Solutions

Marking: **M1** a correct method, **A1** the correct answer, **B1** an independent mark.

Each answer shows the steps, then the **result**.

2.1

`3.5`, `float`

`3`, `int`

`-4`, `int`: floor division rounds down, not towards zero

`1`, `int`

2.2

`3333`: a string times an int repeats it

`7`

`34`: two strings are joined

2.3

```
price = float(input("Price? "))
```

```
print(f"£{price:.2f}")
```

`ValueError`

2.4

0.1 and 0.2 cannot be stored exactly in binary, so the sum is very slightly above 0.3

`abs((0.1 + 0.2) - 0.3) < 1e-9` (or `math.isclose`)

3.1

(a) `2h 15m`

(b) `0h 59m`

(c) `-1h 55m`: `//` rounds down to -1, and `%` then gives $-5-(-60)=55$

(d) a `ValueError`: `int()` cannot read a string with a decimal point

(e) `int`

3.2

(a) `TypeError`: both inputs are strings, and a string cannot be multiplied by a string

(b)

```
price = float(input("Price? "))
qty = int(input("Qty? "))
print(f"Total: {price * qty:.2f}")
```

price converted with `float`; quantity with `int`; f-string with `:.2f`

(c) `Total: 7.50`

3.3

(a) `0.9999999999999999 False`: each 0.10 is stored with a tiny binary error, and the errors add up

(b)

```
total = 0
for _ in range(10):
    total += 10
print(f"£{total // 100}.{total % 100:02d}")
```

counts in `int` pence; pounds by `// 100`; pence by `% 100` padded with `:02d`

(c) whole pence: money must add up exactly, and equality tests and audits must not fail; a float is fine for a measurement such as an average temperature, where tiny errors do not matter

exercises.lol exercises.com.cn