Lists in Memory

Hack110!

When? Saturday, November 9th from 11 am - 11 pm

Where? In Sitterson Lower Lobby

Who can join? Anyone in COMP 110! No prior experience required. Bring a partner or come as yourself (we'll have team-building activities if you want a partner)

Come for a fun day of coding, workshops and events (also **food will be provided**):

- Choose between web development or game development track
- Go to various **workshops & events** such as: Navigating the CS Major, Resume workshop, ice cream station, and kahoot trivia
- Link: Sign up here! Or via the QR code →
- **Sign-up will close MONDAY, OCT 14 AT 11:59 PM**
    - Spots are limited! So make sure you sign-up early!

Sign-Up Form!

Recap…

- A list is a **data structure**–something that lets you organize and store data in a format such that they can be accessed and processed efficiently.
- Syntax:
  - `grocery_list: list[str] = ["eggs", "milk", "bread"]`
- Can be an arbitrary length
- Empty List: use a constructor, list() or the literal, []
- Indexing like strings, but we can *modify* by index
  - Ex: `grocery_list[1] = "almond milk"`
- Methods:
  - Append, to add to the end of the list
  - Pop, to remove an item from the list (by specifying its index)

Lists in Memory: Comparing Lists and Strings

* notes about memory diagrams will be in pink!

module 1:

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

Output

"24"

Stack

a | ~~"24"~~ "246"

b | "24"

Heap

module 2:

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

Output

[2, 4, 6]

Stack

a | id:0

b | id:0

Heap

id:0	list[int]
0	2
1	4
2	6

indices ↑ values ↑

* a and b refer to the same list in the heap! *

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])
```

positional argument!

Output

1
2
3

Stack

Globals

display	id:0

display

RA	7
RV	None
vals	id:1
idx	~~0~~ ~~1~~ ~~2~~ 3

Heap

id:0	fn 1-5

id:1	list[int]
0	1
1	2
2	3

indices

values

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)
```

append method adds the specified int to the end of the list!

positional arguments

2 10

Output

[3,5,7,9]

Stack

Globals

odds_list	id:0
global_odds	id:1

odds_list

RA	11
RV	id:1
min	2
max	10
odds	id:1

x | ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ ~~10~~ 11

Heap

id:0	fn 1-9

id:1	list[int]
0	3
1	5
2	7
3	9

indices ↑ values ↑

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)
```

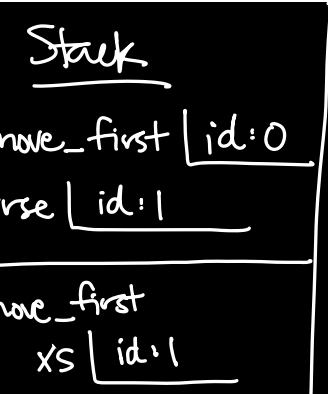
→ None:

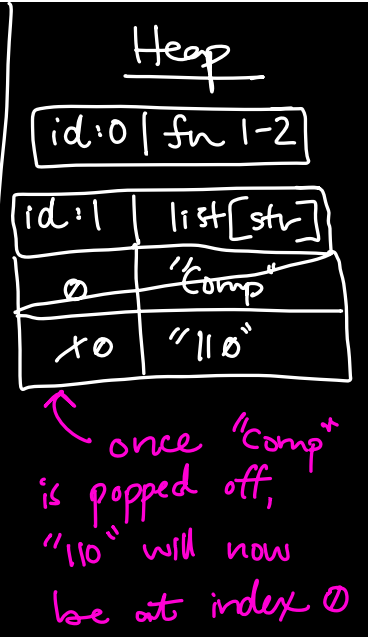
"remove the str at this index"

"pop off"

positional argument

Output





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

```python
def dup(xs: list[str]):
    start_len: int = len(xs)
    i: int = 0
    while i < start_len:
        xs.append(xs[i])
        i += 1

groceries: list[str] = ["apples", "eggs"]
print(dup(groceries))
print(groceries)
```

=> None!

add this string to the end of the list!

0 1

Output

None

["apples", "eggs", "apples", "eggs"]

Stack

Globals

dup	id:0
groceries	id:1

dup

xs	id:1
start_len	2
i	0 X 2

Heap

id:0	fn 1-6

id:1	list[str]
0	"apples"
1	"eggs"
2	"apples"
3	"eggs"