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!

CL22: Sets and Dictionaries

Announcements

- 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

Limits of Lists for collections of data (1/2)

Using a list, we *could* store everyone in COMP110's PID associated with ONYEN

list[str]	
Index	Value
0	""
1	""
… 710,453,081 items elided …	
710453084	"krisj"
… 9,857,700 items elided …	
720310785	"abyrnes1"
… 9,809,924 items elided …	
730120710	"ihinks"

Warm-up question:
Why does using a **`list[str]`** feel wrong/inefficient?

Limits of Lists for collections of data (2/2)

`onyens:`

list[str]	
Index	Value
0	"ihinks"
1	"abyrnes1"
2	"sjiang3"
... 296 items elided ...	
299	"krisj"

`seats:`

list[str]	
Index	Value
0	"A1"
1	"A2"
2	"A3"
... 296 items elided ...	
299	"N17"

Suppose we model quiz seat assignments with lists. One list has seats, the other has the assigned ONYEN at the same index.

Given the onyen "sjiang3", how do you algorithmically find their assigned seat?

We could use the `in` operator (a new concept)…

```
4   pids_of_interest: list[str] = ["sjiang3", "ihinks"]
5   idx: int = 0
6   while idx < len(pids_of_interest):
7       if pids_of_interest[idx] in pids:
8           print("We found a PID in the list!")
9       idx += 1
10
```

… but try to avoid using it on lists!

Enter: sets!

Sets, like lists, are a *data structure* for storing collections of values.

Unlike lists, sets are *unordered* and each value has to be *unique*.

Lists: *always* zero-based, sequential, integer indices!

Benefit of sets: testing for the existence of an item takes only one “operation,” regardless of the set’s size.

```
pids: set[str] = {730120710, 730234567, 730000000}
```

Great! … But what if we want to connect people’s PIDs with their ONYENs?

Enter: Dictionaries!

Dictionaries, like lists, are a *data structure* for storing collections of values.

Unlike lists, dictionaries give *you* the ability to decide what to *index* your data by.

- Lists: *always* zero-based, sequential, integer indices!

Dictionaries are indexed by ***keys*** associated with ***values***. *This is a unique, one-way mapping!*

- Analogous: A real-world dictionary's ***keys*** are *words* and associated ***values*** are *definitions.*

`pid_to_onyen:`

`dict[int, str]`	
`key`	`value`
`730120710`	`"ihinks"`
`710453084`	`"krisj"`
`720310785`	`"abyrnes1"`

`onyen_to_seat:`

`dict[str, str]`	
`key`	`value`
`"ihinks"`	`"A1"`
`"abyrnes1"`	`"A2"`
`"sjiang3"`	`"A3"`
`"krisj"`	`"N17"`

Let’s diagram key concepts

```python
# USD exchange rate to other currencies
exchange: dict[str, float] = {
    "CNY": 7.10,  # Chinese Yuan
    "GBP": 0.77,  # British Pound
    "DKK": 6.86,  # Danish Kroner
}

dollars: float = 100.0

# Access dictionary value by its key
pounds: float = dollars * exchange["GBP"]

# Append a key-value entry to dictionary
exchange["EUR"] = 0.92

# Update a key-value entry in dictionary
exchange["CNY"] -= 1.00

# len is the number of key-value entries
count: int = len(exchange)
```

Let’s explore Dictionary syntax in VSCode together…

In your cl directory, add a file named cl22_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"]
```

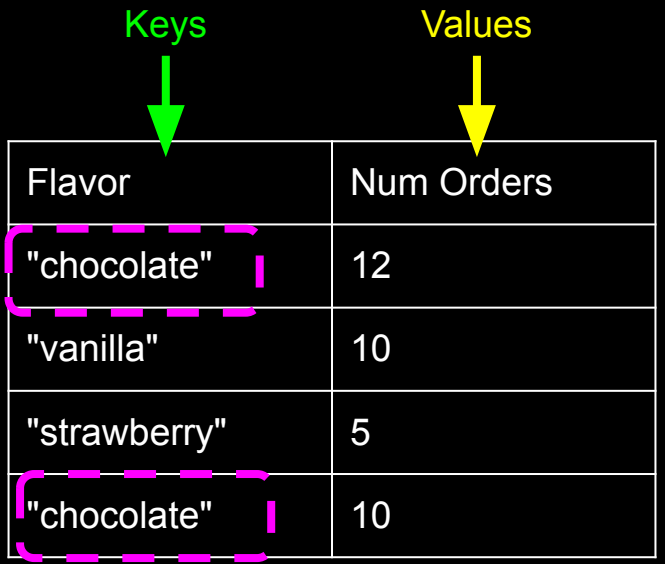
To modify, also use subscription notation:

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

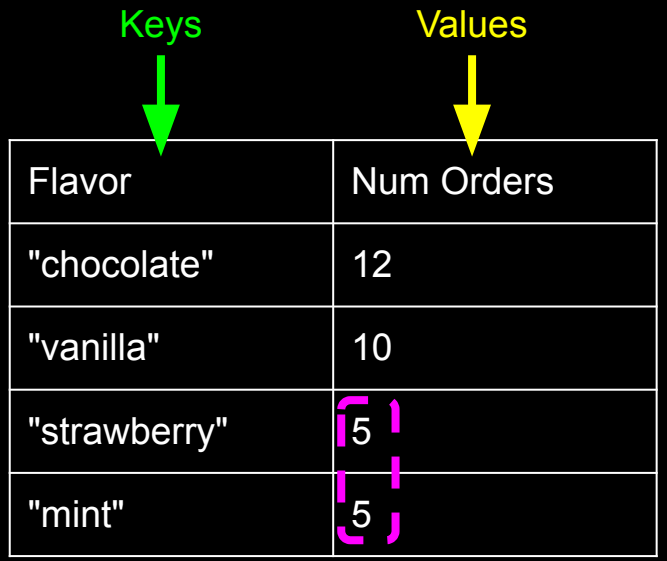
Let's try it!

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***



(Duplicate *values* are okay.)



Check if key in dictionary

<key> in <dict name>

"DC" in temps

"Florida" in temps

Let's try it!

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

"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

Let's try it!

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

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}")
```