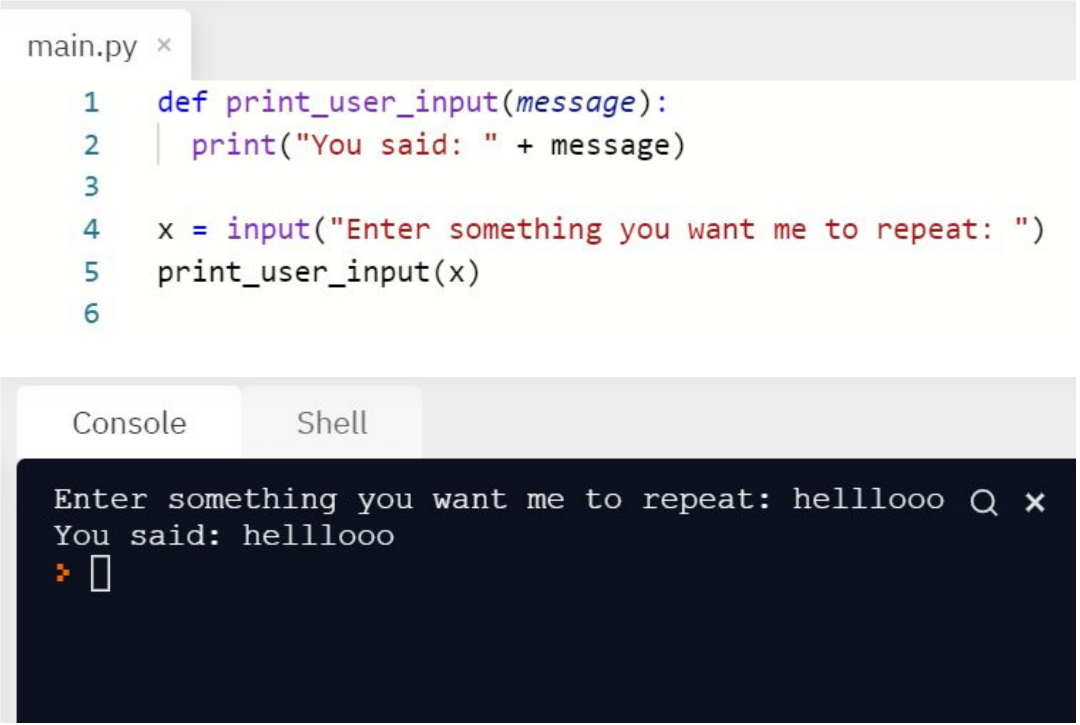
Sometimes, we may want to “give” information to functions by passing (or giving) a value to the function as a parameter input:



```python
def print_user_input(message):
    print("You said: " + message)
```

What will the **print_user_input** function do?

Giving information to functions

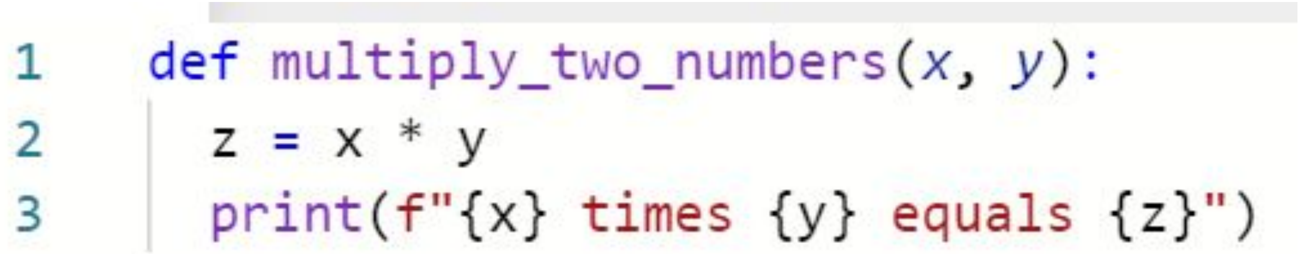


```python
def print_user_input(message):
  print("You said: " + message)

x = input("Enter something you want me to repeat: ")
print_user_input(x)

```

```
Enter something you want me to repeat: helllooo
You said: helllooo
> 
```

Giving information to functions

Sometimes we'll want our function to know multiple pieces of information

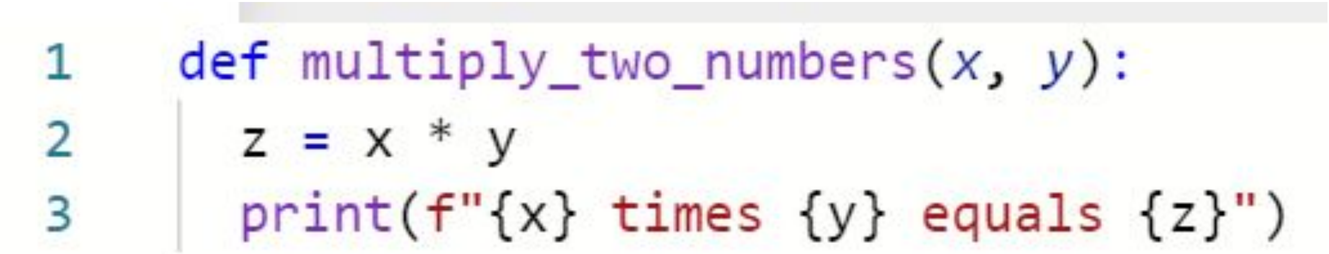


```
def multiply_two_numbers(x, y):
  z = x * y
  print(f"{x} times {y} equals {z}")
```

What does this function do, and how many parameters does the function expect?

Giving information to functions

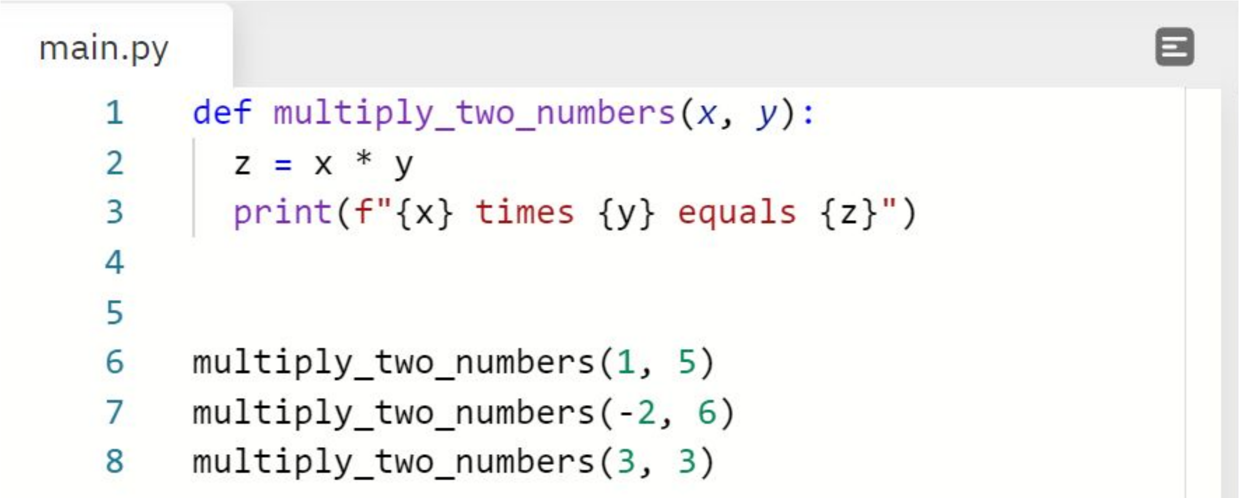
Sometimes we’ll want our function to know multiple pieces of information



```python
def multiply_two_numbers(x, y):
  z = x * y
  print(f"{x} times {y} equals {z}")
```

- Multiplies **x** by **y**, expects 2 parameters
- Python will associate the first value you pass in as the variable **x**
- Python will associate the second value you pass in as the variable **y**

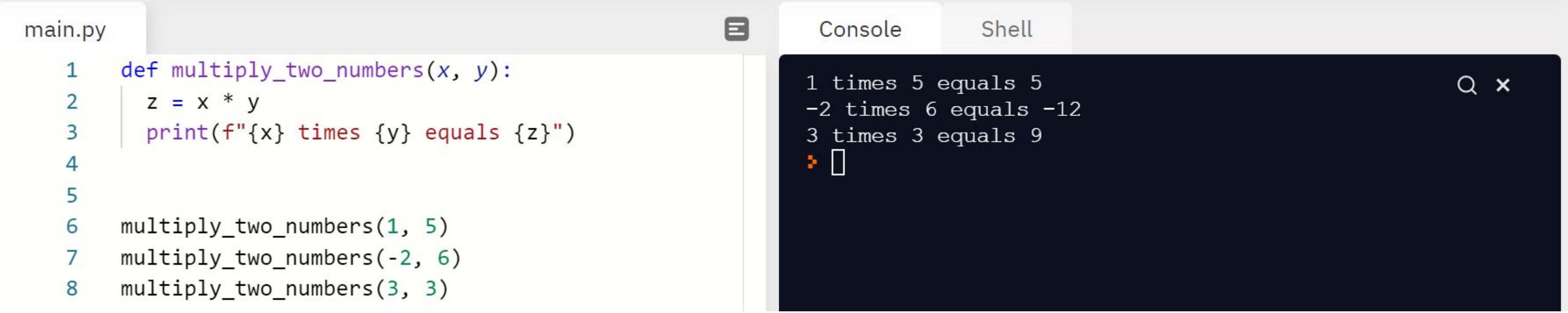
Giving information to functions



```
1  def multiply_two_numbers(x, y):
2    z = x * y
3    print(f"{x} times {y} equals {z}")
4
5
6  multiply_two_numbers(1, 5)
7  multiply_two_numbers(-2, 6)
8  multiply_two_numbers(3, 3)
```

Notice how I’ve called the function in lines 6, 7, 8. What will the output be?

Giving information to functions

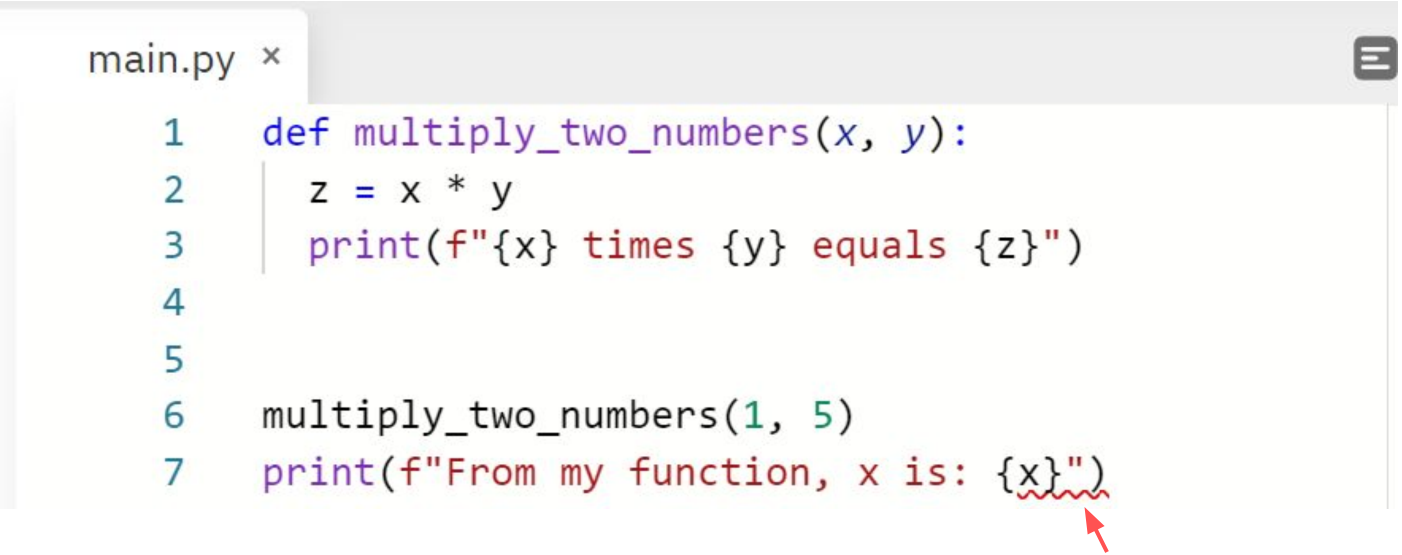


```python
def multiply_two_numbers(x, y):
    z = x * y
    print(f"{x} times {y} equals {z}")


multiply_two_numbers(1, 5)
multiply_two_numbers(-2, 6)
multiply_two_numbers(3, 3)
```

```
1 times 5 equals 5
-2 times 6 equals -12
3 times 3 equals 9
```

Python uses the variables that you’ve named inside the parentheses of the function (aka **x** and **y**) for all of the code that belongs to that function.
That is, Python will remember the values for **x** and **y** inside any of its indented code. HOWEVER, Python will immediately forget about the values of **x** and **y** outside of that function.

Giving information to functions

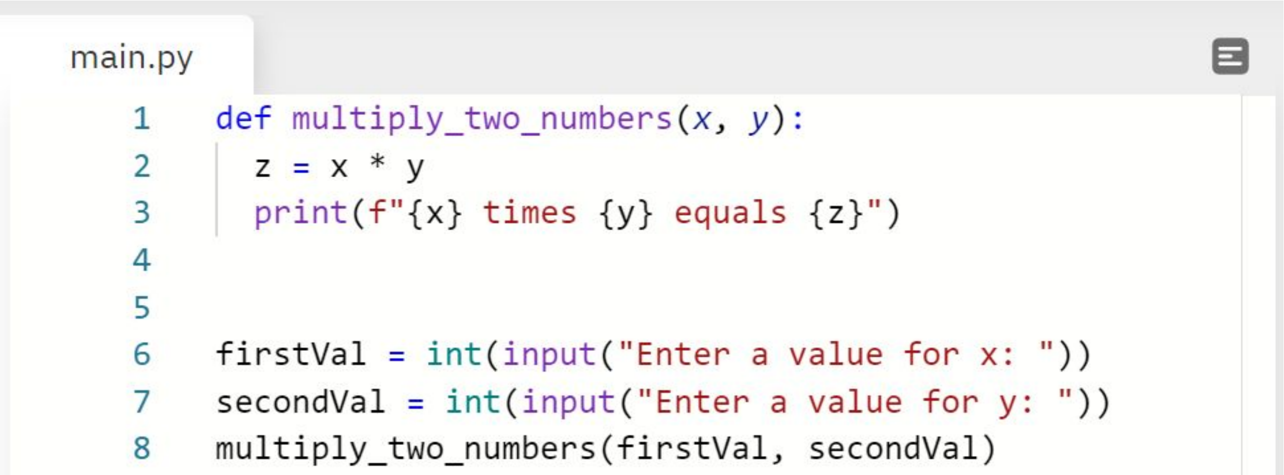


```
1   def multiply_two_numbers(x, y):
2       z = x * y
3       print(f"{x} times {y} equals {z}")
4
5
6   multiply_two_numbers(1, 5)
7   print(f"From my function, x is: {x}")
```

x is only defined inside of the function multiply_two_numbers, NOT outside of the function! Therefore, we can’t use it in line 7 - we get an error.

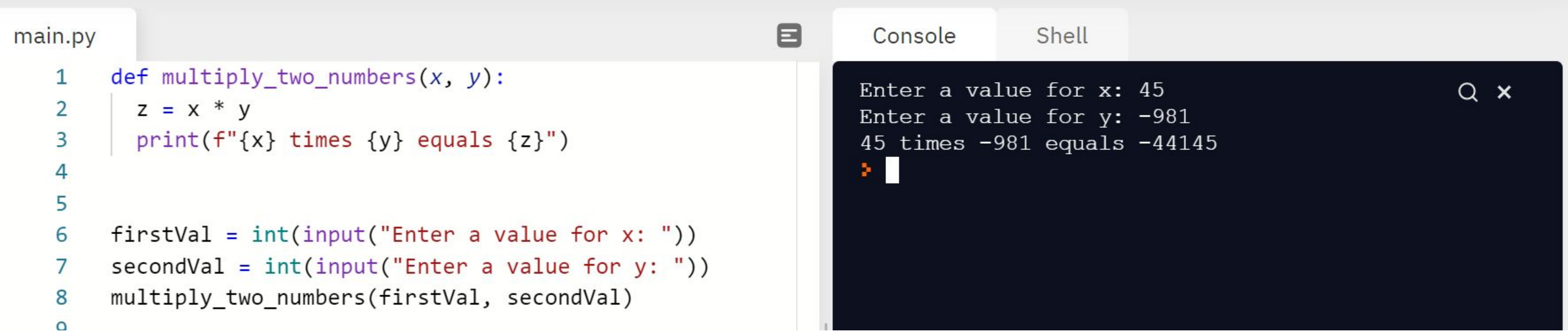
Passing variables to functions

We can pass “hard-coded” values to our function, like how we passed in the integers (1, 5) in the previous slides. But we can also pass in variables that we declare and assign in our code:



```python
def multiply_two_numbers(x, y):
  z = x * y
  print(f"{x} times {y} equals {z}")


firstVal = int(input("Enter a value for x: "))
secondVal = int(input("Enter a value for y: "))
multiply_two_numbers(firstVal, secondVal)
```

Passing variables to functions

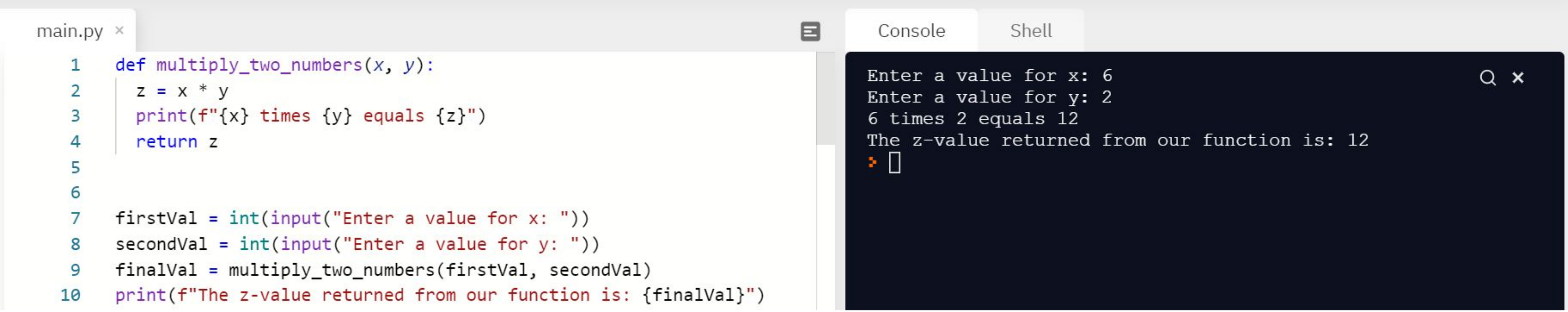


```python
def multiply_two_numbers(x, y):
  z = x * y
  print(f"{x} times {y} equals {z}")


firstVal = int(input("Enter a value for x: "))
secondVal = int(input("Enter a value for y: "))
multiply_two_numbers(firstVal, secondVal)
```

```
Enter a value for x: 45
Enter a value for y: -981
45 times -981 equals -44145
>
```

Notice how our function knew **x** was equal to 45, and **y** was equal to -981, even though we didn't tell the function that *explicitly* like before, we used the values assigned to our variables

Returning values **from** functions



```python
def multiply_two_numbers(x, y):
  z = x * y
  print(f"{x} times {y} equals {z}")
  return z


firstVal = int(input("Enter a value for x: "))
secondVal = int(input("Enter a value for y: "))
finalVal = multiply_two_numbers(firstVal, secondVal)
print(f"The z-value returned from our function is: {finalVal}")
```

```
Enter a value for x: 6
Enter a value for y: 2
6 times 2 equals 12
The z-value returned from our function is: 12
>
```

Notice how in our function **multiply_two_numbers**, we use the keyword **return** to return the value for **z**. Many functions we will write are written to return a specific value, so don't be surprised when you see that "return" keyword at the end of a function!

Task

Let’s say I want to write a function for the RPS game we coded during yesterday’s lab….

- On a piece of paper or in a text editor (you can create a new repl in replit), define a function that can compare the robot’s RPS choice to the user’s RPS choice to determine who wins

```
def functionName(…):
        <code>
```

Ashley will show her solution at 11:55am (10 min) - **Solution is available in Replit under Lectures/Lecture-05**