

Data Types, Operators, Logic & Decisions

Overview: what you'll produce

By the end of this lab you will have a single folder, `code/labs/lab-02/`, containing five working Python scripts, each one runnable on its own from the terminal (plus one optional, harder script if you want an extra challenge):

- `types.py` - checks and converts between Python's basic data types
- `operators.py` - explores how arithmetic operators behave differently depending on type
- `booleans.py` - stores Boolean answers and combines comparisons
- `decisions.py` - uses `if` / `elif` / `else` to make a program choose between paths
- `ticket_price.py` - a small mini-project pricing a cinema ticket based on age and day
- `parcel_cost.py` (optional, harder) - a bigger mini-project for early finishers or independent study

Nothing here is handed in — it's practice, so work at your own pace and don't worry about getting every exercise perfect. Some students will get through this faster than others depending on how much Python experience they already have, and that's expected. If you don't finish everything in the three hours, use your independent study time to complete the remaining exercises.

Before you start

This lab builds on Lab 1 — you should already have a `code/labs/` folder from last time. Save every script inside a new folder called `code/labs/lab-02/` to keep things organised. Check that:

- ☐ You completed Lab 1, and `code/labs/lab-01/` still exists
- ☐ You can open a terminal and VS Code, and remember `cd` to move between folders

Lab objectives

- Recognise Python's basic data types (`int`, `float`, `str`, `bool`) and check them with `type()`
- Convert between types explicitly using `int()`, `float()`, `str()` and `bool()`
- Use arithmetic, comparison and augmented-assignment operators, and see how the same operator can behave differently depending on type
- Store Boolean answers in variables and combine conditions with `and` / `or` / `not`.
- Use `if` / `elif` / `else` to make a program choose between different paths

Timing guide

Time	Exercise	Focus
0:00 – 0:10	Setup	Create working folder
0:10 – 0:35	Exercise 1	Types and conversion
0:35 – 1:00	Exercise 2	Operators in context
1:00 – 1:25	Exercise 3	Booleans and comparisons
1:25 – 1:35	Break	—
1:35 – 2:10	Exercise 4	Decisions with if / elif / else
2:10 – 2:50	Exercise 5	Mini project: Cinema Ticket Price
2:50 – 3:00	Wrap-up	Self-check, stretch goals

A note on types

Every value in Python has a type, and the type determines what you're allowed to do with it. A number can be added to another number; two pieces of text can be joined together; but adding a number to a piece of text makes no sense to Python, and it will refuse with an error rather than guess what you meant. You've already used the four basic types without necessarily naming them: whole numbers are `int`, numbers with a decimal point are `float`, text is `str`, and a True/False answer is `bool`. You can always check what type a value has with Python's built-in `type()` function — that's exactly what Exercise 1 has you do.

Exercise 1 — Types and conversion (25 min)

Create a new file called `types.py` for this exercise.

1. Store an integer, a float, a string and a Boolean in four separate variables.
2. Print each variable's type using `type()`. Syntax hint: `print(type(your_variable))`.
3. Python shows types as `<class 'int'>` and so on — read the output and match each one back to the variable that produced it.
4. Ask the user to enter their age with `input()`, and print its type. Notice it's always `str`, even though it looks like a number.
5. Convert that input to an `int` using `int()`, then print its new type.

6. Try converting the text `"hello"` to a number with `int("hello")`, run it, and read the error Python gives you. This is why converting real user input needs care.
7. Use `str()`, `float()` and `bool()` on a couple of other values, printing each result's type to confirm the conversion worked.

Checkpoint

`types.py` should print the type of an int, a float, a string and a Boolean, show that `input()` always returns a string, and demonstrate converting between types with `int()`, `float()`, `str()` and `bool()`.

Stretch goal: use `bool()` on `0`, `1`, `""` and `"hello"` and see which count as `True` and which as `False` — a quirk of Python worth knowing.

Exercise 2 — Operators in context (25 min)

Create a new file called `operators.py` for this exercise.

1. Store two numbers in variables and use `+`, `-`, `*`, `/` and `%` between them, printing each result.
2. Use `**` to raise one number to the power of another, and print the result.
3. Store two strings and use `+` to join them together. Notice this is a completely different behaviour from `+` on numbers.
4. Try using `-` between two strings, run it, and read the error Python gives you. Not every operator makes sense for every type.
5. Use augmented assignment to update a running total step by step, printing it after each change.
Syntax hint: `total += 5` is short for `total = total + 5`.

Checkpoint

`operators.py` should show `+`, `-`, `*`, `/`, `%` and `**` working on numbers, `+` concatenating two strings, at least one operator producing an error between incompatible types, and a running total updated with augmented assignment.

Stretch goal: try `"ha" * 3` and see what happens — a case where an operator behaves usefully across two different types.

— 10 minute break —

Exercise 3 — Booleans and comparisons (25 min)

Create a new file called `booleans.py` for this exercise.

1. Store `True` and `False` in two variables, and print their type with `type()` — confirm `bool` is a type of its own, alongside `int`, `float` and `str`.
2. Compare two numbers using `<`, `>`, `==` and `!=`, printing each result. Store one comparison's result in a variable instead of printing it directly. Syntax hint: `result = a < b`.
3. Ask the user to enter two numbers, convert them, and compare them using `>=` and `<=`, printing which is larger (or whether they're equal).
4. Combine two comparisons with `and` to check whether a number falls between two bounds. Syntax hint: `lower < x and x < upper`.
5. Combine two comparisons with `or` to check whether a number falls outside a safe range. Syntax hint: `x < lower or x > upper`.
6. Use `not` to flip a Boolean result and print the flipped value. Syntax hint: `not some_condition`.

Checkpoint

`booleans.py` should store and print at least one Boolean directly, use each comparison operator (`<`, `>`, `==`, `!=`, `>=`, `<=`) at least once, and combine comparisons using `and`, `or` and `not`.

Stretch goal: predict the result of `5 > 3 and 2 > 4` before running it, then check your answer.

Exercise 4 — Decisions: `if`, `elif`, `else` (35 min)

Create a new file called `decisions.py` for this exercise.

1. Ask the user to enter a number and convert it. Use `if` / `else` to print whether it's positive or not. Syntax hint:

```
if condition:
    ...
else:
    ...
```

2. Extend it to `if` / `elif` / `else` to separately report positive, negative, or zero. Syntax hint:

```
if condition1:
    ...
elif condition2:
    ...
else:
    ...
```

3. Ask for the user's age and use `if` / `elif` / `else` to print a category: child (under 13), teenager (13-17), or adult (18+). Put the lowest bound first — `elif` only runs once everything above it has failed.

4. Combine two conditions in a single `if` using `and` — for example, check that a number is both even and positive. Syntax hint: `if condition1 and condition2:`.

Checkpoint

`decisions.py` should use `if` / `elif` / `else` to sort a number into at least three categories, and combine two conditions with `and` inside a single `if`.

Stretch goal: add an `elif` branch that uses `or` to catch two separate special cases in one line.

Exercise 5 — Mini project: Cinema Ticket Price (40 min)

Bring together everything from this lab into one program, `ticket_price.py`, that works out the price of a cinema ticket.

Requirements:

- Ask for the customer's age, converting it to a number
- Ask whether it's a weekday (yes/no), and turn the answer into a Boolean, e.g. `answer.lower() == "yes"`
- Use `if` / `elif` / `else` on age to set a base price: under 5 free, 5-15 £6.50, 16-64 £12.50, 65+ £9.00
- If it's a weekday, apply a 10% discount using augmented assignment (`price *= 0.9`)
- Print the final price formatted to two decimal places
- Organise the program's logic inside a `main()` function, called at the bottom of the file

Example run:

```
Enter age: 15
Is it a weekday? (yes/no): yes
Your ticket costs £5.85 (weekday discount applied)
```

Checkpoint

Running `ticket_price.py` should ask for age and whether it's a weekday, apply the correct age-based price band, apply a 10% discount on weekdays, and print a correctly formatted final price, with its logic organised inside `main()`.

Exercise 6 — Mini project: Parcel Delivery Cost Calculator (optional)

This one is optional — a harder mini-project for early finishers or for independent study, going beyond the core three-hour session. It isn't in the timing guide above.

Bring together everything from this lab into one program, `parcel_cost.py`, that works out the cost of shipping a parcel. It uses the same ideas as Exercise 5, but combines more of them, with more branches and more conditions layered together.

Requirements:

- Ask for the parcel's weight in kg, converting it to a number
- Ask for the destination zone ("local", "national" or "international"), matched without worrying about capitalisation (hint: `answer.lower()`)
- Use `if` / `elif` / `else` on the zone to set a rate per kg: local £0.80, national £1.50, international £4.00 — and if the zone doesn't match any of these, print a clear error message instead of guessing a price
- Calculate a base cost as `weight * rate`
- Ask whether the parcel needs express delivery (yes/no); if so, add a flat £5.00 surcharge using augmented assignment
- Ask whether the sender wants insurance (yes/no). If they do, ask for the parcel's declared value, then add £3.50 if that value is over £100, or £1.50 otherwise — this needs a decision inside a decision
- Ask for an optional discount code. If it exactly matches "STUDENT10" and the cost so far is over £10, apply a 10% discount using augmented assignment
- Print a short breakdown (base cost, whether express was applied, the insurance fee, whether a discount was applied) followed by the final total, formatted to two decimal places
- Organise the program's logic inside a `main()` function, called at the bottom of the file

Example run:

```
Enter parcel weight in kg: 3
Enter destination zone (local/national/international): national
Does it need express delivery? (yes/no): yes
Do you want insurance? (yes/no): yes
Enter declared value of parcel: 120
Enter a discount code (or press enter to skip): STUDENT10
Base cost: £4.50
Express surcharge: £5.00
Insurance fee: £3.50
Discount applied: -10%
Total cost: £11.70
```

Checkpoint

Running `parcel_cost.py` should correctly price a parcel using at least three destination-zone bands (with a sensible message for an unrecognised zone), apply an express surcharge, apply the correct insurance fee based on the declared value, apply a discount only when both the code matches and the cost exceeds the threshold, and print a clear breakdown with the total formatted to two decimal places, all organised inside `main()`.

Stretch goals (if time remains)

- Let the user enter a discount code that gives an extra fixed reduction

- Reject negative ages with a friendly message instead of computing a nonsense price
- Combine three conditions in one `if` using both `and` and `or`, and use parentheses to control how Python groups them

Self-check

Before you finish up, look back over what you've written — this isn't handed in, but it's worth confirming you've hit the key ideas:

- ☐ `code/labs/lab-02/` contains: `types.py`, `operators.py`, `booleans.py`, `decisions.py`, `ticket_price.py`
- ☐ Every script runs from the terminal without errors: `python <file>.py`
- ☐ At least one type conversion (`int()`, `float()`, `str()` or `bool()`) appears in your code
- ☐ At least one `if` / `elif` / `else` chain sorts a value into three or more categories
- ☐ `ticket_price.py`'s logic is organised inside `main()`, called at the bottom of the file
- ☐ (Optional) `parcel_cost.py` applies the correct zone rate, express surcharge and insurance fee, and only applies the discount when both conditions hold