Functions

INTRODUCTION TO PYTHON

Hugo Bowne-Anderson
Data Scientist at DataCamp

Functions

- Nothing new! `type()` •
- Piece of reusable

  code • Solves

  particular task • Call

  function instead of

  writing code yourself

Example

```
fam = [1.73, 1.68, 1.71, 1.89]
fam
```

```
[1.73, 1.68, 1.71, 1.89]
```

```
max(fam)
```

```
1.89
```

max()

Example

```python
fam = [1.73, 1.68, 1.71, 1.89]
fam
```

```
[1.73, 1.68, 1.71, 1.89]
```

```python
max(fam)
```

```
1.89
```

Example

[1.73, 1.68, 1.71, 1.89]
max()

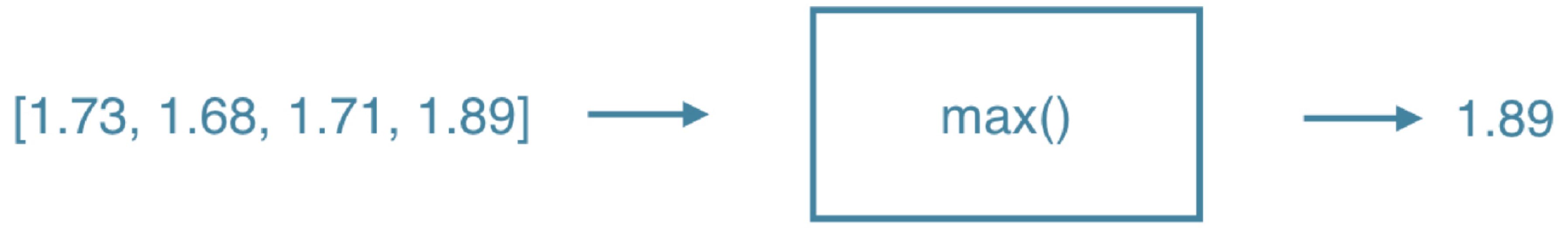
```python
fam = [1.73, 1.68, 1.71, 1.89]
fam
```

```
[1.73, 1.68, 1.71, 1.89]
```

```python
max(fam)
```

```
1.89
```

Example



```
fam = [1.73, 1.68, 1.71, 1.89]
fam
```

```
[1.73, 1.68, 1.71, 1.89]
```

```
max(fam)
```

```
1.89
```

Example

```
tallest = max(fam)
tallest
```

```
1.89
```

round()

```
round(1.68, 1)
```

```
1.7
```

```
round(1.68)
```

```
2
```

```
help(round) # Open up documentation
```

```
Help on built-in function round in module builtins:

round(number, ndigits=None)
    Round a number to a given precision in decimal digits.

    The return value is an integer if ndigits is omitted or None.
    Otherwise the return value has the same type as the number. ndigits may be negative.
```

round()

```
help(round)
```

```
Help on built-in function round in module builtins:

round(number, ndigits=None)
    Round a number to a given precision in decimal digits.

    The return value is an integer if ndigits is omitted or None.
    Otherwise the return value has the same type as the number. ndigits may be negative.
```

round()

```python
help(round)
```

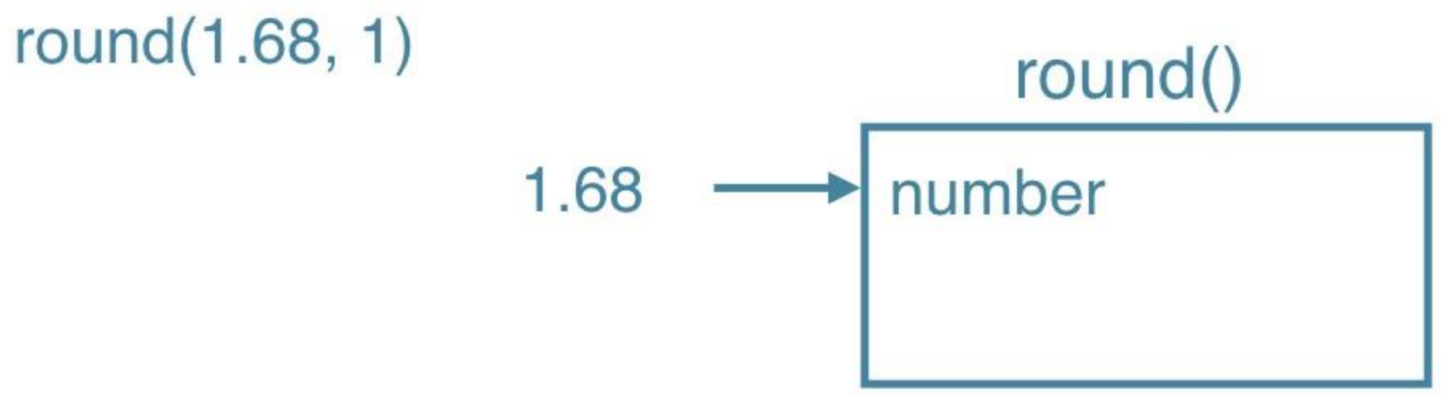
```
round(number, ndigits=None)
    Round a number to a given precision in decimal digits.

    The return value is an integer if ndigits is omitted or None.
    Otherwise the return value has the same type as the number. ndigits may be negative.
```

round(1.68, 1)
round()

round()

```
help(round)
```

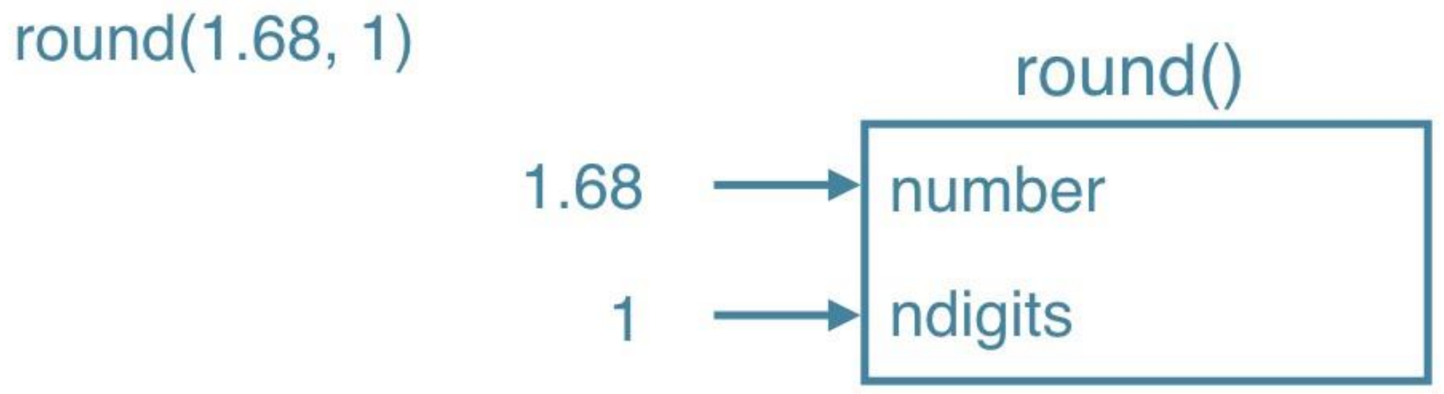
```
round(number, ndigits=None)
    Round a number to a given precision in decimal digits.

    The return value is an integer if ndigits is omitted or None.
    Otherwise the return value has the same type as the number. ndigits may be negative.
```



round()

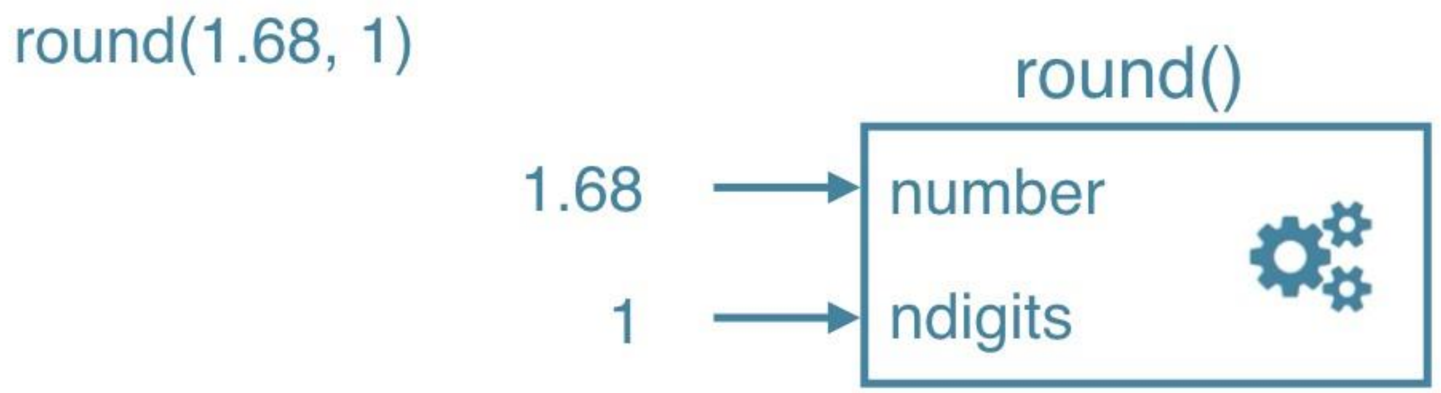
```
help(round)
```

```
    Round a number to a given precision in decimal digits.

    The return value is an integer if ndigits is omitted or None.
    Otherwise the return value has the same type as the number. ndigits may be negative.
```



```python
help(round)
```

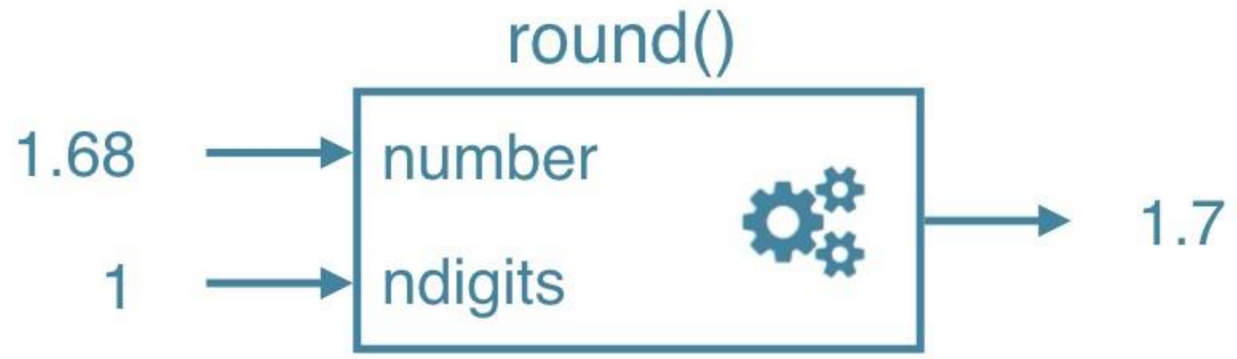
```
    The return value is an integer if ndigits is omitted or None.
    Otherwise the return value has the same type as the number. ndigits may be negative.
```



```
help(round)
```

```
    The return value is an integer if ndigits is omitted or None.
    Otherwise the return value has the same type as the number. ndigits may be negative.
```

round(1.68, 1)



```python
help(round)
```

```
    Otherwise the return value has the same type as the number. ndigits may be negative.
```

```
Help on built-in function round in module builtins:

round(number, ndigits=None)
    Round a number to a given precision in decimal digits.

    The return value is an integer if ndigits is omitted or None.
    Otherwise the return value has the same type as the number. ndigits may be negative.
```

round()

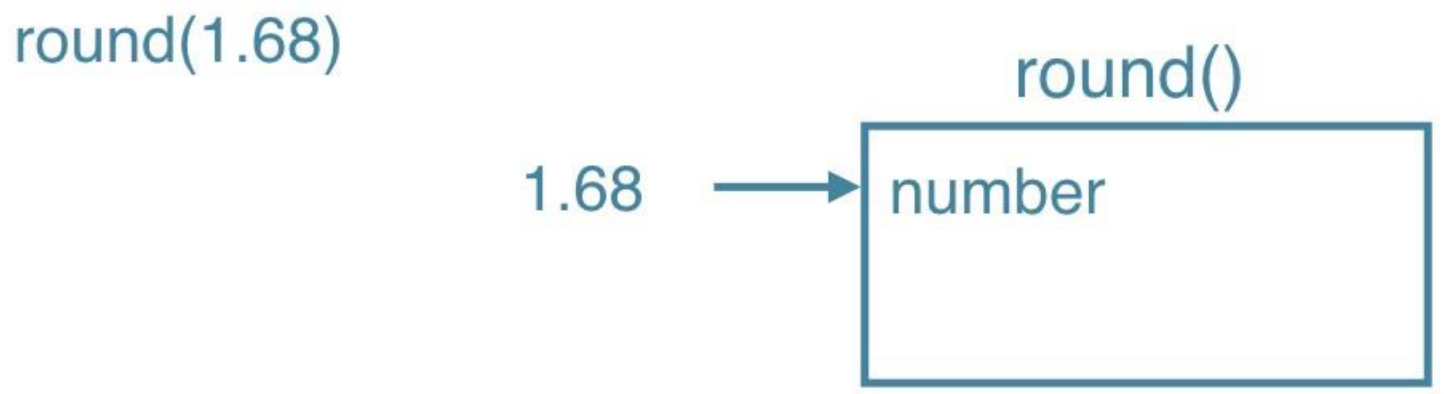
```
help(round)
```

round(1.68)
round()

```
Help on built-in function round in module builtins:

round(number, ndigits=None)
    Round a number to a given precision in decimal digits.

    The return value is an integer if ndigits is omitted or None.
    Otherwise the return value has the same type as the number. ndigits may be negative.
```

round()

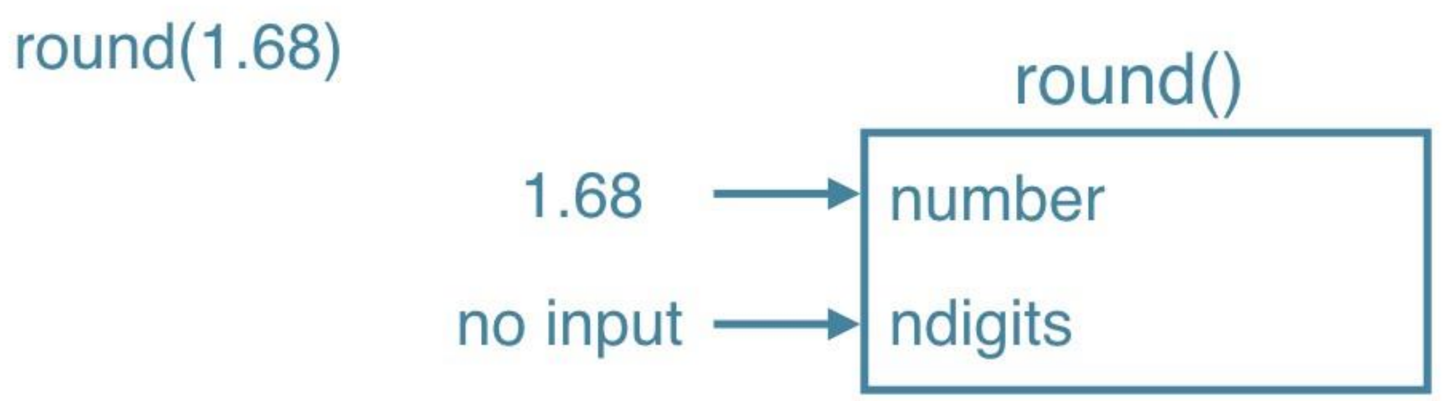
```
help(round)
```



```
Help on built-in function round in module builtins:


round(number, ndigits=None)
    Round a number to a given precision in decimal digits.


    The return value is an integer if ndigits is omitted or None.
    Otherwise the return value has the same type as the number. ndigits may be negative.
```

round()

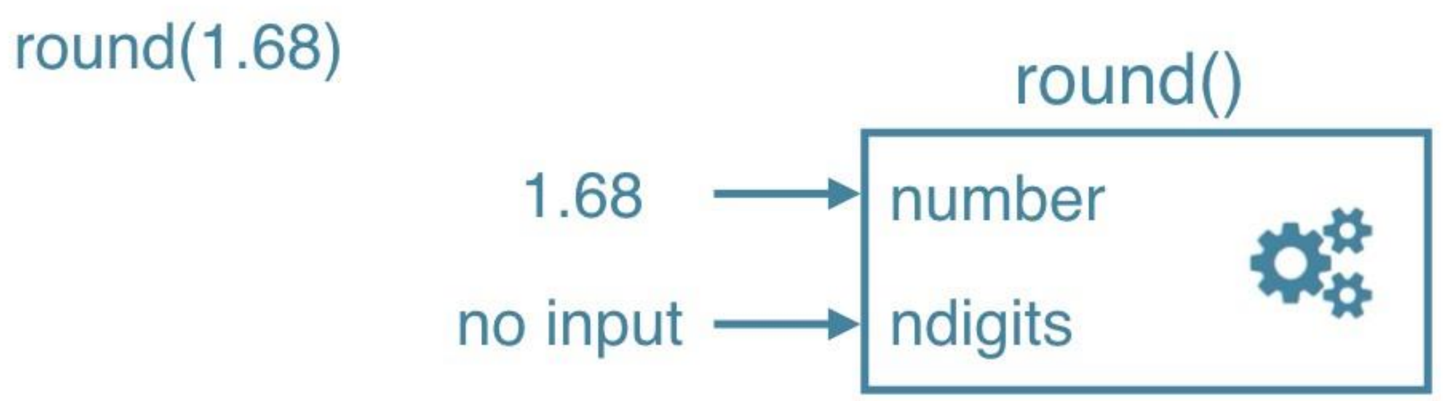
```
help(round)
```



```
Help on built-in function round in module builtins:

round(number, ndigits=None)
    Round a number to a given precision in decimal digits.

    The return value is an integer if ndigits is omitted or None.
    Otherwise the return value has the same type as the number. ndigits may be negative.
```

round()

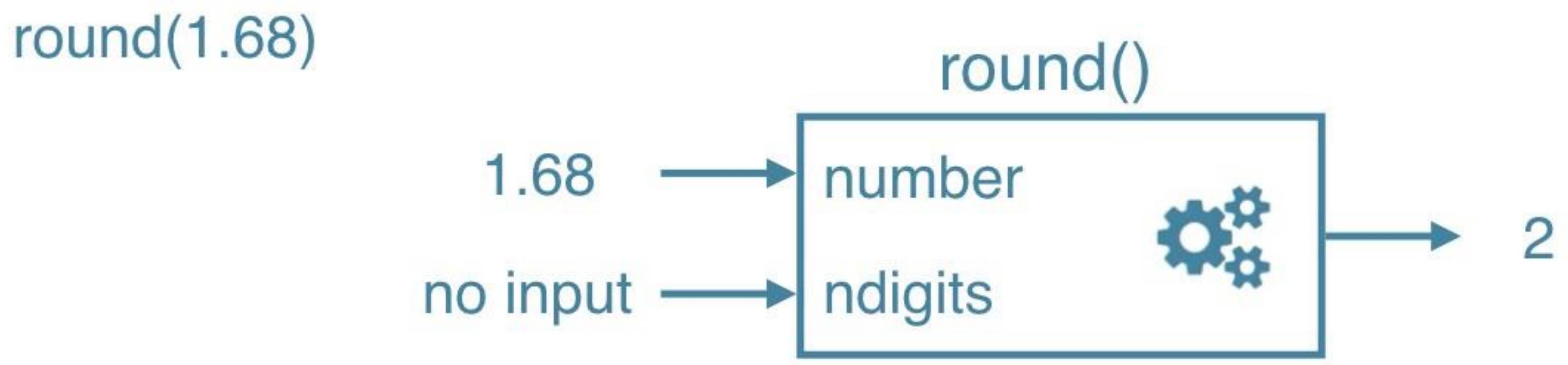
```
help(round)
```



```
Help on built-in function round in module builtins:

round(number, ndigits=None)
    Round a number to a given precision in decimal digits.

    The return value is an integer if ndigits is omitted or None.
    Otherwise the return value has the same type as the number. ndigits may be negative.
```

round()

```
help(round)
```



round()

```
Help on built-in function round in module builtins:

round(number, ndigits=None)
    Round a number to a given precision in decimal digits.

    The return value is an integer if ndigits is omitted or None.
    Otherwise the return value has the same type as the number. ndigits may be negative.
```

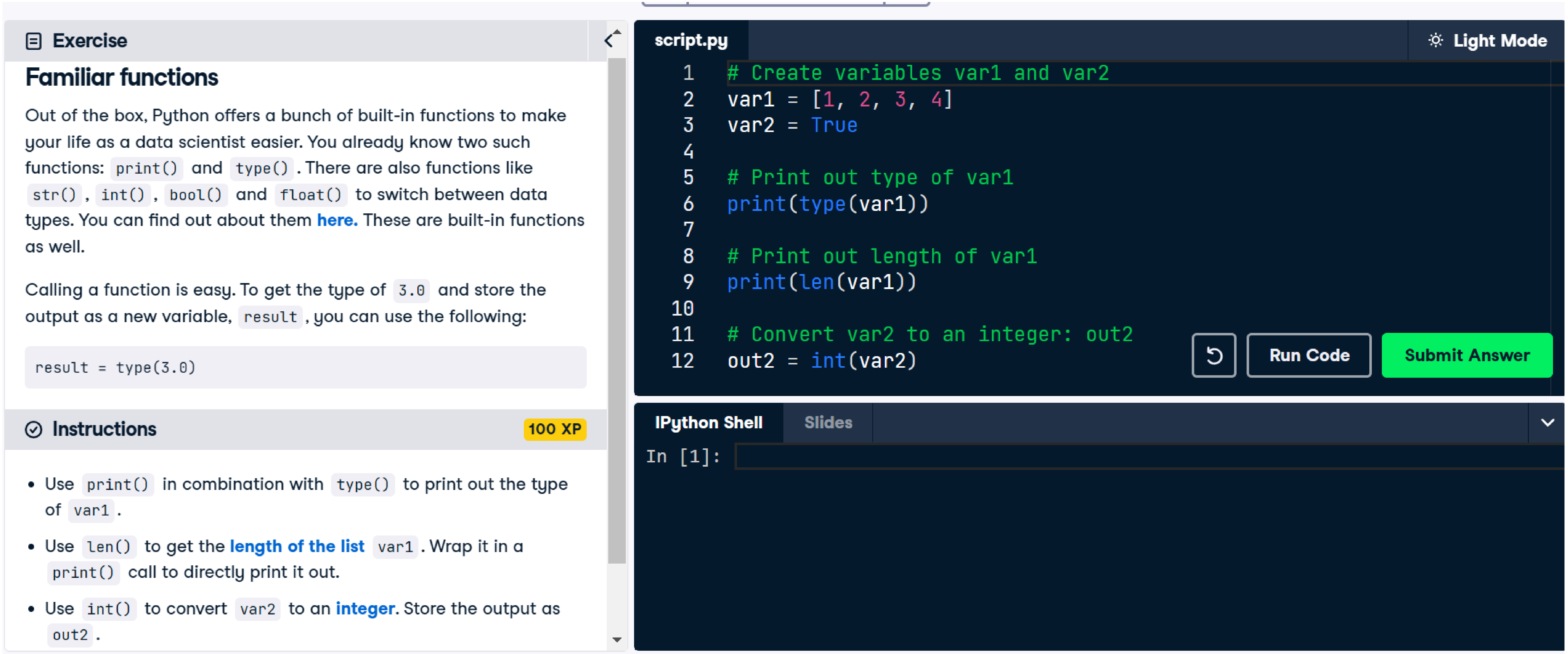
- `round(number)`
- `round(number, ndigits)`

Find functions

- How to know?
    - Standard task -> probably function exists!
- The internet is your friend

Let's practice!

INTRODUCTION TO PYTHON



Familiar functions

Out of the box, Python offers a bunch of built-in functions to make your life as a data scientist easier. You already know two such functions: `print()` and `type()`. There are also functions like `str()`, `int()`, `bool()` and `float()` to switch between data types. You can find out about them here. These are built-in functions as well.

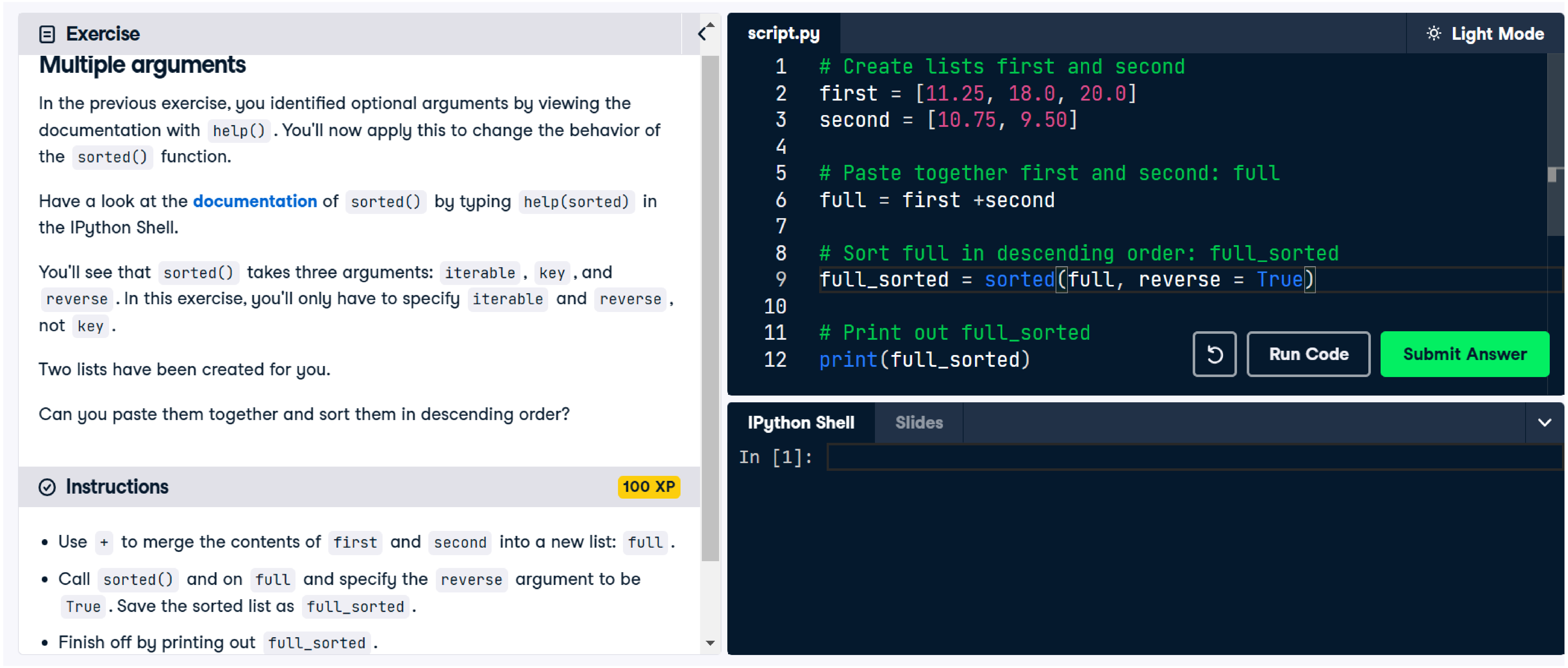
Calling a function is easy. To get the type of `3.0` and store the output as a new variable, `result`, you can use the following:

```
result = type(3.0)
```

- Use `print()` in combination with `type()` to print out the type of `var1`.
- Use `len()` to get the length of the list `var1`. Wrap it in a `print()` call to directly print it out.
- Use `int()` to convert `var2` to an integer. Store the output as `out2`.

```
# Create variables var1 and var2
var1 = [1, 2, 3, 4]
var2 = True

# Print out type of var1
print(type(var1))

# Print out length of var1
print(len(var1))

# Convert var2 to an integer: out2
out2 = int(var2)
```

```
In [1]:
```



Multiple arguments

In the previous exercise, you identified optional arguments by viewing the documentation with `help()`. You'll now apply this to change the behavior of the `sorted()` function.

Have a look at the **documentation** of `sorted()` by typing `help(sorted)` in the IPython Shell.

You'll see that `sorted()` takes three arguments: `iterable`, `key`, and `reverse`. In this exercise, you'll only have to specify `iterable` and `reverse`, not `key`.

Two lists have been created for you.

Can you paste them together and sort them in descending order?

- Use `+` to merge the contents of `first` and `second` into a new list: `full`.
- Call `sorted()` and on `full` and specify the `reverse` argument to be `True`. Save the sorted list as `full_sorted`.
- Finish off by printing out `full_sorted`.

```python
# Create lists first and second
first = [11.25, 18.0, 20.0]
second = [10.75, 9.50]

# Paste together first and second: full
full = first +second

# Sort full in descending order: full_sorted
full_sorted = sorted(full, reverse = True)

# Print out full_sorted
print(full_sorted)
```

```
In [1]:
```

datacamp

Methods

INTRODUCTION TO PYTHON

Hugo Bowne-Anderson
Data Scientist at DataCamp

Built-in Functions

- Maximum of list: max()
- Length of list or string: len()
- Get index in list: ?
- Reversing a list: ?

Back 2 Basics

```
sister = "liz"
```

Object

```
height = 1.73
```

Object

```
fam = ["liz", 1.73, "emma", 1.68,
       "mom", 1.71, "dad", 1.89]
```

Object

```
sister = "liz"
```

```
height = 1.73
```

Back 2 Basics

```
fam = ["liz", 1.73, "emma", 1.68,
       "mom", 1.71, "dad", 1.89]
```

- Methods: Functions that belong to objects

```
sister = "liz"
```

```
height = 1.73
```

	type
Object	str
Object	float
Object	list

Back 2 Basics

```
fam = ["liz", 1.73, "emma", 1.68,
       "mom", 1.71, "dad", 1.89]
```

- Methods: Functions that belong to objects

	type	examples of methods

Object	str	capitalize()
		replace()

Object	float	bit_length()
		conjugate()

Object	list	index()
		count()

list methods

```
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam.index("mom") # "Call method index() on fam"
```

```
4
```

```
fam.count(1.73)
```

```
1
```

str methods

```
sister
```

```
'liz'
```

```
sister.capitalize()
```

```
'Liz'
```

```
sister.replace("z", "sa")
```

```
'lisa'
```

Methods

- Everything = object
- Object have methods associated, depending on type

```
sister.replace("z", "sa")
```

```
'lisa'
```

```
fam.replace("mom", "mommy")
```

```
AttributeError: 'list' object has no attribute 'replace'
```

Methods

```
sister.index("z")
```

```
2
```

```
fam.index("mom")
```

```
4
```

Methods (2)

```
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89]
```

```
fam.append("me")
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89, 'me']
```

```
fam.append(1.79)
fam
```

```
['liz', 1.73, 'emma', 1.68, 'mom', 1.71, 'dad', 1.89, 'me',
1.79]
```

Summary

Functions

```
type(fam)
```

```
list
```

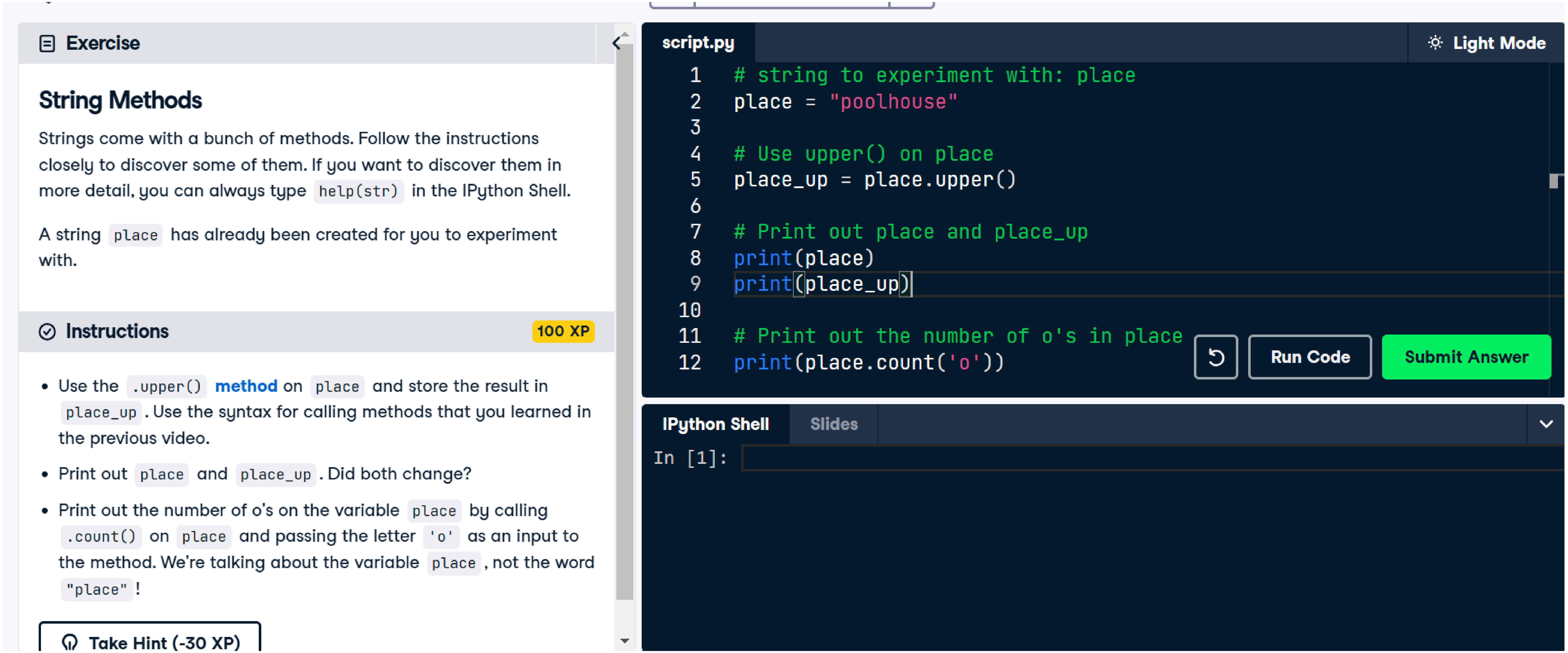
Methods: call functions on objects

```
fam.index("dad")
```

```
6
```

Let's practice!

INTRODUCTION TO PYTHON


Exercise

String Methods

Strings come with a bunch of methods. Follow the instructions closely to discover some of them. If you want to discover them in more detail, you can always type `help(str)` in the IPython Shell.

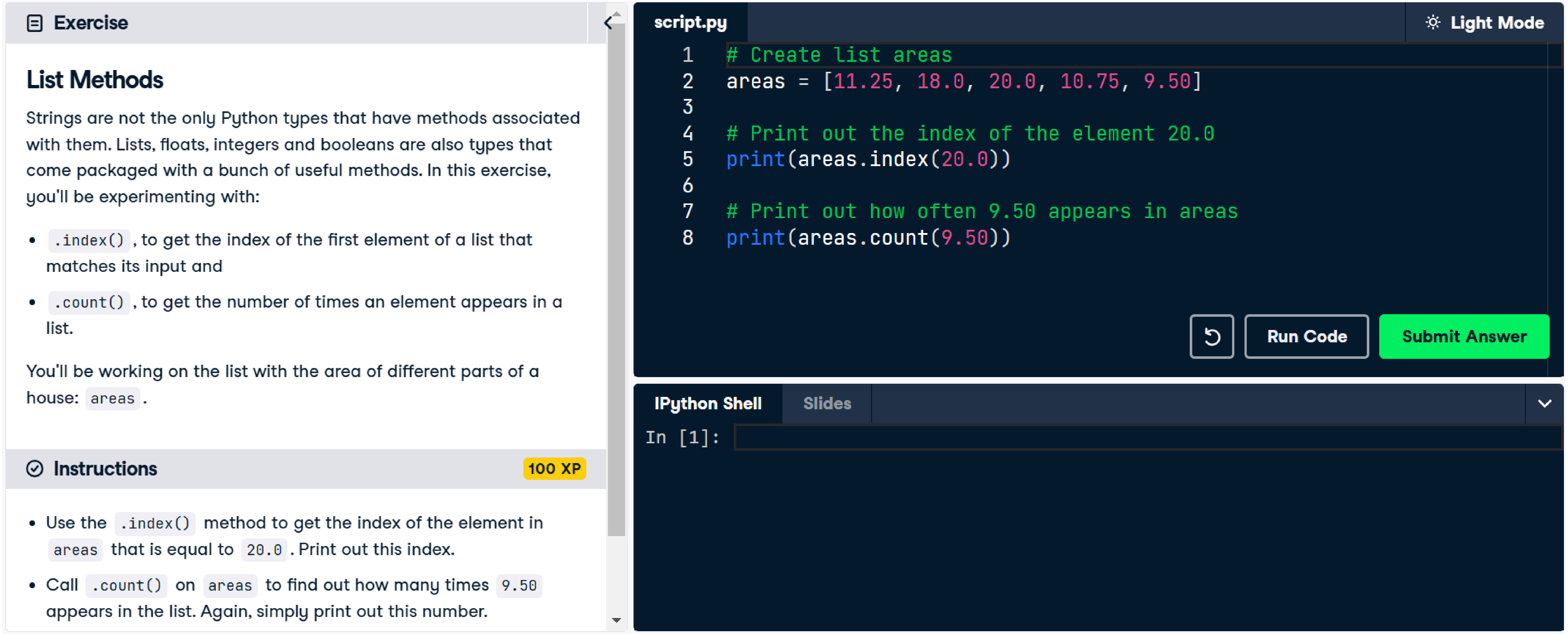
A string `place` has already been created for you to experiment with.

Instructions 100 XP

- Use the `.upper()` **method** on `place` and store the result in `place_up`. Use the syntax for calling methods that you learned in the previous video.
- Print out `place` and `place_up`. Did both change?
- Print out the number of o's on the variable `place` by calling `.count()` on `place` and passing the letter `'o'` as an input to the method. We're talking about the variable `place`, not the word `"place"`!

```python
# string to experiment with: place
place = "poolhouse"

# Use upper() on place
place_up = place.upper()

# Print out place and place_up
print(place)
print(place_up)

# Print out the number of o's in place
print(place.count('o'))
```

```
In [1]:
```



Exercise

List Methods

Strings are not the only Python types that have methods associated with them. Lists, floats, integers and booleans are also types that come packaged with a bunch of useful methods. In this exercise, you'll be experimenting with:

- `.index()`, to get the index of the first element of a list that matches its input and
- `.count()`, to get the number of times an element appears in a list.

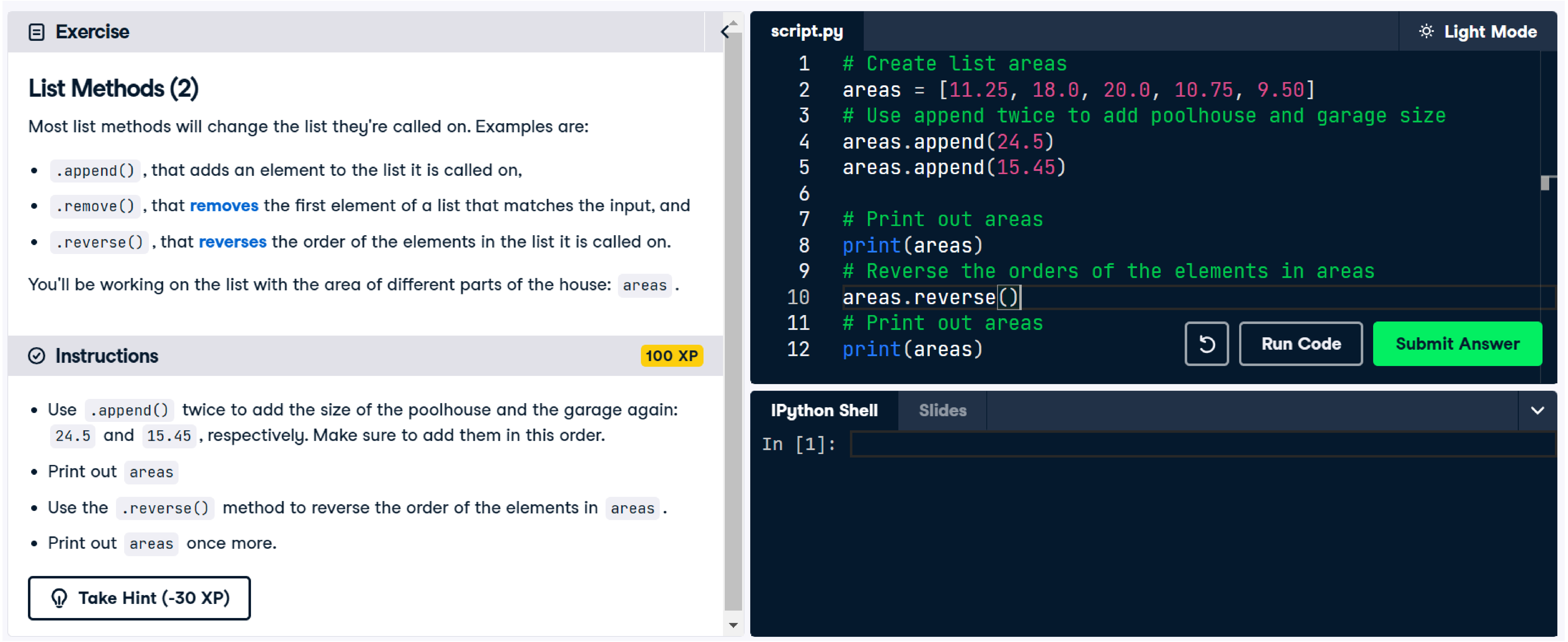
You'll be working on the list with the area of different parts of a house: `areas`.

Instructions 100 XP

- Use the `.index()` method to get the index of the element in `areas` that is equal to `20.0`. Print out this index.
- Call `.count()` on `areas` to find out how many times `9.50` appears in the list. Again, simply print out this number.

script.py

```python
# Create list areas
areas = [11.25, 18.0, 20.0, 10.75, 9.50]

# Print out the index of the element 20.0
print(areas.index(20.0))

# Print out how often 9.50 appears in areas
print(areas.count(9.50))
```



Exercise

List Methods (2)

Most list methods will change the list they're called on. Examples are:

- `.append()`, that adds an element to the list it is called on,
- `.remove()`, that **removes** the first element of a list that matches the input, and
- `.reverse()`, that **reverses** the order of the elements in the list it is called on.

You'll be working on the list with the area of different parts of the house: `areas`.

Instructions 100 XP

- Use `.append()` twice to add the size of the poolhouse and the garage again: `24.5` and `15.45`, respectively. Make sure to add them in this order.
- Print out `areas`
- Use the `.reverse()` method to reverse the order of the elements in `areas`.
- Print out `areas` once more.

script.py

```python
# Create list areas
areas = [11.25, 18.0, 20.0, 10.75, 9.50]
# Use append twice to add poolhouse and garage size
areas.append(24.5)
areas.append(15.45)

# Print out areas
print(areas)
# Reverse the orders of the elements in areas
areas.reverse()
# Print out areas
print(areas)
```

IPython Shell

```
In [1]:
```

datacamp

Packages

INTRODUCTION TO PYTHON

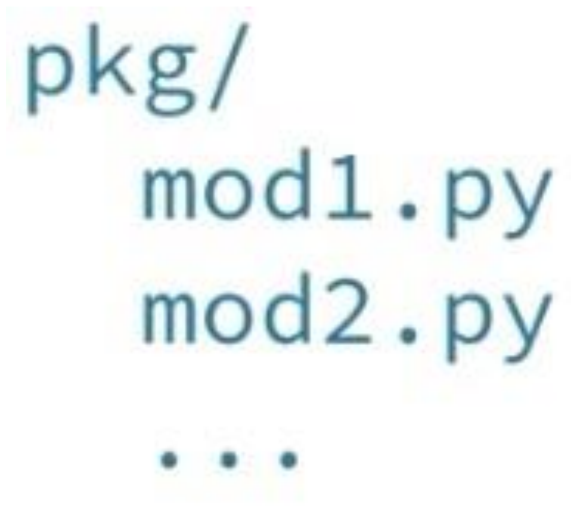
Hugo Bowne-Anderson
Data Scientist at DataCamp

Motivation

- Functions and methods are powerful
- All code in Python distribution?
  - Huge code base: messy
  - Lots of code you won’t use
  - Maintenance problem

Packages

- Directory of Python Scripts
- Each script = module
- Specify functions, methods, types
- Thousands of packages available
  - NumPy
  - Matplotlib
  - scikit-learn



```
pkg/
   mod1.py
   mod2.py
   ...
```

Install package

- https://pip.pypa.io/en/stable/installation/

- Download `get-pip.py`

- Terminal:

- `python3 get-pip.py`
- `pip3 install numpy`

Import package

```python
import
array([1, 2, 3])
```

```
import numpy
```

```python
numpy.array([1, 2, 3])
```

```
array([1, 2, 3])
```

```python
import numpy as np
np.array([1, 2, 3])
```

```
array([1, 2, 3])
```

```python
from numpy import array
array([1, 2, 3])
```

```
array([1, 2, 3])
```

from numpy import array

-

```
from numpy import array

fam = ["liz", 1.73, "emma", 1.68,
    "mom", 1.71, "dad", 1.89]

...
fam_ext = fam + ["me", 1.79]

...
print(str(len(fam_ext)) + " elements in fam_ext")

...
np_fam = array(fam_ext)
```

my_script.py

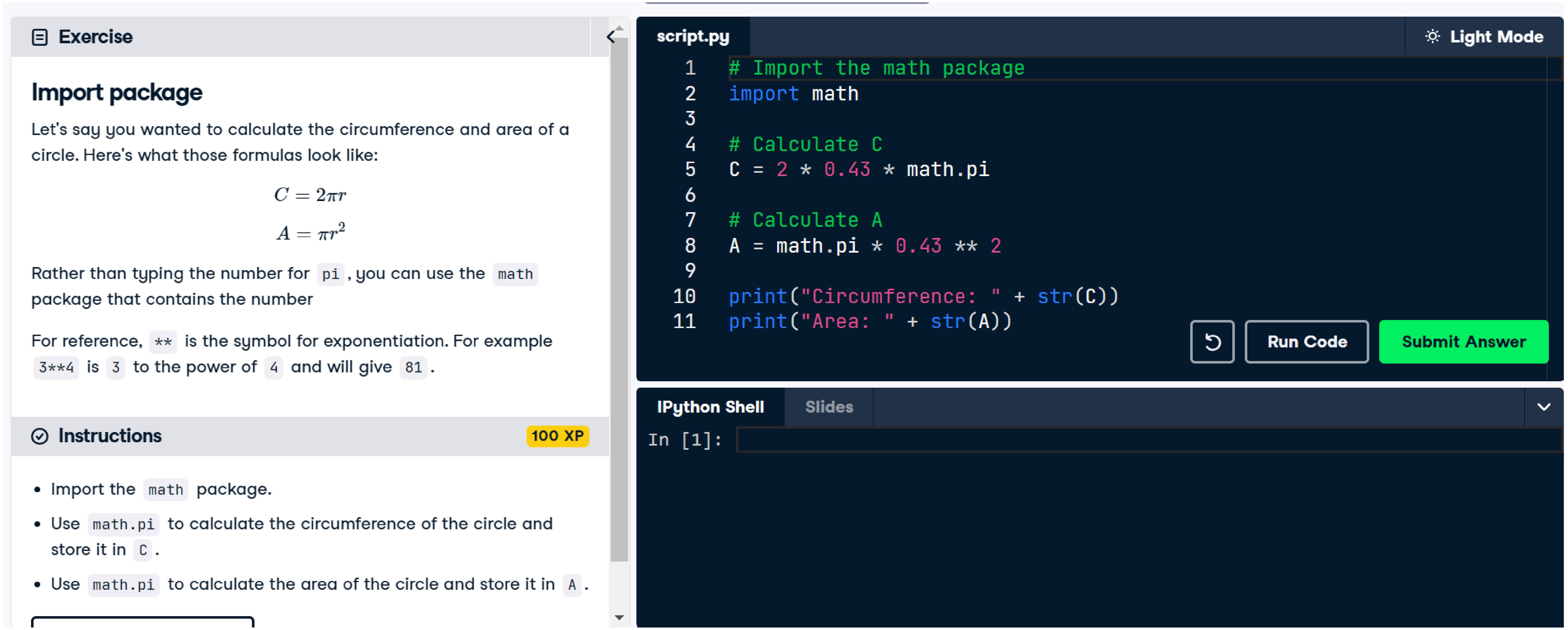
- Using NumPy, but not very clear

import numpy

```
import numpy as np

fam = ["liz", 1.73, "emma", 1.68,
    "mom", 1.71, "dad", 1.89]


...
fam_ext = fam + ["me", 1.79]


...
print(str(len(fam_ext)) + " elements in fam_ext")


...
np_fam = np.array(fam_ext) # Clearly using NumPy
```

Let's practice!

INTRODUCTION TO PYTHON



Import package

Let's say you wanted to calculate the circumference and area of a circle. Here's what those formulas look like:

$$C = 2\pi r$$

$$A = \pi r^2$$

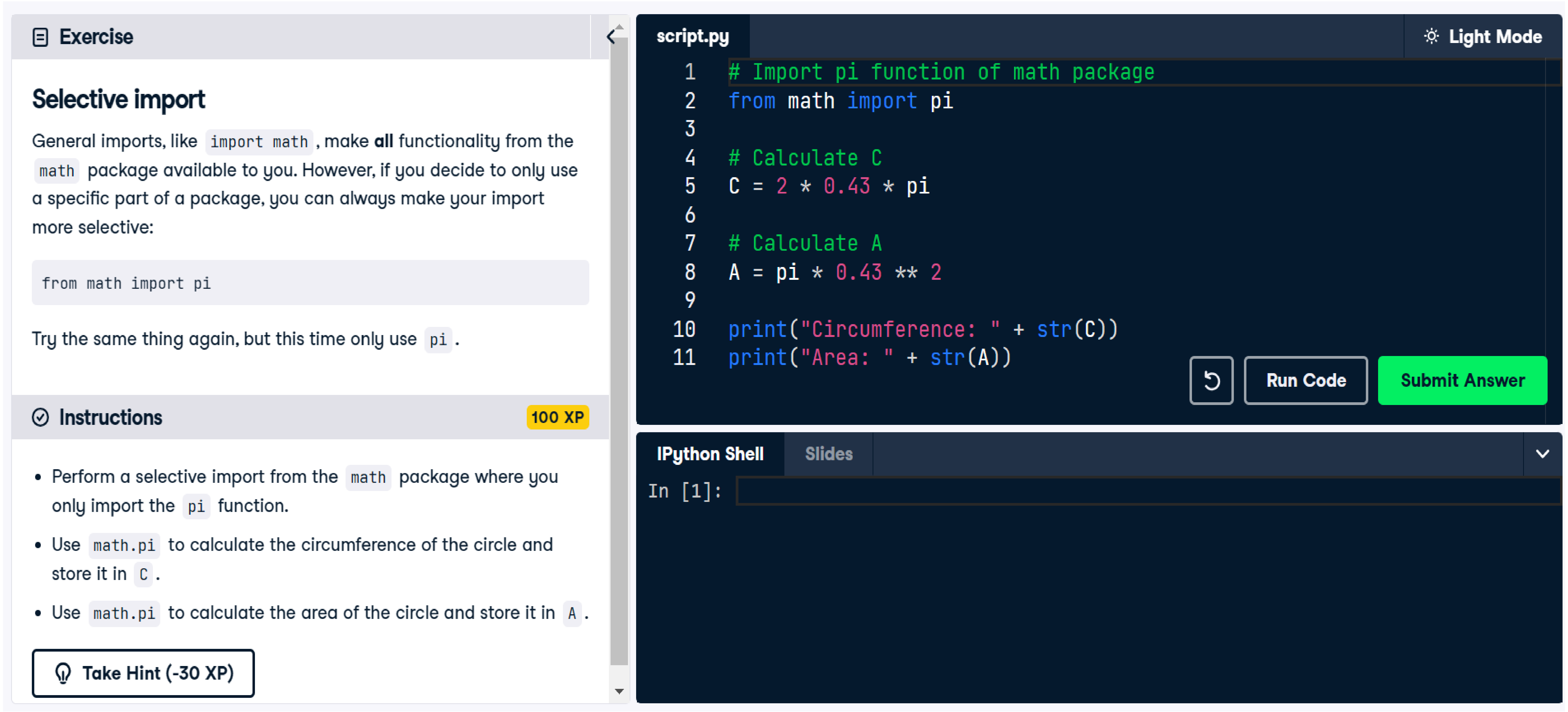
Rather than typing the number for `pi`, you can use the `math` package that contains the number

For reference, `**` is the symbol for exponentiation. For example `3**4` is `3` to the power of `4` and will give `81`.

- Import the `math` package.
- Use `math.pi` to calculate the circumference of the circle and store it in `C`.
- Use `math.pi` to calculate the area of the circle and store it in `A`.

```python
# Import the math package
import math

# Calculate C
C = 2 * 0.43 * math.pi

# Calculate A
A = math.pi * 0.43 ** 2

print("Circumference: " + str(C))
print("Area: " + str(A))
```

```
In [1]:
```



Exercise

Selective import

General imports, like `import math`, make **all** functionality from the `math` package available to you. However, if you decide to only use a specific part of a package, you can always make your import more selective:

```
from math import pi
```

Try the same thing again, but this time only use `pi`.

Instructions 100 XP

- Perform a selective import from the `math` package where you only import the `pi` function.
- Use `math.pi` to calculate the circumference of the circle and store it in `C`.
- Use `math.pi` to calculate the area of the circle and store it in `A`.

```python
# Import pi function of math package
from math import pi

# Calculate C
C = 2 * 0.43 * pi

# Calculate A
A = pi * 0.43 ** 2

print("Circumference: " + str(C))
print("Area: " + str(A))
```

```
In [1]:
```