Python Programming: Functions

Learning Objectives

After this lesson, you will be able to…

- Identify when to use a function.
- Create and call a function with arguments.
- Return a value from a function.

Let’s Consider a Repetitive program...

Consider a program that prints a $5 shipping charge for products on a website:

```
print("You've purchased a Hanging Planter.")
print("Thank you for your order. There will be a $5.00 shipping charge for t

# 10 minutes later...
print("You've purchased a Shell Mirror.")
print("Thank you for your order. There will be a $5.00 shipping charge for t

# 5 minutes later...
print("You've purchased a Modern Shag Rug.")
print("Thank you for your order. There will be a $5.00 shipping charge for t
```

What if there are 1,000 orders?

Functions

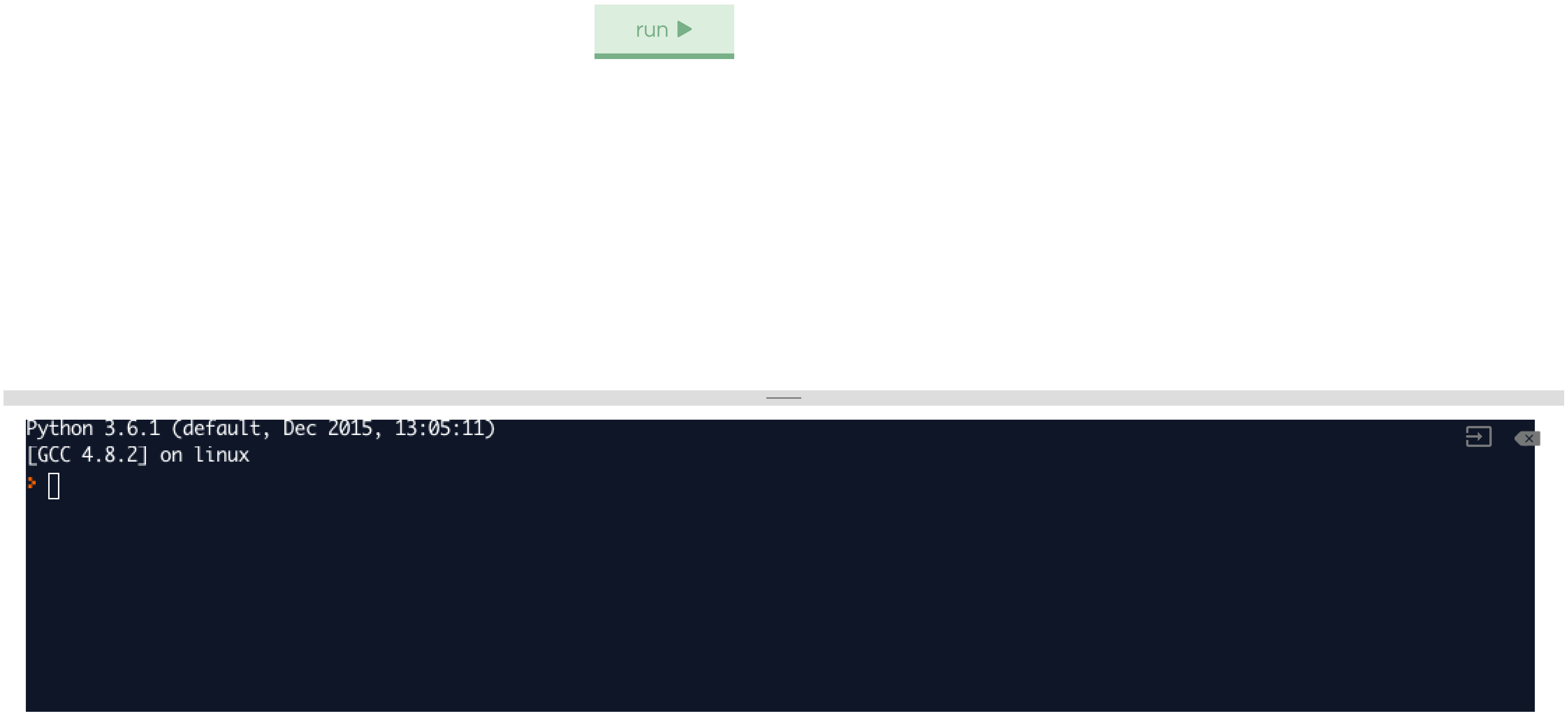
We can write a **function** to print the order.

A function is simple — it's a reusable piece of code. We only define it once. Later, we can use its name as a shortcut to run that whole chunk of code.

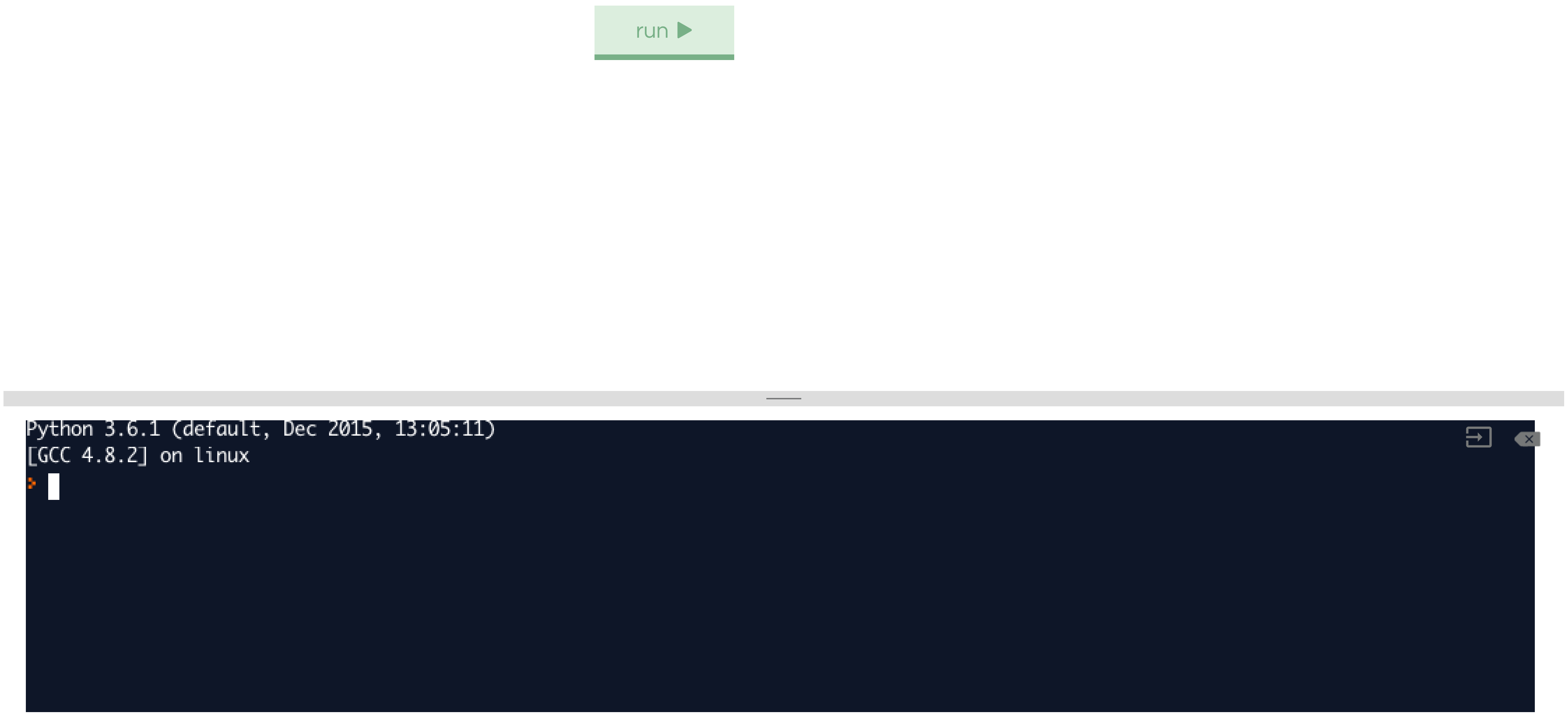
- Functions are defined using the `def` syntax.
  - `def` stands for "define."
- In this case, we're *defining* a function named 'function_name.'

```
def function_name():
  # What you want the function to do


# Call the function by name to run it:
function_name()


# 10 minutes later...
function_name()
```

Protip: Don't forget the `()`, and be sure to indent!



```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
> 
```

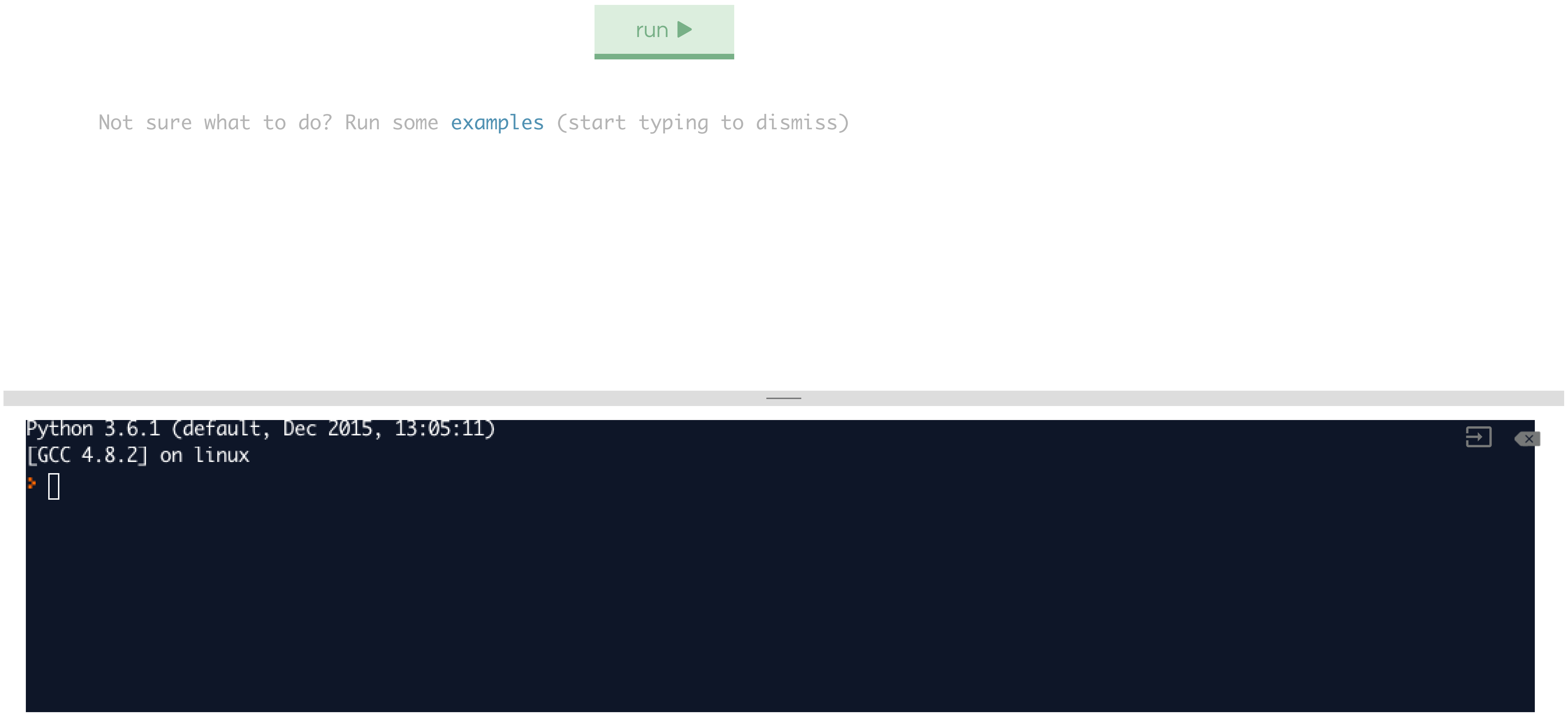


```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
```

Multi-Line Functions

How many lines of code can a function have? - As many lines of code as you’d like! - Just indent each line.

```
def welcome():
  print("Hello!")
  print("Bonjour!")


welcome()
```



Not sure what to do? Run some examples (start typing to dismiss)

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
```

You Do: FizzBuzz

This is a *very* common programming question. It's often on job interviews and a buzzword in the industry as a simple but common task to show your understanding.

Open a new Python file, `fizzbuzz.py`.

- Write a program that prints the numbers from 1 to 101.
- But, for multiples of three, print "Fizz" instead of the number.
- For multiples of five, print "Buzz".
- For numbers which are multiples of both three and five, print "FizzBuzz".

Quick Review: Functions

Functions are reusable chunks of code. They can have anything in them.

- Define functions using the `def` keyword.
- A function must be **called** before the code in it will run!
- You will recognize function calls by the `()` at the end.

```
# This part is the function definition!
def say_hello():
    print("hello world!")


# This part is actually calling/running the function!
say_hello()
```

You can call them as many times as you'd like, but they need to be defined above the code where you call them.

Up next: Parameters!

Discussion: Parameters

Remember this?

```
def print_order():
  print("Thank you for your order. There will be a $5.00 shipping charge for

print("You've purchased a Hanging Planter.")
print_order()

print("You've purchased a Shell Mirror.")
print_order()

print("You've purchased a Modern Shag Rug.")
print_order()
```

There's still repetition. How do you think we could improve it?

Addressing the repetition

We can dynamically pass a function values. This is a **parameter**.

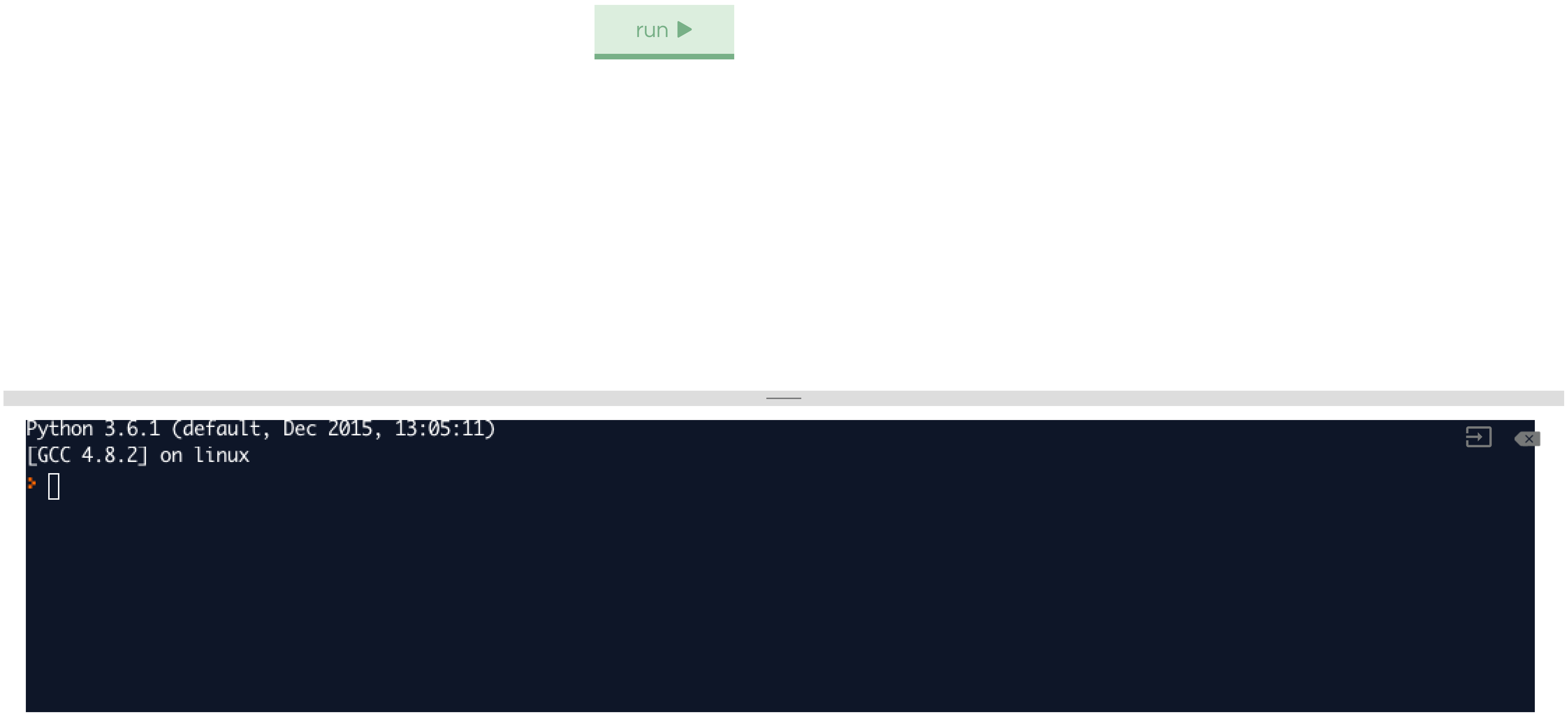
```
def print_order(product):
  print("Thank you for ordering the", product, ".")


print_order("Hanging Planter")
# Prints "Thank you for ordering the Hanging Planter."
print_order("Shell Mirror")
# Prints "Thank you for ordering the Shell Mirror."
print_order("Modern Shag Rug")
# Prints "Thank you for ordering the Modern Shag Rug."
```

Terminology Recap

Parameter: The variable that's defined in a function's declaration.

Argument: The actual value passed into the function when the function is called.

```
def my_function(parameter):
  # Does something.


my_function(argument)
```



```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
› 
```

Partner Exercise: Thanks a Latte

Pair up! Decide who will be the driver and who will be the navigator.

Imagine that you are tasked with creating a program to calculate the total amount, including sales tax, for each item at a coffee shop.

Create a new file, `latte.py`, and type the two functions below into it, which will calculate the total amount for two drinks:

Pro tip: Don’t just copy! Typing will be good practice.

Partner Exercise: Thanks a Latte

```python
def latte_total():
  price = 5.50
  sales_tax_rate = .10
  total_amount = price + (price * sales_tax_rate)
  print("The total is $", total_amount)

latte_total()

def americano_total():
  price = 4.75
  sales_tax_rate = .10
```

Keep it DRY (Don’t Repeat Yourself)

But what if we have several drinks at the coffee shop?

With your partner, think about a function that could print the total of any drink if you pass it the price, like this…

```
def calculate_total(price):
  #your code here


calculate_total(5.5) # This  was the latte
calculate_total(4.75) # This was the Americano
```

Your task: Write this!

Latte: Solution

How did it go?

Is this close to yours?

```
def calculate_total(price):
  sales_tax_rate = .10
  total_amount = price + (price * sales_tax_rate)
  print("The total is $", total_amount)

calculate_total(5.5) # This will print 6.05.
calculate_total(4.75) # This will print 5.225.
```

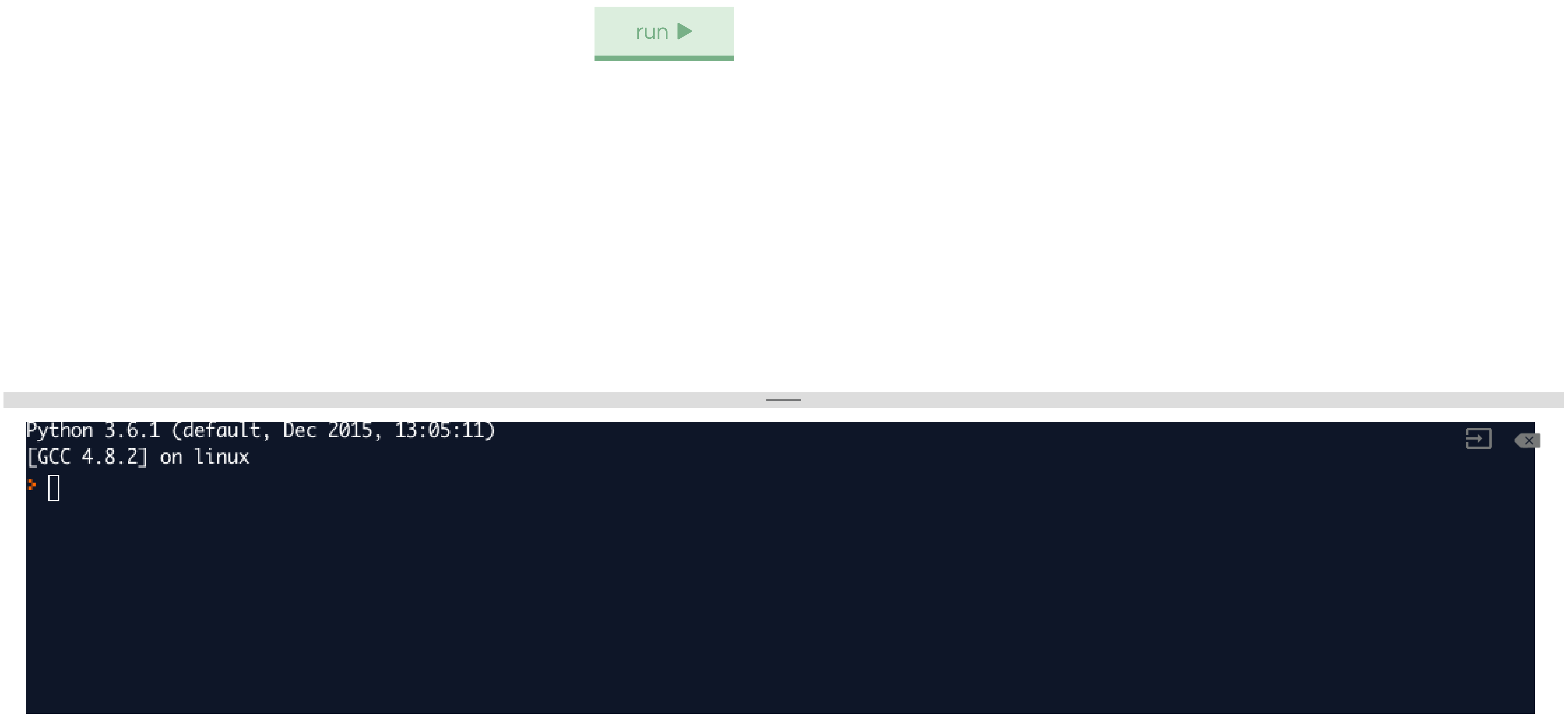
Multiple Parameters: Part 1

What about changing sales tax? We can pass as many values into the function as we want - we can have as many parameters as we want.

Here, we have a second parameter, `taxes`:

```
def calculate_total(price, taxes):
  total_amount = price + (price * taxes)
  print("The total is $", total_amount)


calculate_total(5.5, .10) # "price" is 5.5; "taxes" is .10. This will print
calculate_total(4.75, .12) # "price" is 4.75; "taxes" is .12. This will prin
```

Protip: Use a comma-separated list — (parameter1, parameter2, parameter3, parameter4)



```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
> 
```

Partner Exercise: Functions With Logic

With the same partner, switch drivers. You can use the same file or start a new one.

Let's go back to our shipping example. Depending on the order amount, our user might get free shipping, so the print statement is different.

Use this starter code, which works for one product. Can you build a function from it that works for any `product` and `order_amount`?

```
product = "Hanging Planter"
order_amount = 35

print("Thank you for ordering the Hanging Planter.")
if order_amount >= 30:
    print("It's your lucky day! There is no shipping charge for orders over
else:
    print("There will be a $5.00 shipping charge for this order.")
```

- **Hint:** You can put any code you'd like inside a function.
- **Reminder:** Don't forget to indent!

Quick Review: Functions with Parameters

Parameter: The variable that’s defined in a function’s declaration.

Argument: The actual value passed into the function when the function is called.

Order matters!

```
def do_something(parameter1, parameter2):
  # Does something.


do_something(argument1, argument2)
do_something(a_different_argument_1, a_different_argument_2)
```

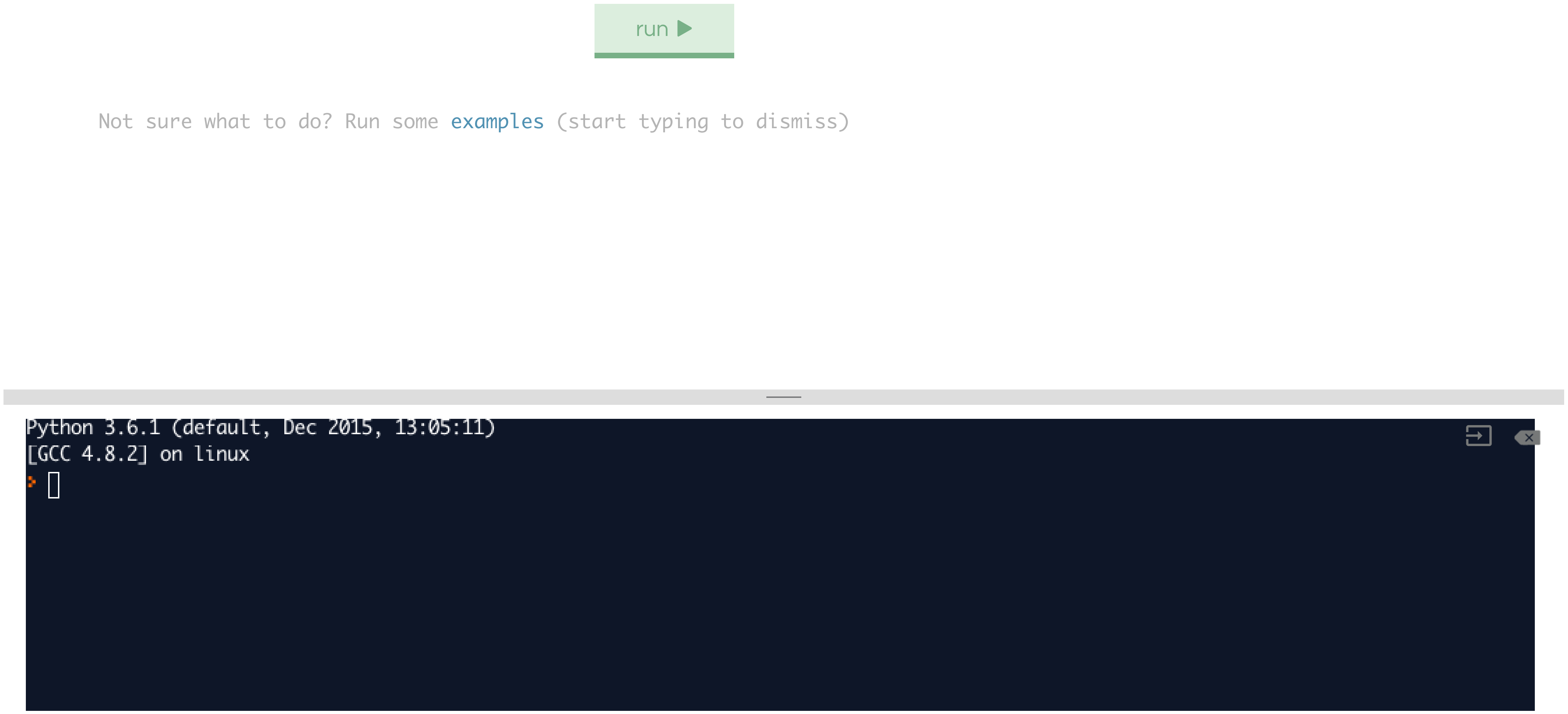
Next up: Returns.

The Return

Sometimes, we want values *back* from functions.

```
def calculate_total(price, taxes):
  total_amount = price + (price * taxes)
  print 'The total is $', total_amount
  # Send the total_amount for the drink back to the main program.
  return total_amount


# This just calls the function -  we've seen this.
calculate_total(5.5, .10)


# This is new! Save the amount of this drink into a variable "latte_total."
latte_total = calculate_total(5.5, .10)
```

- `total_amount` is returned to the main program.
- The value in `total_amount` is saved as `latte_total`.



Not sure what to do? Run some examples (start typing to dismiss)

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
›
```

Discussion: Return Statements With Logic

The `return` statement *exits a function*, not executing any further code in it. What do you think the following will print?

```
def mystery():
   return 6
   return 5


my_number = mystery()
print my_number
```

Discussion: What Will Happen?

What do you think will print out?

```
def add_bonus_points(score):
    if score > 50:
        return score + 10
    score += 20
    return score

total_points = add_bonus_points(55)
print(total_points)
```

Exiting a Function

We can also use `return` by itself as a way to exit the function and prevent any code that follows from running.

```python
def rock_and_roll(muted):
   song = "It's only Rock 'N' Roll"
   artist = "Rolling Stones"

   if (muted == True):
       return
       # Here, we use return as a way to exit a function
       # We don't actually return any value.
   print("Now playing: ", song, " by ", artist)

rock_and_roll(True)
```

Quick Knowledge Check

Looking at this code, where will the function stop if `x` is `10`?

```
def categorize(x):
  if (x < 8):
      return 8
  x += 3
  if (x < 15):
      return x
  return 100
```

Another Knowledge Check

Take this simple `adder` function:

```
def adder(number1, number2):
 return number1 + number2
```

Which of the following statements will result in an error?

A. `adder(10, 100.)`

B. `adder(10, '10')`

C. `adder(100)`

D. `adder('abc', 'def')`

E. `adder(10, 20, 30)`

Quick Review: Return Statements

Return statements allow us to get values back from functions:

```
def add_two(number):
 total = number + 2
 print(total)
 return total

final_var = add_two(3)
print final_var
```

Return statements also exit the function - no further code in the function happens!

```
def add_bonus_points(score):
    if score > 50:
        return score + 10
    score += 30
    return score

total_points = add_bonus_points(55)
```

Temperature Conversion

When we were learning about conditionals, we took a look at a program that let us know if it was too hot or too cold:

```
temperature = 308
if temperature > 299:
  print("It's too hot!")
elif temperature <= 299 and temperature > 288:
  print("It's just right!")
elif temperature <= 288 and temperature > 277:
  print("It's pretty cold!")
else:
  print("It's freezing!")
```

That’s good logic, but Kelvins aren’t incredibly useful on a day-to-day basis, unless you’re a scientist. Let’s use a function to convert temperatures.

You Do: Temperature Conversion

```
temperature = 308
if temperature > 299:
  print("It's too hot!")
elif temperature <= 299 and temperature > 288:
  print("It's just right!")
elif temperature <= 288 and temperature > 277:
  print("It's pretty cold!")
else:
  print("It's freezing!")
```

Here are the formulas to use:

- **Celsius to Kelvin** : `K = °C + 273`
- **Fahrenheit to Kelvin** : `K = (5/9) * (°F - 32) + 273`

Try to use one function to convert from either Fahrenheit or Celsius based upon a second parameter.

You Do: Temperature Conversion

Did you come up with something like this?

```
def convert(temp, scale):
    if scale == "Fahrenheit":
        return (5/9) * (temp - 32) + 273
    else:
        return temp +  273


temperature = convert(50,"Fahrenheit")
if temperature > 299:
  print("It's too hot!")
elif temperature <= 299 and temperature > 288:
  print("It's just right!")
```

Partner Exercise: Building a Copy

Get with a partner. Decide who will drive and who will navigate.

In a new local file, write a function, `copy_list`, that takes in a list, `original_list`, as a parameter. Your function should create a new list, `my_new_list` with the contents of the original list. Your function should return `my_new_list`.

Example:

```
my_list = [1, 2, 3]
my_new_list = copy_list(my_list)
print(my_new_list)
# Will print [1, 2, 3]
```

Hint: you'll need to declare `my_new_list` above (outside of) your `for` loop.

Make sure you run your function to check!

Partner Exercise: Reversing a List

With the same partner, switch driver and navigator.

In a local file (it can be the same one, if you'd like), write a function, `reverse_list`, that takes in a list, `my_list`, as a parameter. Your function should reverse the list in place and return it.

Example:

```
my_list = [1, 2, 3]
reversed_list = reverse_list(my_list)
print(reversed_list)
# Will print [3, 2, 1]
```

Make sure you run your function to check!

You Do: Reversing a List

Now, work on your own.

In a local file, write a function, `check_list_equality`, that takes in two lists, `first_list` and `second_list`, as parameters. Your function should return `True` if the two lists contain the same elements in the same order. Otherwise, it returns `False`.

Example:

```
list_one = [1, 2, 3]
list_two = [1, 2, 3]
list_three = [3, 2, 1]
print(check_list_equality(list_one, list_two)) # True
print(check_list_equality(list_one, list_three)) # False
```

Hint: Start by just making sure the lists have the same length!

Hint: You'll only need one `for` loop.

Summary + Q&A:

Can you now:

- Identify when to use a function?
- Create and call a function with arguments?
- Return a value from a function?