SHPE
Leading Hispanics in STEM

March 29-30, 2025

UNC- Chapel HIill -Sitterson Hall

SOLHACKS

UNC Chapel Hill's First Hackathon for Latinos in Tech

Creating a welcoming and inclusive environment for Latinos in tech, fostering representation, building networks, and empowering innovation in the community

- Students of all backgrounds
- Beginner friendly
- Sponsorship fair
- Cool tech prizes

REGISTER NOW!

@solhacksunc

Latinos in Tech

Hack110 Sign-Up Form!

When? Saturday, April 5th from 10 AM - 12 AM (Midnight)

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 (**food and CLE credit 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 and MORE!
- Link: Sign-Up Here! Or via the QR code
- **Sign-Up form EXTENDED TO Monday, March 31st at 11:59 pm**
  - Spots are limited! So we'll prioritize interest!
  - If you have a partner, **ONLY ONE OF YOU** has to sign up - you will just enter your partner's info in the form.

Sign-Up Here!

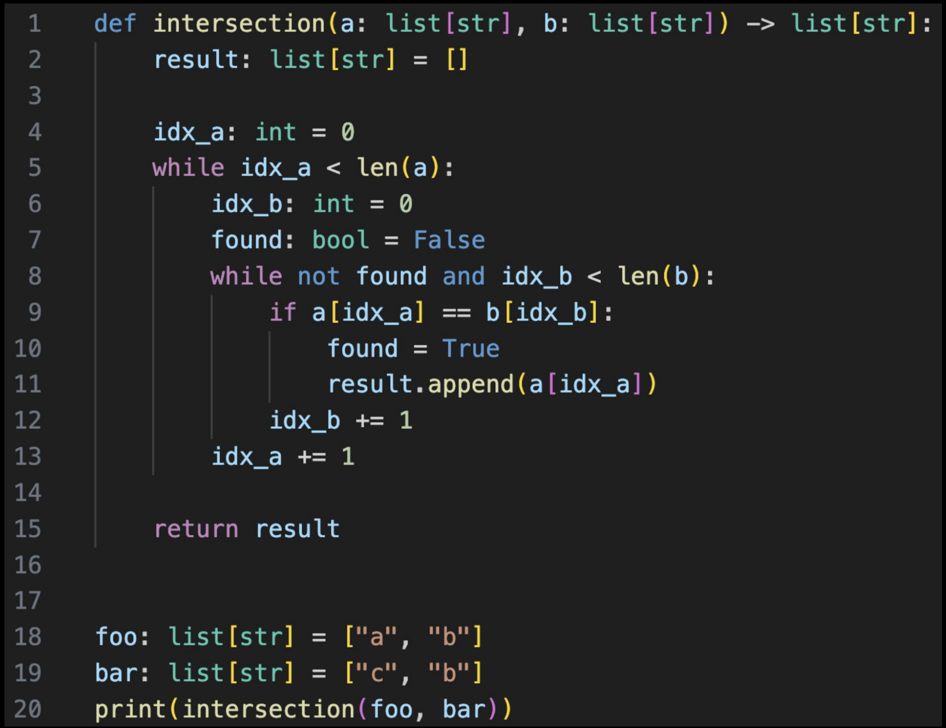
COMP110

CL22: Sets and Dictionaries

Announcements

- Quiz 02 grades will be released today – median ~85%!
- LS11 – Dictionaries due today
- EX03 released today, due *next Wednesday* (March 26)!
- Quiz 03 on Friday, March 28
    - Review Session on Wednesday (March 26) at 6:15pm in Fred Brooks (FB) 009

Warm-up diagram



```
1   def intersection(a: list[str], b: list[str]) -> list[str]:
2       result: list[str] = []
3   
4       idx_a: int = 0
5       while idx_a < len(a):
6           idx_b: int = 0
7           found: bool = False
8           while not found and idx_b < len(b):
9               if a[idx_a] == b[idx_b]:
10                  found = True
11                  result.append(a[idx_a])
12              idx_b += 1
13          idx_a += 1
14  
15      return result
16  
17  
18  foo: list[str] = ["a", "b"]
19  bar: list[str] = ["c", "b"]
20  print(intersection(foo, bar))
```

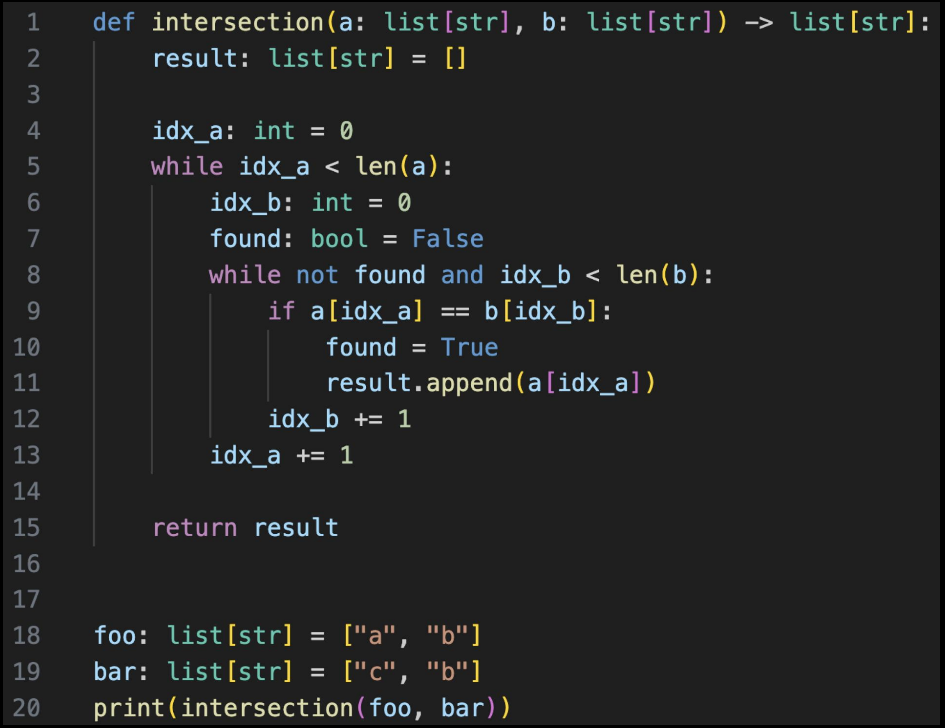
After diagramming:

Assume our unit of "operation" is the number of times the block of lines #9-12 are evaluated.

Q1. Can different values of a and b lead to a difference in the number of operations required for the intersection function evaluation to complete?

Q2. If so, provide example item values for a and b which require the fewest operations to complete? Then try for the maximal operations to complete?

Q3. Assuming the item values of a and b are random and unpredictable, about how many operations does this function take to complete?



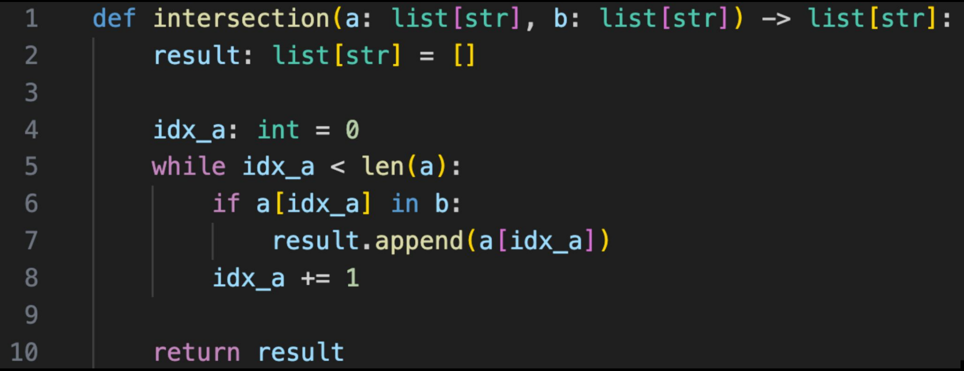
```python
def intersection(a: list[str], b: list[str]) -> list[str]:
    result: list[str] = []

    idx_a: int = 0
    while idx_a < len(a):
        idx_b: int = 0
        found: bool = False
        while not found and idx_b < len(b):
            if a[idx_a] == b[idx_b]:
                found = True
                result.append(a[idx_a])
            idx_b += 1
        idx_a += 1

    return result


foo: list[str] = ["a", "b"]
bar: list[str] = ["c", "b"]
print(intersection(foo, bar))
```

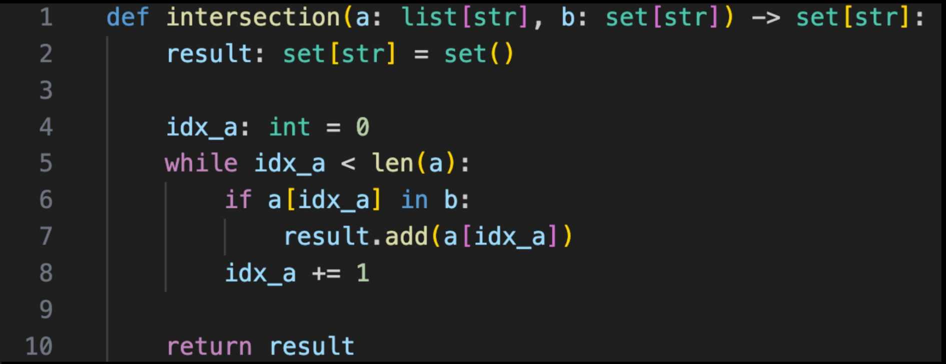
Comparing lists and sets



```python
def intersection(a: list[str], b: list[str]) -> list[str]:
    result: list[str] = []

    idx_a: int = 0
    while idx_a < len(a):
        if a[idx_a] in b:
            result.append(a[idx_a])
        idx_a += 1

    return result
```



```python
def intersection(a: list[str], b: set[str]) -> set[str]:
    result: set[str] = set()

    idx_a: int = 0
    while idx_a < len(a):
        if a[idx_a] in b:
            result.add(a[idx_a])
        idx_a += 1

    return result
```

Suppose a and b each had 1,000,000 elements, the worst case difference here is approximately 1,000,000 operations versus 1,000,000**2 or 1,000,000,000,000 operations.

If your device can perform 100,000,000 operations per second, then...

A call to a will complete in 2.78 hours and b will complete in 1/100th of a second.

Let’s explore Dictionary syntax in VSCode together…

In your cl directory, add a file named cl23_dictionaries.py with the following starter:

```
"""Examples of dictionary syntax with Ice Cream Shop order tallies."""

ice_cream: dict[str, int] = {
  "chocolate": 12,
  "vanilla": 8,
  "strawberry": 4,
}
```

Save, then open up this file in Trailhead's REPL and we will explore key syntax together.

Ready to go? Try evaluating the following expression:

```
ice_cream["vanilla"] += 110
```

Syntax

Data type:

```
name: dict[<key type>, <value type>]
temps: dict[str, float]
```

Construct an empty dict:

```
temps: dict[str, float] = dict() or
temps: dict[str, float] = {}
```

Construct a populated dict:

```
temps: dict[str, float] = {"Florida": 72.5, "Raleigh": 56.0}
```

Let's try it!

Create a dictionary called ice_cream that stores the following orders

Keys	**Values**
chocolate	12
vanilla	8
strawberry	5

Length of dictionary

```
len(<dict name>)

len(temps)
```

Let's try it!

Print out the length of ice_cream.

What exactly is this telling you?

Adding elements

We use subscription notation.

```
<dict name>[<key>] = <value>

temps["DC"] = 52.1
```

Let’s try it!

Add 3 orders of "mint" to your ice_cream dictionary.

Access + Modify

To access a value,
use subscription notation:

```
<dict name>[<key>]
temps["DC"]
```

To modify, also use subscription notation:

```
<dict name>[<key>] = new_value
temps["DC"] = 53.1  or  temps["DC"] += 1
```

Let's try it!

Print out how many orders there are of "chocolate".
Update the number of orders of Vanilla to 10.

Important Note: Can’t Have Multiple of Same ***Key***

Keys Values

Flavor	Num Orders
"chocolate"	12
"vanilla"	10
"strawberry"	5
"chocolate"	10

(Duplicate *values* are okay.)

Keys Values

Flavor	Num Orders
"chocolate"	12
"vanilla"	10
"strawberry"	5
"mint"	5

Check if key in dictionary

<key> in <dict name>

"DC" in temps

"Florida" in temps

<u>Let’s try it!</u>

Check if both the flavors "mint" and "chocolate" are in ice_cream.

Write a conditional that behaves the following way:
If "mint" is in ice_cream, print out how many orders of "mint" there are.
If it’s not, print "no orders of mint".

Removing elements

Similar to lists, we use pop()

```
<dict name>.pop(<key>)
```

```
temps.pop("Florida")
```

Let’s try it!

Remove the orders of "strawberry" from ice_cream.

"for" Loops

Let’s try it!

Use a for loop to print:
chocolate has 12 orders.
vanilla has 10 orders.
strawberry has 5 orders.

"for" loops iterate over the *keys* by default

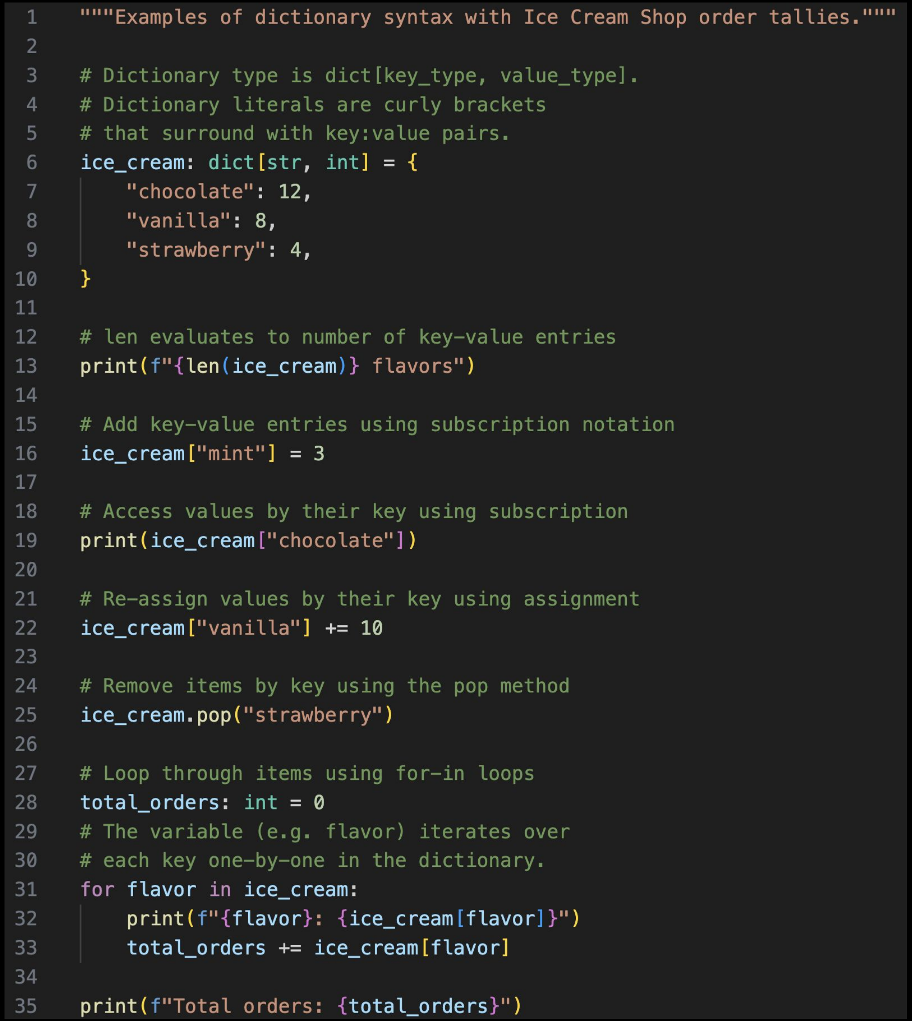
```
for key in ice_cream:
    print(key)
```

```
for key in ice_cream:
    print(ice_cream[key])
```

Flavor	Num Orders
"chocolate"	12
"vanilla"	10
"strawberry"	5

Final Notes

This is the code we worked through together in class, for reference.



```python
"""Examples of dictionary syntax with Ice Cream Shop order tallies."""

# Dictionary type is dict[key_type, value_type].
# Dictionary literals are curly brackets
# that surround with key:value pairs.
ice_cream: dict[str, int] = {
    "chocolate": 12,
    "vanilla": 8,
    "strawberry": 4,
}

# len evaluates to number of key-value entries
print(f"{len(ice_cream)} flavors")

# Add key-value entries using subscription notation
ice_cream["mint"] = 3

# Access values by their key using subscription
print(ice_cream["chocolate"])

# Re-assign values by their key using assignment
ice_cream["vanilla"] += 10

# Remove items by key using the pop method
ice_cream.pop("strawberry")

# Loop through items using for-in loops
total_orders: int = 0
# The variable (e.g. flavor) iterates over
# each key one-by-one in the dictionary.
for flavor in ice_cream:
    print(f"{flavor}: {ice_cream[flavor]}")
    total_orders += ice_cream[flavor]

print(f"Total orders: {total_orders}")
```