Lists in Memory

Recap…

- A list is a **data structure**–something that lets you reason about multiple items.
- Syntax:
  - grocery_list: list[str] = [“eggs”, “milk”, “bread”]
- Can be an arbitrary length
- Empty List: list() or []
- Indexing like strings, but can *modify* by index
- Methods: append and pop

Lists in Memory: Comparing Lists and Strings

```
1   a: str = "24"
2   b: str = a
3   a += "6"
4   print(b)
```

```
1   a: list[int] = [2,4]
2   b: list[int] = a
3   a.append(6)
4   print(b)
```

Lists + Functions

Functions can:

- Take lists as arguments
- Return or create lists
- *Modify* lists!

Taking a List as an Argument

```
1  def display(vals: list[int]) -> None:
2      idx: int = 0
3      while idx < len(vals):
4          print(vals[idx])
5          idx += 1
6
7  display([1,2,3])
```

Creating + Returning a List

```
1  def odds_list(min: int, max: int) -> list[int]:
2      """returns list of odds between min and max"""
3      odds: list[int] = list()
4      x: int = min
5      while x <= max:
6          if x % 2 == 1:
7              odds.append(x)
8          x += 1
9      return odds
10
11 global_odds: list[int] = odds_list(2,10)
12 print(global_odds)
```

Modifying a List

```
1  def remove_first(xs: list[str]):
2      xs.pop(0)
3
4  course: list[str] = ["Comp", "110"]
5  remove_first(course)
```

Coding Example (if we have time)

- Let's implement a function where we can call with 2 arguments:
  - A “needle" int value we are searching for
  - A "haystack" list of values we are searching in

The return value of the function should be True if in the haystack at least once and False otherwise

The name of the function will be contains