Unit 3 Lab: Intermediate Variables

Lesson Objectives

After this lesson, you will be able to…

- Create and floor floats.
- Use special string characters.
- Format strings.

Introducing: Floats

Did you notice that until now, we've only used whole numbers? Whole numbers are integers or, in programming terms, `int`.

Where are all the decimal points?

3.3, 1.1, and 2.2 are all **floats**.

- Short for "floating point value"
- A number with a decimal point. Even 2.0 is a float - it has the decimal!
- Just another numerical variable!

```
an_int = 3 # Int!
a_float = 3.0 # Float!
x = 2.5 # Float!
z = 3.5 + 2.5 # Adding floats - normal math.
y = x + z
print(y) # Prints 8.5.
sum = an_int + a_float # What if we add an int and a float?
print(sum) # Prints 6.0. Adding an int to a float will still make a float!
```

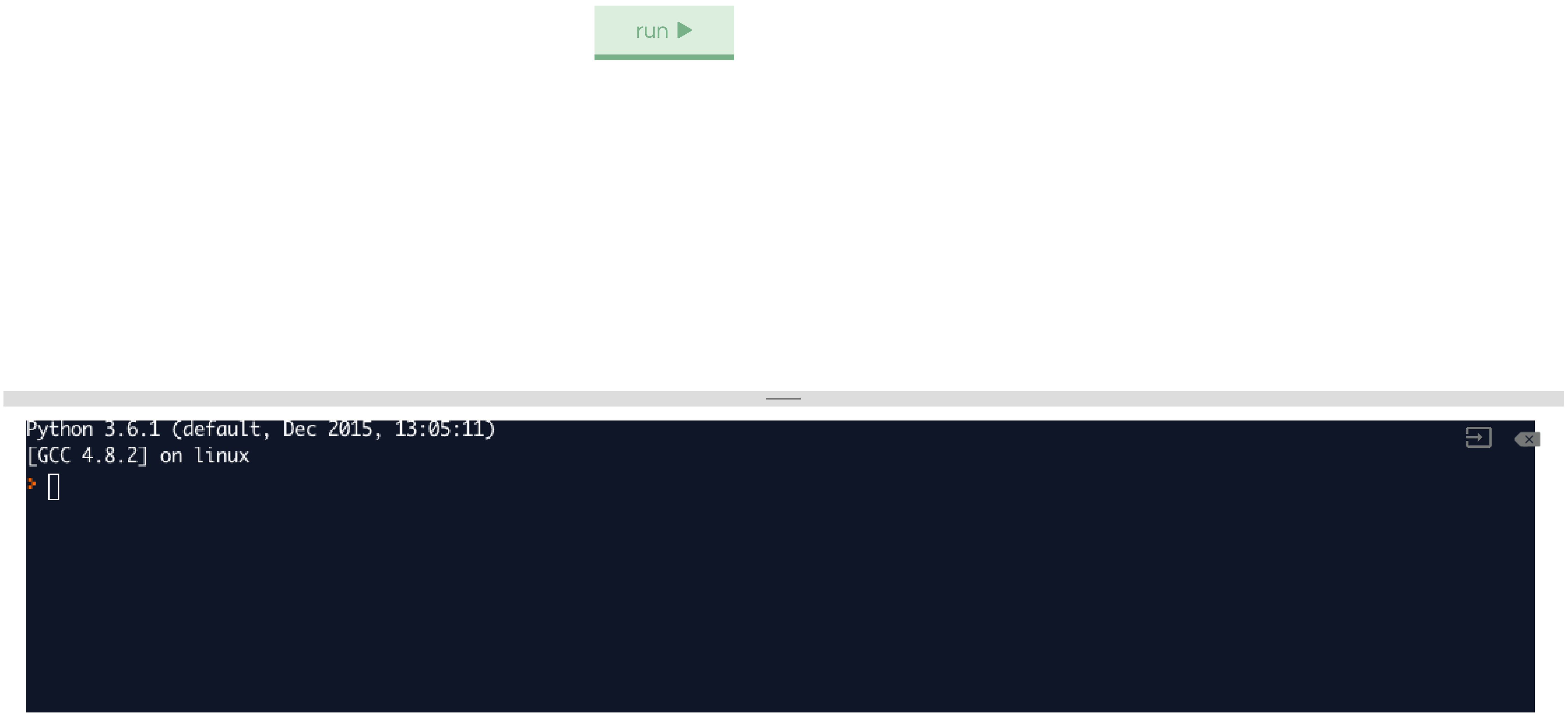
Int / Int == Float ?!

A quotient is not necessarily a whole number! * `5 / 2 == 2.5` * `1 / 3 == 1.333...`

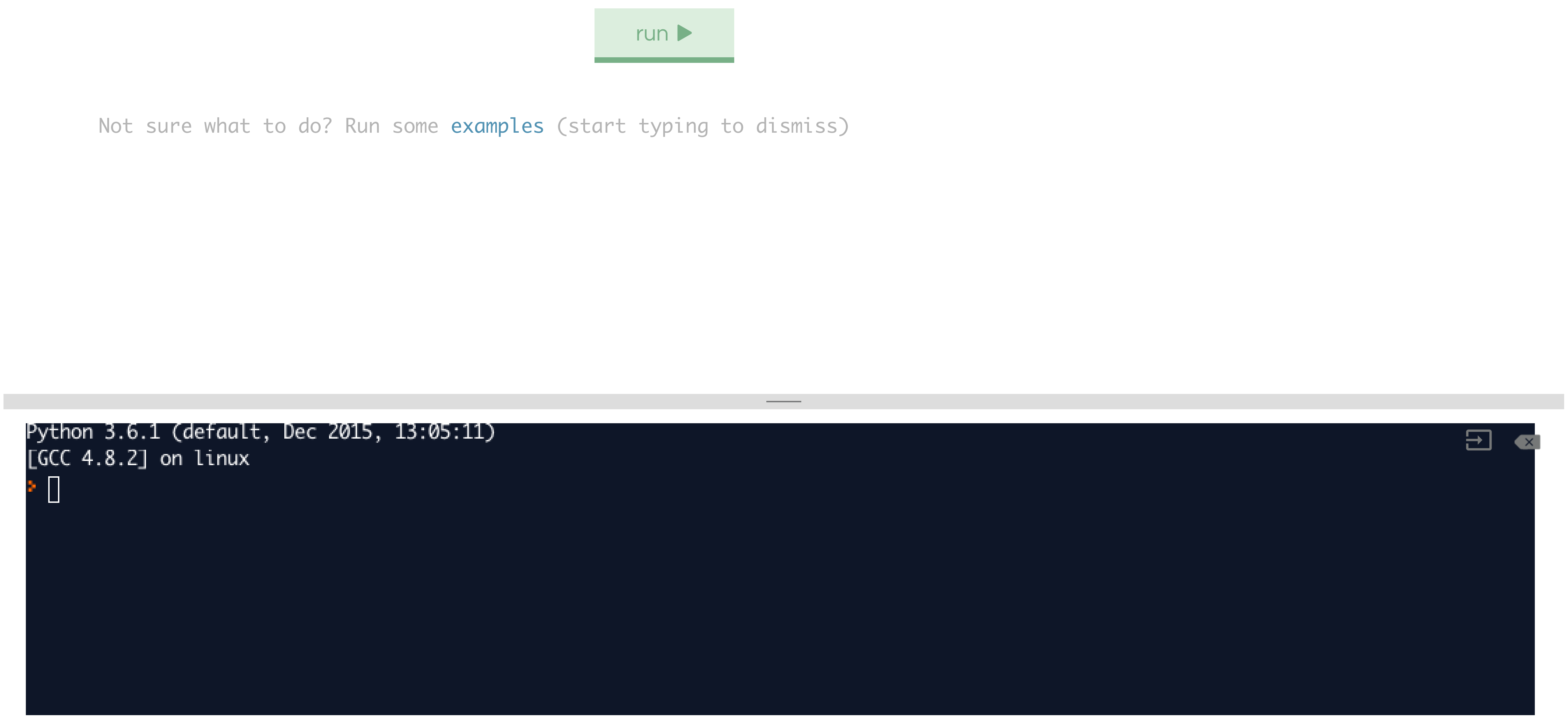
Therefore, quotients are always floats - even when they look like ints. Python doesn't distinguish!

- `6 / 2 == 3.0`
- `8 / 4 == 2.0`

Protip: This is called **implicit type conversion** - Python changed our numbers from ints to floats automatically.



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



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
> 
```

Quick Review: Floats

In programming:

- An *int* is a whole number: `1`, `0`, `-5`.
- A *float* is a number with a decimal point: `1.6`, `-28.2`, `0.0`.
- Doing any math with a float results in a float: `6 + 3.0 = 9.0`.
- Dividing integers results in a float: `4 / 2 = 2.0`

You can use *explicit type conversion* to turn one variable type into another:

- `int()` converts to an integer: `int(6.0) # 6`
- `float()` converts to a float: `float(6) # 6.0`
- `str()` converts to a string: `str(6) # "6"`

Up next: Floor Division.

Finding the Midpoint

One intermediate variable down! Let’s move on past floats.

What if we want to find the middle index of a list?

```
# An odd numbered list (length of 5)
characters = ["Green Arrow", "Super Girl", "The Flash", "Wonder Woman", "Bat


index = len(characters) / 2 # Index is 2.5


print(characters[index]) # There's no element 2.5!
```

We want `2`. Any ideas? This is a very common use case - there must be a way!

Protip: Remember, indexes start at 0!

Introducing Floor Division

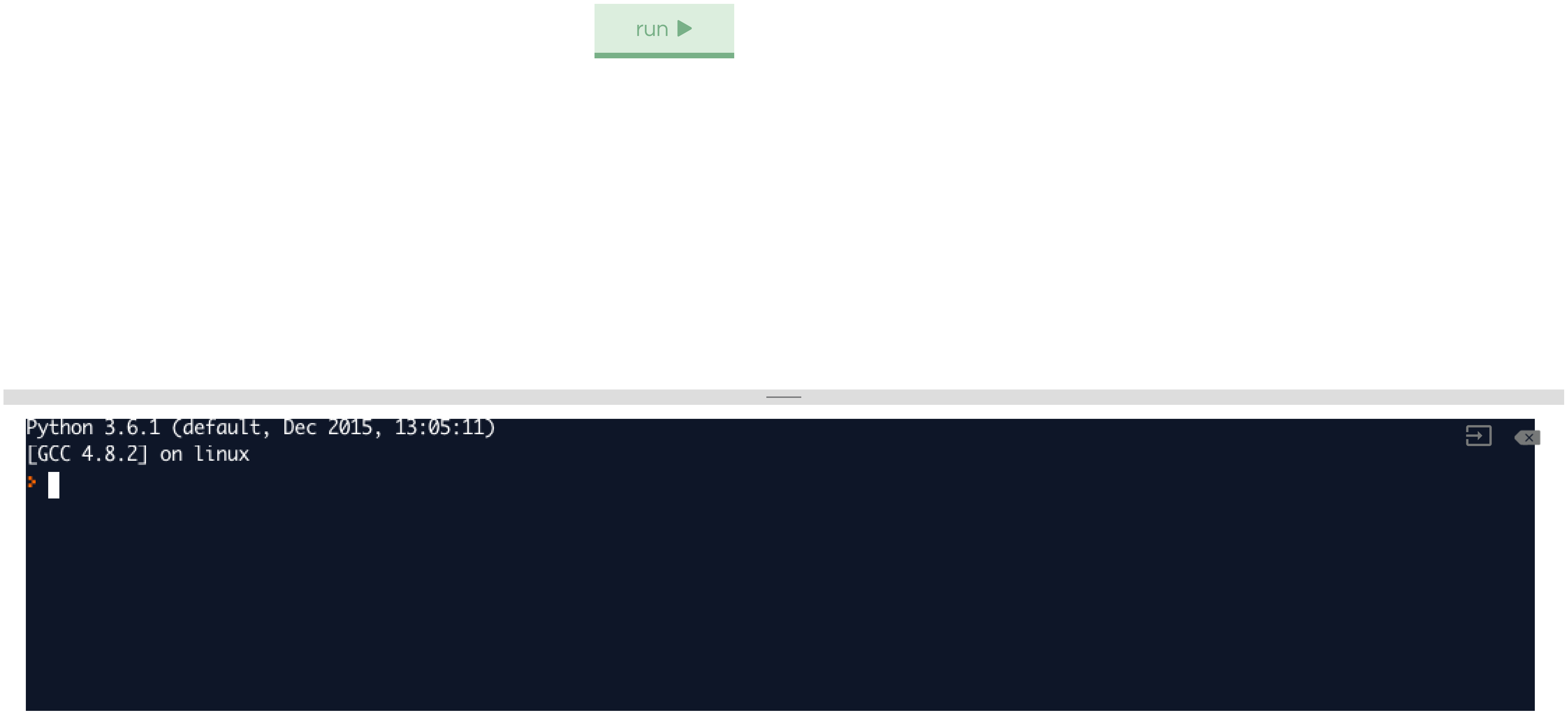
Python has a shortcut.

Floor division (a.k.a. integer division):

- We use `//` instead of just `/`.
- Does normal division, then drops the decimal and returns an int.
- Think of the floor - it's beneath you. We floor by rounding **down**. The decimal is chopped! `2.8` will become `2`, not `3`.

```
# Gives 2.5
float_index = 5 / 2


# Gives 2!
int_index = 5 // 2
```



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

Quick Review:

Floor division:

- Drops the decimal point - always rounds down.
- Performed using `//` instead of just `/`.
- Returns an int instead of a float.

```
# Gives 2.5
regular_division = 5 / 2


# Gives 2!
floor_divison = 5 // 2
```

Next up: Specialty Strings!

Switching Gears: Strings

Our intermediate variables checklist: - Floats - Floor division

What about strings? We might want:

- Printing special characters: A newline, a tab, or a quote inside of a string.
- Formatting
  - A string.
  - The way an integer or float prints out.

Discussion: How would you go about printing a new line between strings, like below?

```
Hello!

This is a line later.
```

Special String Characters

Name	Escape Character	Notes
Newline		Whitespace: Inserts another line
Tab		Whitespace: Inserts a tab
Quote	"	Print a double quote, don't end the string
Backslash	\	Prints \

```
quote = "\"These are not the droids you're looking for.\"\n\n\t-Obi-Wan Kenc


print(quote)
```

This prints, *including* the quotation marks:

```
"These are not the droids you're looking for."



    - Obi-Wan Kenobi
```

String Format

What else with strings?

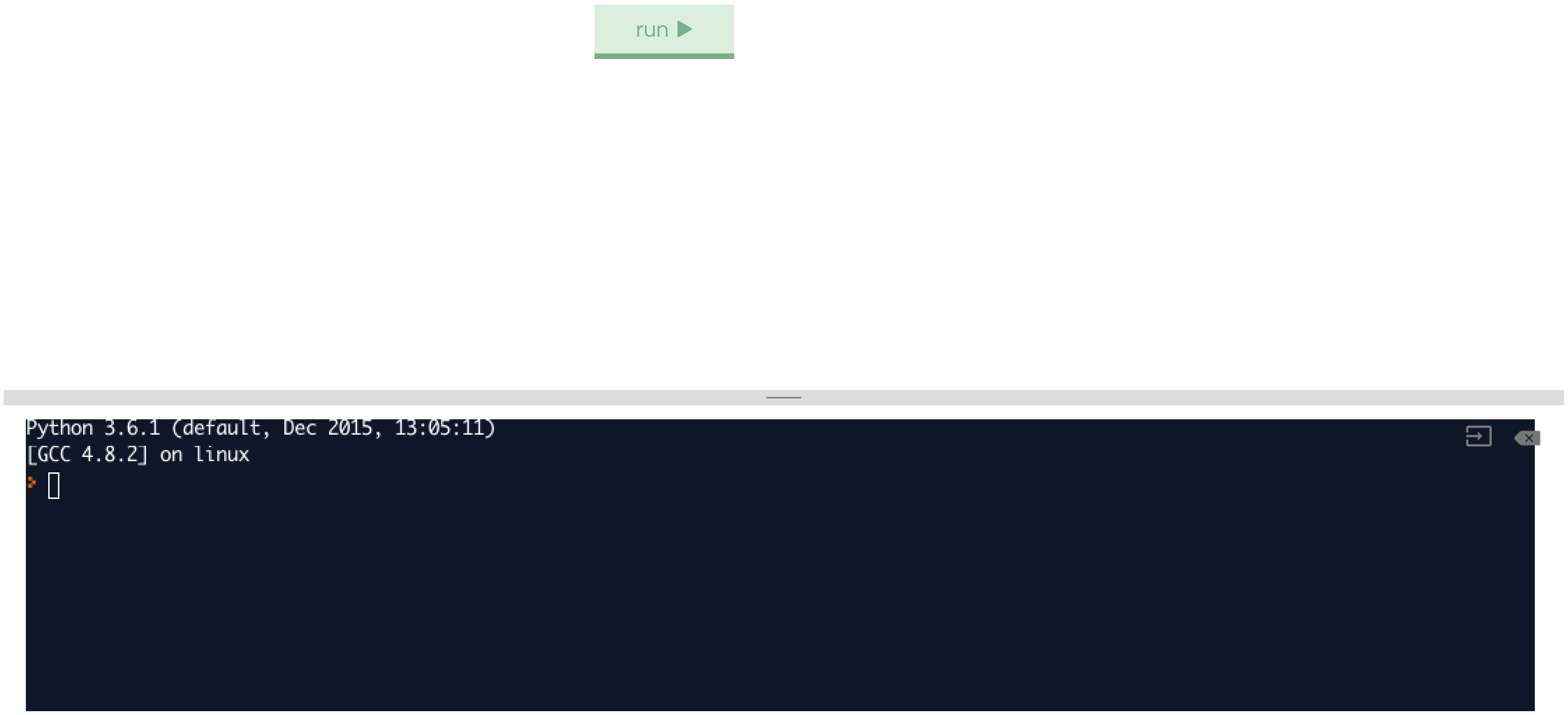
String formatting uses index numbers, in `{}`, as placeholders for strings we later specify in `format`.

Indexes inside the braces refer to the arguments, in order!

```
## Indexes count from 0. ##
x = "{0}, {1}, {2}".format("man", "bear", "pig")
print(x) # prints "man, bear, pig"


## They don't need to be  in order ##
x = "{1}, {0}, {2}".format("man", "bear", "pig")
print(x) # prints "bear, man, pig"


## We can repeat! ##
x = "{0} {1} {0} {1} {0}".format("Hello", "World")
print(x) # prints "Hello World Hello World Hello"
```



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

Quick Review

Special strings:

- A backslash `\` escapes special characters: `\"` will print a quote and `\\` prints a `\`.
- `\n` creates a New line; `\t` creates a Tab.

String formatting:

- Can be used when printing or creating new strings.
- Use `{x}`; `x` corresponds to the number of the argument.

```
x = "{0}, {1}, {2}".format("man", "bear", "pig")
print(x) # prints "man, bear, pig"


x = "{1}, {0}, {2}".format("man", "bear", "pig")
print(x) # prints "bear, man, pig"


x = "{0} {1} {0} {1} {0}".format("Hello", "World")
print(x) # prints "Hello World Hello World Hello"
```

Number Format

What about number formatting?

- Specify a float's precision (how many decimal points are shown).
- Add commas to an integer (so it's more readable!).

```
x = format(1/3, '.2f')
print(x) # Technically, 1/3 is .333333333333. This prints "0.33"
x = format(2.0024292, '.3f')
print(x) # This prints "2.002"
```

```
x = format(5200, ',d')
print(x) # Prints "5,200"
```

Note: Number formatting creates strings!

You Do: Bring It All Together!

- Open a new file and name it “solution.py”.
- Make a dictionary called “sports” with at least 4 key / value pairs.
  - Keys are the names (e.g., tennis, soccer, volleyball).
  - Values are the the number of people that play in a game.
- Use a loop to print out all the keys and values.
  - Output:

```
I like "tennis".

There are usually 2 players in tennis.
```

  - Note the new line and quotes, and use `format` to print out your string!
- BONUS: Every other sport, indent by another tab.
  - 0 tabs: Tennis.
  - 1 tab level: Soccer.
  - 2 tab levels: Volleyball.

HINT: Use floor division for the bonus! `number_of_tabs = loop_counter // 2`

Summary and Q&A

- Floats (`2.52`)
- Floor (`int_index = 5 // 2`) - creates an int.
- Escape characters (`\\`, `\n`, `\r`, `\t`, `\"`)
- Formatting:

```
x = "{0}{1}{0}".format("Hello", "World")
print(x) # prints "HelloWorldHello"


x = format(5200, ',d') # "5,200" -> A string!


x = format(1/3, '.2f') # 0.33
```

- Type conversion:
  - `int()`
  - `float()`
  - `str()`

Additional Resources

- Floating Point (Docs)
- Decimal Module
- Floor Division
- List of Escape Characters
- List of Unicode Characters
- Obscure Unicode Characters
- Unicode Database