

Project 3 - Taxi fare

Write a program to calculate a taxi fare:

- allow the user to enter the journey distance in km
- allow the user to enter the number of passengers
- calculate the fare by charging:
 - £2 for each passenger, regardless of distance
 - a further £1.50 per km, regardless of how many passengers
- output the taxi fare

Project 3 - Taxi fare

Add in:

- validate the inputs for distance (1 to 250 km) and passengers (1 to 7). Don't accept values outside of these ranges - the user should have to enter again (hint: while loop(s))
- use sensible error messages
- add a title and welcome message; add a concatenated output which includes how the fare has been made up (e.g. "3 passengers at £2 each going 4 km at £1.50 per km for a total fare of £12.00)
- add in a "late night" extra charge. This will double the fare

Project 3 - Taxi fare

Extension tasks:

- A. the fare needs to be doubled if it's a Sunday and multiplied by 1.5 if it's a Saturday. The late night charge applies on top of this
- B. ensure that the program doesn't crash if the wrong type of value is entered - e.g. passengers should be an integer; if the user enters 1.5 the program should not crash and they should be told what their error is (hint: try-except)

Project 3 - Taxi fare

The rest of these slides are at Grade 8-9
(OK, maybe 6 or 7 if you want a
challenge)

Proceed at your own risk

Project 3 - Taxi fare

Further Extension:

The distance should be a float data type. It should then be rounded **up** to the next kilometre (so, 6.8 = 7; 7 = 7; 7.1 = 8)

This is hard to do and requires integer division and modulus division. Try:

```
distance = distance // 5 + (distance % 5 > 0)
```

(see next slide for explanation...)

Project 3 - Taxi fare

```
distance = distance // 1 + (distance % 1 > 0)
```

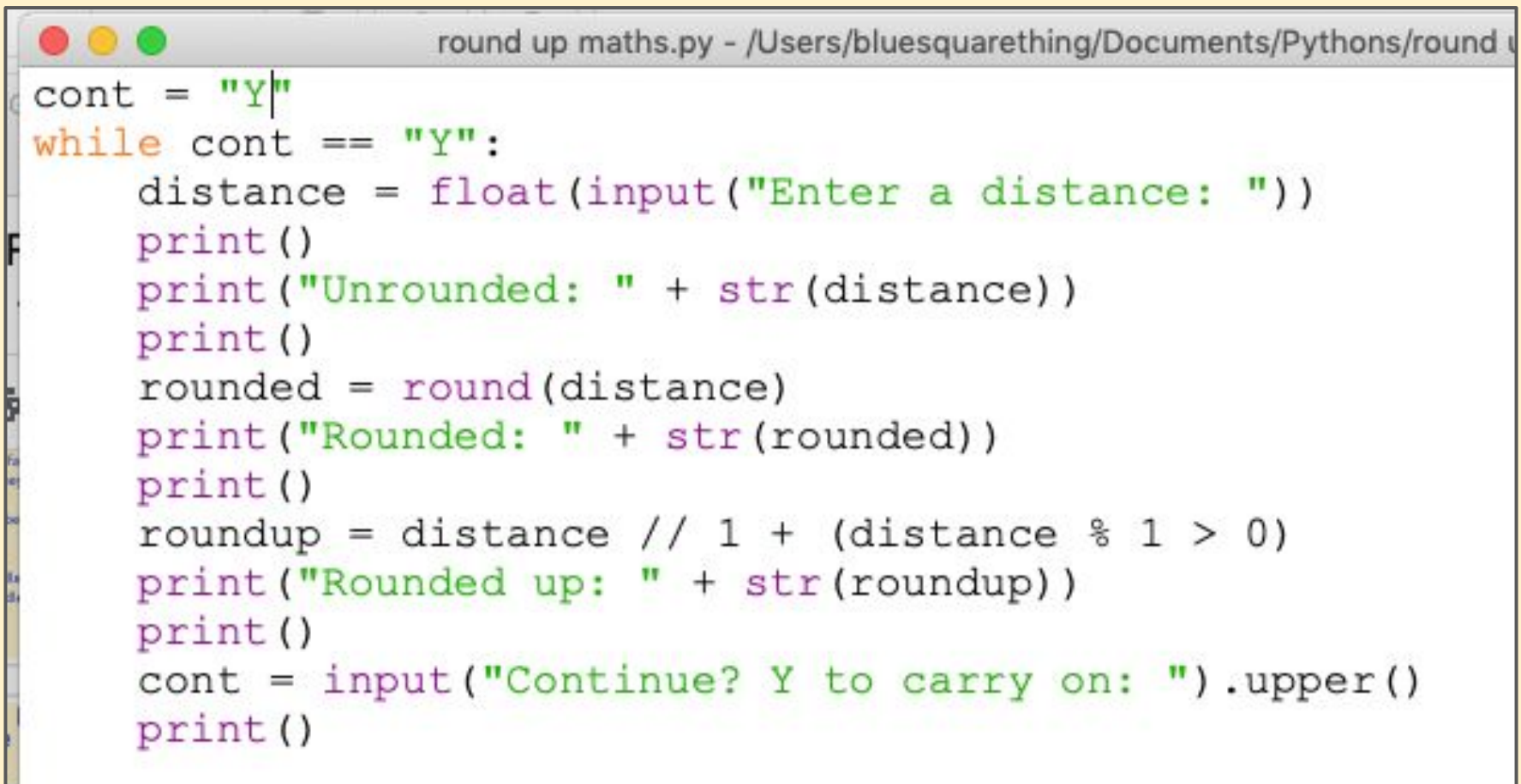
// is integer division. This gives the whole number but no decimal value. So, $7.4 // 1$ divides 7.4 by 1, but discards the .4 bit to give just 7.

% is modulus division - which returns just the remainder. $7.4 \% 1$ returns .4. This is > 0 so the **logical** outcome of $(distance \% 1 > 0)$ is **True** - a **Boolean** value.

Boolean True has the mathematical value 1, so if the remainder is anything other than 0, 1 gets added at this point - which has the effect of rounding everything up!

Project 3 - Taxi fare

Here's a screenshot of a program I wrote to test this:

A screenshot of a text editor window titled "round up maths.py - /Users/bluesquarething/Documents/Pythons/round u". The window contains Python code for a program that asks for a distance, rounds it, and asks if the user wants to continue. The code is as follows:

```
cont = "Y"
while cont == "Y":
    distance = float(input("Enter a distance: "))
    print()
    print("Unrounded: " + str(distance))
    print()
    rounded = round(distance)
    print("Rounded: " + str(rounded))
    print()
    roundup = distance // 1 + (distance % 1 > 0)
    print("Rounded up: " + str(roundup))
    print()
    cont = input("Continue? Y to carry on: ").upper()
    print()
```

Project 3 - Taxi fare

There is another way using a library:

```
import math #at the top of the program
distance = math.ceil(distance)
```

This uses a **function library** called `math`. Just like `random` or `time`, it needs to be imported before it can be used.

Function libraries are really helpful - essentially someone else has done the hard work and produced functions which have been tested and error checked and will work reliably.

But you do need to know about integer and modulus division anyway... `Ceil` stands for **ceiling** by the way... (and returns an integer always, not a float)