Week 1 - Lesson 2: Functions and Comprehensions

Revision summary (v2) - Dec 27, 2025

1) Functions (the clean-code superpower)

A function is a named mini-program: it takes inputs (parameters), does work, and returns an output.

Key rule: **return** sends a value back to the caller. Code after return in the same function does not run.

Example (return a value, don't print inside):

```
def add(a, b):
    return a + b

x = add(2, 3)  # x is 5
```

2) Mutable vs Immutable (why lists behave differently)

Python uses **pass-by-assignment**: variables point to objects. If you modify a **mutable** object inside a function, the caller can see the change.

Immutable (cannot change in place): int, float, str, bool, tuple.

Mutable (can change in place): list, dict, set.

Immutable example (no change to the caller):

```
def bump(x):
    x += 1

a = 5
bump(a)
print(a)  # 5
```

Mutable example (caller changes):

```
def add_one(lst):
    lst.append(1)

x = [5]
add_one(x)
print(x)  # [5, 1]
```

3) List Comprehensions (short loop + filter)

Comprehensions are a compact way to build a new list.

If you want a quick rule:
[OUTPUT for ITEM in LIST if CONDITION]

Example: clean tasks (strip spaces, drop empties):

```
tasks = ["study", "", "gym", "  ", "sleep"]
clean = [t.strip() for t in tasks if t.strip() != ""]
print(clean)  # ['study', 'gym', 'sleep']
```

Tip: if a comprehension becomes hard to read, switch to a normal loop.

4) Tiny cheat-sheet (what tool to use)

Task	Best tool
Build a result list	[expr for x in items if condition]
Need index + value	for i, x in enumerate(items): ...
Search first match	loop + if + break (or next(..., None))
Keep asking user	while True: validate -> break

5) Break vs Continue (control flow inside loops)

break exits the loop. **continue** skips the rest of the current iteration and goes to the next one.

Example:

```
for i in [1, 2, 3]:
    if i == 2:
        continue
    print(i)
# prints 1 then 3
```

End-of-lesson checklist

• I can write small functions that return values.

• I know when a list changes in place (mutable) vs ints/strings (immutable).

• I can read and write a simple list comprehension.

• I can use enumerate(items) when I need index + value.

• I know the difference between break and continue.