IN-PLACE CONCATENATION AND REPETITION

Concatenation +

Let's use Python's `list` as an example

We can concatenate two lists together by using the + operator

This will create a new list combining the elements of both lists

```
l1 = [1, 2, 3]              id(l1) = 0xFFF100

l2 = [4, 5, 6]              id(l2) = 0xFFF200

l1 = l1 + l2   → [1, 2, 3, 4, 5, 6]   id(l1) = 0xFFF300
```

In-Place Concatenation `+=`

Recall that for numbers I have said many times that

`a = a + 10` and `a += 10` meant the same thing?

That's true for numbers… but not in general!

it's true for numbers, strings, tuples → in general, true for immutable types

but not lists!

```
l1 = [1, 2, 3]          id(l1) = 0xFFF100

l2 = [4, 5, 6]          id(l2) = 0xFFF200

l1 += l2  → [1, 2, 3, 4, 5, 6]     id(l1) = 0xFFF100
```

the list was mutated

In-Place Concatenation `+=`

For immutable types, such as number, strings, tuples the behavior is different

`t += t1` has the same effect as `t = t + t1`

Since `t` is immutable, `+=` does NOT perform in-place concatenation

Instead it creates a new tuple that concatenates the two tuples and returns the new object

```
t1 = (1, 2, 3)          id(t1) = 0xFFF100

t2 = (4, 5, 6)          id(t2) = 0xFFF200

t1 += t2  → (1, 2, 3, 4, 5, 6)    id(t1) = 0xFFF300
```

In-Place Repetition *=

Similar result hold for the * and *= operator

```
l1 = [1, 2, 3]                  id(l1) = 0xFFF100

l1 = l1 * 2     → [1, 2, 3, 1, 2, 3]     id(l1) = 0xFFF200
```

But the in-place repetition operator works this way:

```
l1 = [1, 2, 3]                  id(l1) = 0xFFF100

l1 *= 2         → [1, 2, 3, 1, 2, 3]     id(l1) = 0xFFF100
```

the list was mutated

Code Exercises