AGGREGATORS

Aggregators

Functions that iterate through an iterable and return a single value that (usually) takes into account every element of the iterable

`min(iterable)` → minimum value in the iterable

`max(iterable)` → maximum value in the iterable

`sum(iterable)` → sum of all the values in the iterable

Associated Truth Values

You should already know this, but let's review briefly:

Every object in Python has an associated truth value `bool(obj)` → `True / False`

Every object has a `True` truth value, except:

- `None`
- `False`
- `0` in any numeric type (e.g. `0`, `0.0`, `0+0j`, ...)
- empty sequences (e.g. `list`, `tuple`, `string`, ...)
- empty mapping types (e.g. `dictionary`, `set`, ...)
- custom classes that implement a `__bool__` or `__len__` method that returns `False` or `0`

which have a `False` truth value

The `any` and `all` functions

`any(iterable)` → returns `True` if any (one or more) element in `iterable` is truthy

→ `False` otherwise

`all(iterable)` → returns `True` if all the elements in `iterable` are truthy

→ `False` otherwise

Leveraging the `any` and `all` functions

Often, we are not particularly interested in the direct truth value of the elements in our iterables

→ want to know if any, or all, satisfy some condition → if the condition is True

A function that takes a single argument and returns True or False is called a predicate

We can make `any` and `all` more useful by first applying a predicate to each element of the iterable

Example

Suppose we have some iterable `l = [1, 2, 3, 4, 100]`

and we want to know if: every element is less than `10`

First define a suitable predicate: `pred = lambda x: x < 10`

Apply this predicate to every element of the iterable:

```
results = [pred(1), pred(2), pred(3), pred(4), pred(100)]
```

→ `[True, True, True, True, False]`

Then we use `all` on these results `all(results)` → `False`

How do we apply that predicate?

The `map` function `map(fn, iterable)`

→ applies `fn` to every element of `iterable`

A comprehension: `(fn(item) for item in iterable)`

Or even:

```
new_list = []
for item in iterable:
    new_list.append(fn(item))
```

Coding Exercises