Exercise 2

Functions are powerful. Try writing some yourself.

As before, don't forget to run the setup code below before jumping into question 1.

In [1]:
```
# SETUP. You don't need to worry for now about what this code does or how it
import sys
from pathlib import Path
learntools_dir = Path().absolute().parents[1]
sys.path.append(str(learntools_dir))
from learntools.core import binder; binder.bind(globals())
from learntools.python.ex2 import *
print('Setup complete.')
```

```
Setup complete.
```

1.

Complete the body of the following function according to its docstring.

HINT: Python has a built-in function `round`.

In [5]:
```
def round_to_two_places(num):
    """Return the given number rounded to two decimal places.

    >>> round_to_two_places(3.14159)
    3.14
    """
    # Replace this body with your own code.
    # ("pass" is a keyword that does literally nothing. We used it as a plac
    # because after we begin a code block, Python requires at least one line
    return round(num, ndigits=2)

# Check your answer
q1.check()
```

Correct

In [4]:
```
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 negati
ve.
```

In [3]:
```
# Uncomment the following for a hint

q1.hint()

# Or uncomment the following to peek at the solution

# q1.solution()
```

Hint: Run `help(round)` in the console (or in a code cell) to learn more about the round function. You'll need to use the function's optional second argument.

2.

The help for `round` says that `ndigits` (the second argument) may be negative. What do you think will happen when it is? Try some examples in the following cell.

In [16]:
```
# Put your test code here
round(123456.142323433443, -3)
```

Out[16]:
```
123000.0
```

Can you think of a case where this would be useful? Once you're ready, run the code cell below to see the answer and to receive credit for completing the problem.

In [ ]:
```
# Check your answer (Run this code cell to receive credit!)
q2.solution()
```

3.

In the previous exercise, the candy-sharing friends Alice, Bob and Carol tried to split candies evenly. For the sake of their friendship, any candies left over would be smashed. For example, if they collectively bring home 91 candies, they'll take 30 each and smash 1.

Below is a simple function that will calculate the number of candies to smash for *any* number of total candies.

Modify it so that it optionally takes a second argument representing the number of friends the candies are being split between. If no second argument is provided, it should assume 3 friends, as before.

Update the docstring to reflect this new behaviour.

In [19]:
```
def to_smash(total_candies, n=3):
    """Return the number of leftover candies that must be smashed after dist
    the given number of candies evenly between 3 friends.

    >>> to_smash(91)
    1
    """
    return total_candies % n

# Check your answer
q3.check()
```

Correct

In [ ]:
```
# q3.hint()
```

In [ ]:
```
# q3.solution()
```

4. (Optional)

It may not be fun, but reading and understanding error messages will be an important part of your Python career.

Each code cell below contains some commented buggy code. For each cell...

1. Read the code and predict what you think will happen when it's run.
2. Then uncomment the code and run it to see what happens. (**Tip**: In the kernel editor, you can highlight several lines and press `ctrl` + `/` to toggle commenting.)
3. Fix the code (so that it accomplishes its intended purpose without throwing an exception)

In [21]:
```
round_to_two_places(9.9999)
```

Out[21]: 10.0

In [23]:
```
x = -10
y = 5
# Which of the two variables above has the smallest absolute value?
smallest_abs = min(abs(x), abs(y))
```

In [25]:
```
def f(x):
    y = abs(x)
    return y
```

```
print(f(5))
```

```
5
```

Keep Going

Nice job with the code. Next up, you'll learn about *conditionals*, which you'll need to **write interesting programs**.
