Objectives

- Dictionaries

Lab Preparation Suggestions

- Review frequently
  - Learning a new language
  - Better to have some practice every day (rather than every week)
- Review example programs
  - Do you [still] understand them after class?
- Active work in interactive text book
  - Don’t just click the boxes
- Focus is on the current week, but we are using tools we learned in the last ~8 weeks.

LOOKUP ALTERNATIVES

List/String Lookup

- How do we “lookup” a value in a list or a character in a string?
- Answer:
  - By its index/position
- Requires:
  - Knowing the index where a value is located

Alternative Lookup

- Alternative: look up something by its *key*
  - Example: When I lookup my friend’s phone number in my contacts, I don’t know that the number is at position X in my contacts. I look up my friend’s number by her *name*.
  - Need a fast way to figure out “given this *key*, what is the *value* associated with it?”
- This type of data structure is known as a ***dictionary*** in Python
  - Maps a **key** to a **value**
  - Contacts’ key: name; value: phone number

Examples of Dictionaries

Dictionary	Keys	Values
Dictionary		
Textbook's index		
Cookbook		
URL (Uniform Resource Locator)		

- Any other things we've done/used in class?

Examples of Dictionaries

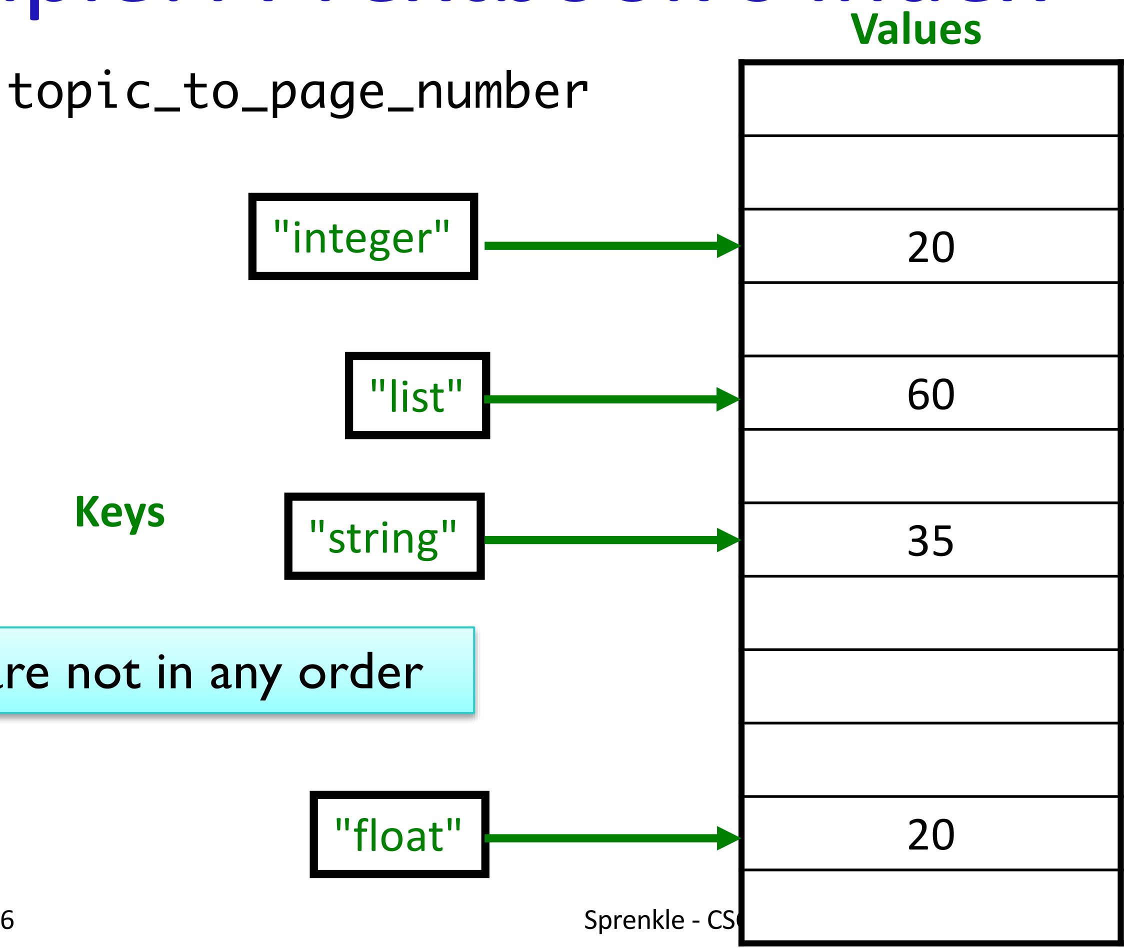
Dictionary	**Keys**	**Values**
Dictionary	Word	Definition
Textbook's index	Keyword	Page number
Cookbook	Food type	Recipes
URL (Uniform Resource Locator)	URL	Web page

- Any other things we've done/used in class?

Examples of Dictionaries

- Real-world:
  - Dictionary
  - Textbook’s index
  - Cookbook
  - URL (Uniform Resource Locator)
- Examples from class
  - Variable name → value
  - Function name → function definition
  - ASCII value → character

Example: A Textbook’s Index



Lots of empty space to add new entries

Keys are not in any order

Keys are unique; values are not necessarily unique

Dictionaries in Python

- Map **keys** to **values**
  - Keys are probably **not** alphabetized
  - Mappings are from ***one*** key to **one** value
    - Keys are ***unique***, Values are not necessarily unique
      - Example: student id → last name
    - Keys must be **immutable** (numbers, strings)
- Similar to Hashtables/Hashmaps in other languages

How would we handle if there is *more than one value* for a given key?

Creating Dictionaries in Python

Syntax:

```
{<key>:<value>, …,
 <key>:<value>}
```

```
empty = {}
char_to_ascii = { 'a':97, 'b':98, …, 'z':122 }
```

Dictionary Operations

Indexing	`<dict>[<key>]`
Length (# of keys)	`len(<dict>)`
Iteration	`for <key> in <dict>:`
Membership	`<key> in <dict>`
Deletion	`del <dict>[<key>]`

Unlike strings and lists, doesn't make sense to do slicing, concatenation, repetition for dictionaries

Accessing Values Using Indexing

- Syntax:

  `<dictionary>[<key>]`

- Examples:

```
char_to_ascii['z']

name_to_phone_num['friendname']
```

- `KeyError` if key is not in dictionary
  - Runtime error; exits program

Dictionary Methods

Method Name	Functionality
`<dict>.clear()`	Remove all items from dictionary
`<dict>.keys()`	Returns a copy of dictionary's keys (a set-like object)
`<dict>.values()`	Returns a copy of dictionary's values (a set-like object)
`<dict>.get(x [, default])`	Returns `<dict>[x]` if `x` is a key; Otherwise, returns None (or default value)

Accessing Values Using **get** Method

- Syntax: `<dict>.get(x [,default])`
  - Semantics: Returns `<dict>[x]` if x is a key
    Otherwise, returns None (or default value)
- Examples:

```
charToAscii.get('z')

nameToPhoneNum.get('friendname')
```

- If no mapping, None is returned instead of `KeyError`

Accessing Values: Look Before You Leap

- Typically, you will check if dictionary has a key before trying to access the key

```
if 'friend' in nameToPhoneNum :
     number = nameToPhoneNum['friend']
```

Know mapping exists before trying to access

- Or handle if get returns default

```
number = nameToPhoneNum.get('friend')
if number is None:
     # do something …
```

No phone number exists

Recall: Special Value **None**

- Special value we can use
  - E.g., Return value from function when there is an error
- If you execute

```
list = list.sort()
print(list)
```

  - Prints None because `list.sort()` does **not *return*** anything

Example Using **None** as an Error

```
def encryptLetter( letter, key ):
    """
    Pre: letter is a single lowercase letter, …
    returns the lowercase letter encoded by the key.
    If letter is not a lowercase letter, returns None
    """
    if letter < 'a' or letter > 'z’: # various ways to implement
         return None
     #As usual …
```

```
# example use
encLetter = encryptLetter(char, key)
if encLetter is None:
    print("Can't encrypt character", char, "in message: ")
```

Inserting Key-Value Pairs

- Syntax:

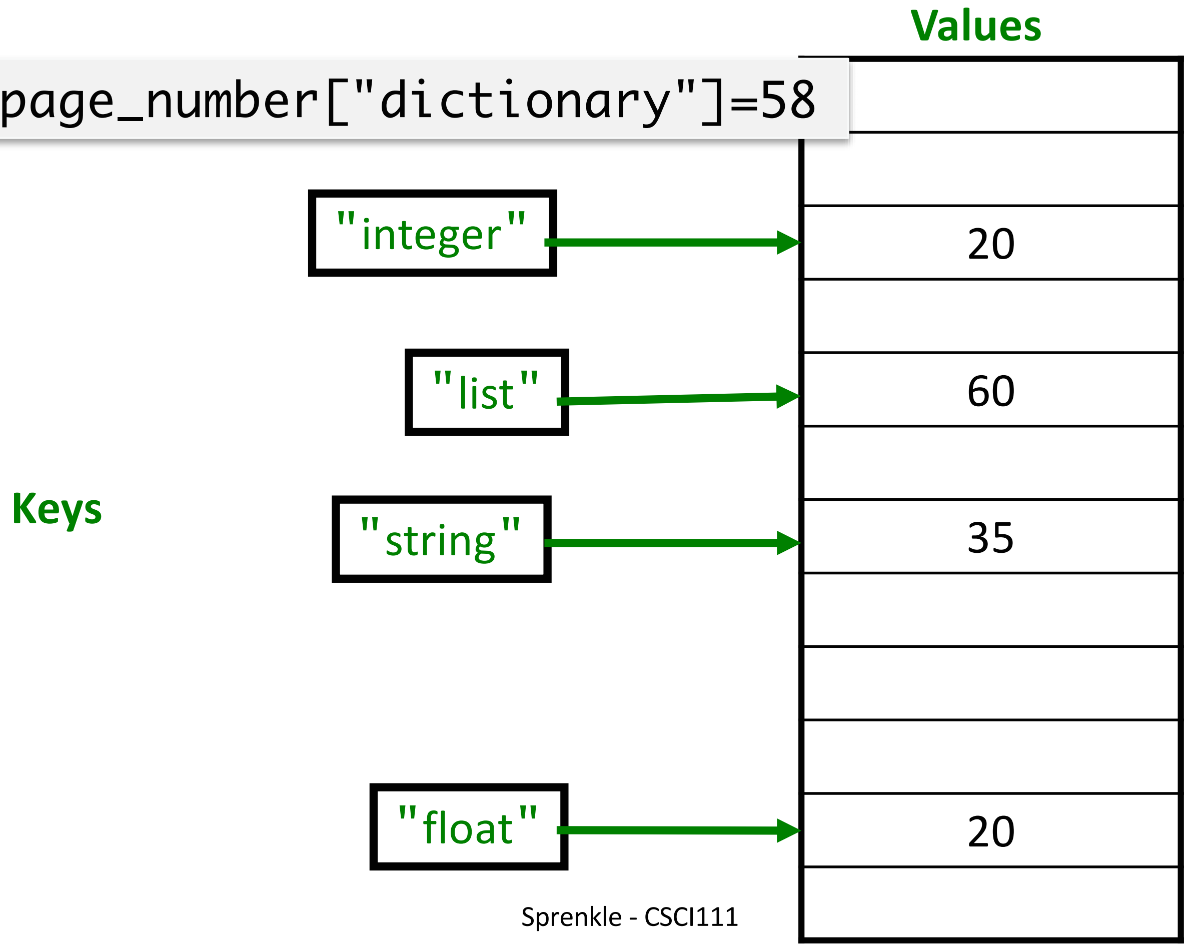
  `<dictionary>[<key>] = <value>`

- `char_to_ascii['a'] = 97`
  - Creates new mapping of 'a' → 97

`ascii_dictionary.py`

Textbook’s Index: Before Insertion

```
topic_to_page_number["dictionary"]=58
```



Values

Keys

"integer" 20

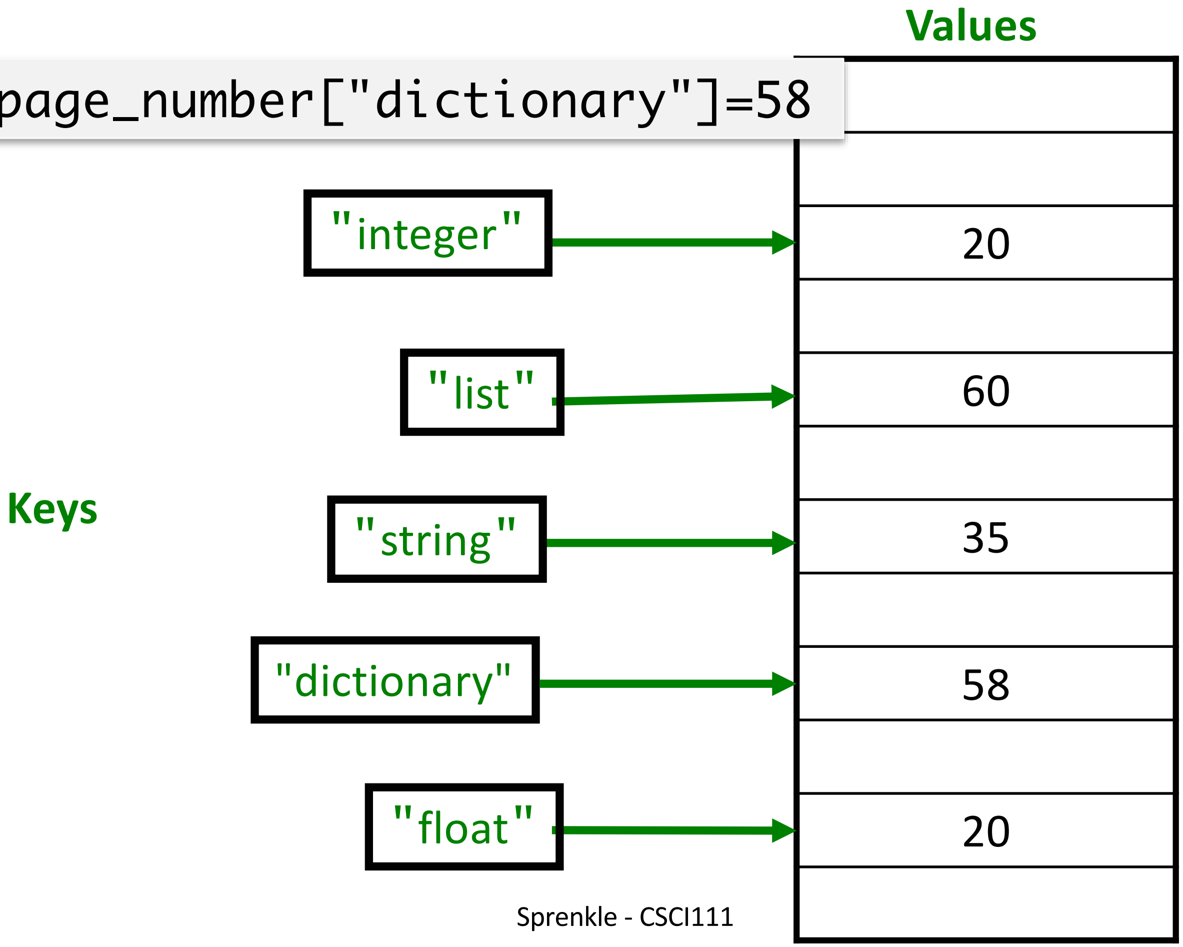
"list" 60

"string" 35

"float" 20

Textbook's Index: After Insertion

```
topic_to_page_number["dictionary"]=58
```



Values

Keys

"integer" → 20

"list" → 60

"string" → 35

"dictionary" → 58

"float" → 20

Adding/Modifying Key-Value Pairs

- Syntax:

```
<dictionary>[<key>] = <value>
```

- Example:

```
    name_to_phone_num['registrar'] = 8455
```

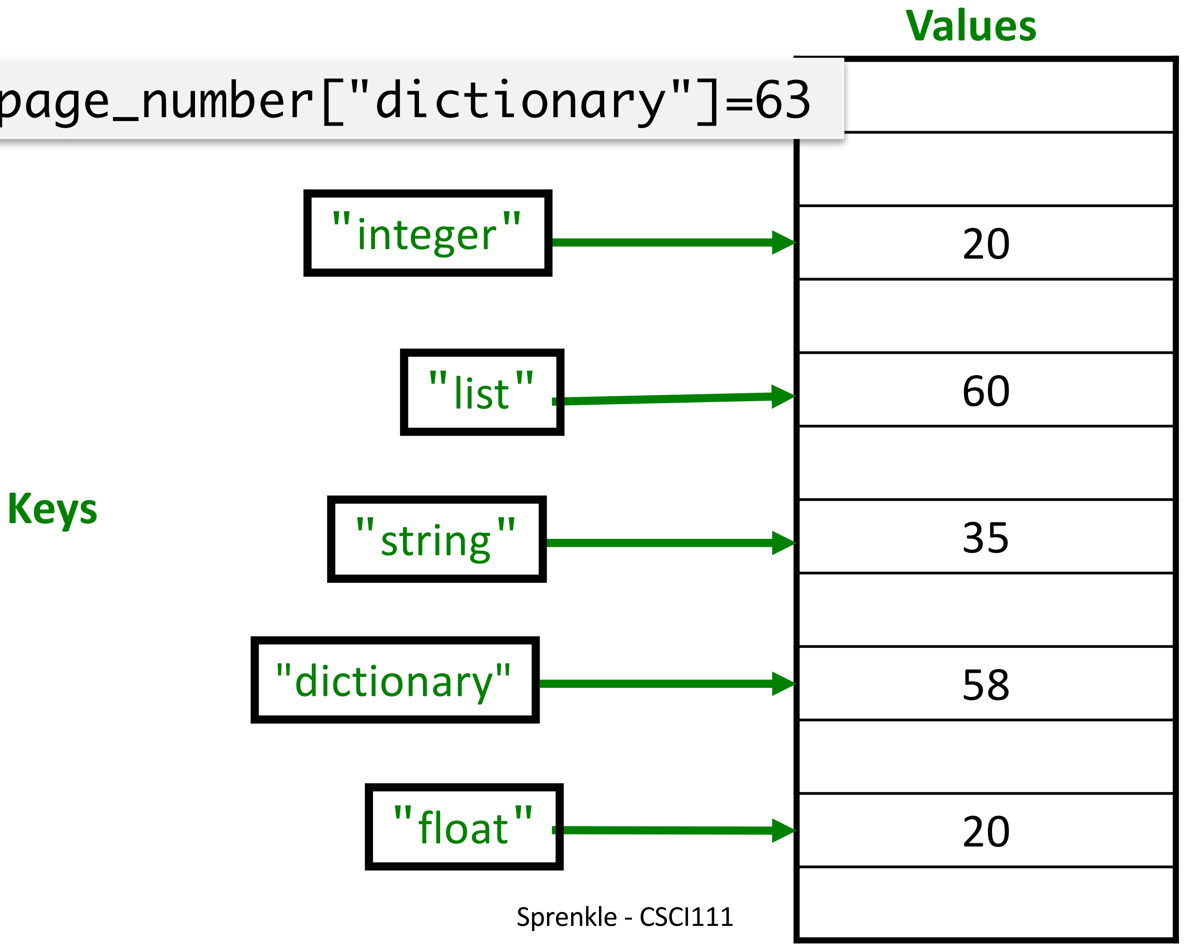
  - ➢ Adds mapping for `'registrar'` to 8455

  OR

  - ➢ If mapping already existed, *modifies* old mapping to 8455

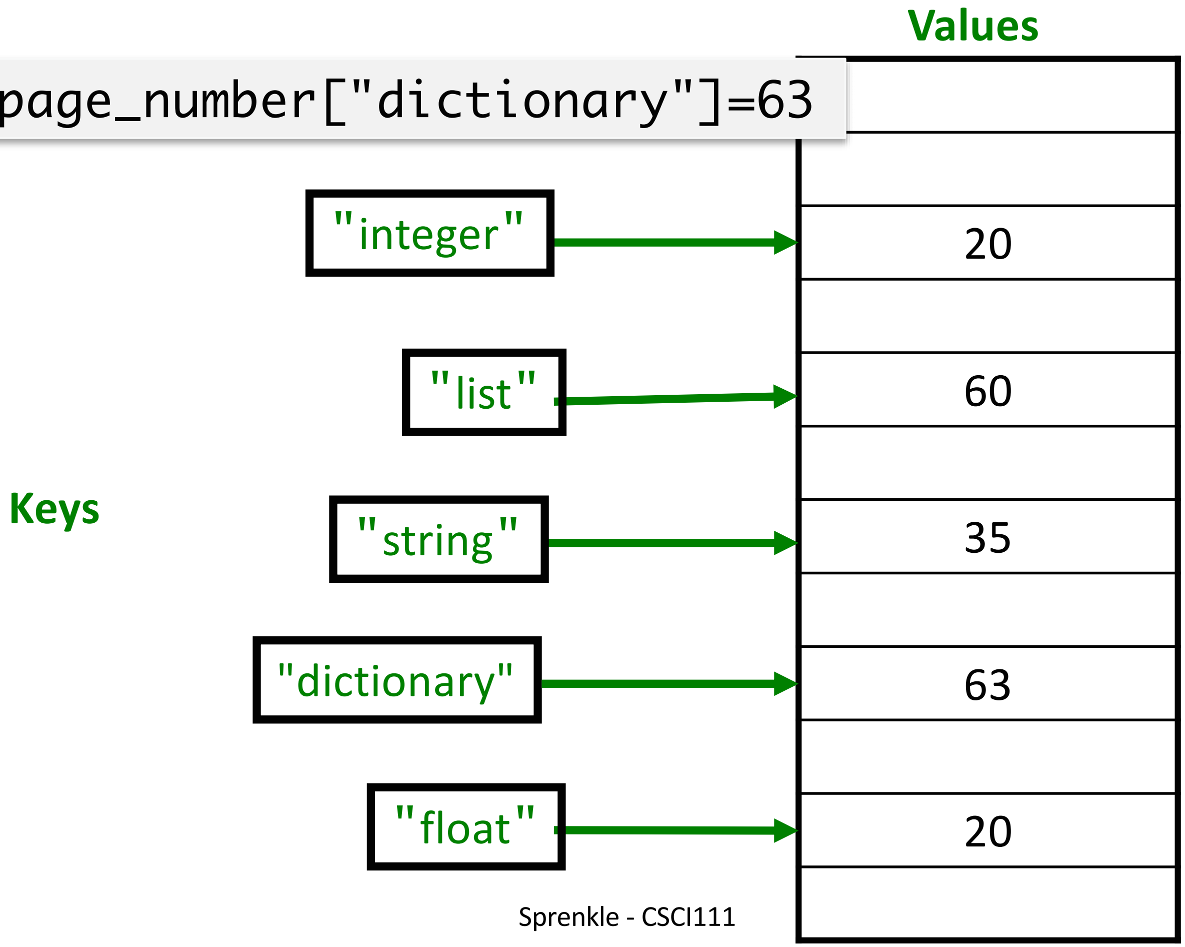
Textbook’s Index: Before Modification

```
topic_to_page_number["dictionary"]=63
```



Textbook's Index: After Modification

```
topic_to_page_number["dictionary"]=63
```



Values

Keys

"integer" → 20

"list" → 60

"string" → 35

"dictionary" → 63

"float" → 20

Methods `keys()` and `values()`

- Don’t return a `list` object
- But can be used similarly to a list
- If you want to make them into a list, use list converter:

```
keys = list(mydict.keys())
```

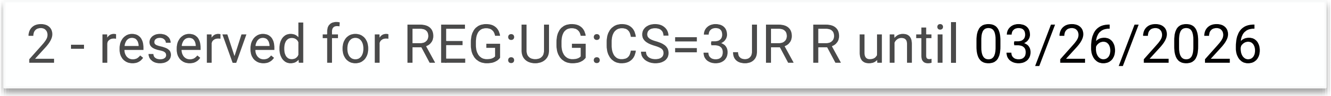
Using Dictionaries

`using_dictionary.py`

- Demonstrates lots of operations, methods, etc. in using dictionaries

Course Registration

- What does this mean under eligibility for a course listing in Workday?



2 - reserved for REG:UG:CS=3JR R until 03/26/2026

Representing Information

- Tag → User-friendly display
  - Ex: REG:UG:CS=3JR R → Junior or 3rd year
- Significance of date
  - 03/26/2026 → initial registration

Problem

- Given a file (`data/roster.dat`) of the form

  <firstname> <gradyear>
- Goal: quickly find the classyear of a particular student
  - Specifically, want to
    - Repeatedly prompt user for a first name of a student (given)
    - Display that student's graduation year

```
Whose class year? Bobby
Bobby is in the class of 2026
```

Example file:

```
Name1 2027
Name2 2028
Name3 2026
Name4 2028
Name5 2026
…
```

- Consider
  - How would we solve this before learning dictionaries?
  - How would we represent this information with dictionaries?
    - What is the key? What is the value?
  - If that dictionary existed, how would we implement the user input part?
  - How do we parse the file to create the dictionary?

`years_dictionary.py`

Solutions: Before Dictionaries

- Lots of possibilities
- One possibility:
  - Read through the file, looking for name; stop when found
- Another possibility:
  - Create two lists: one for first names, one for class years
  - Read the file, split each line of the file, add the first name and class year to the appropriate lists
  - Find the first name in the list → index of element in list
  - Use that index to find the class year in the other list

Analyzing Before Dictionaries Solutions

- Not ideal because…
  - Reading file multiple times
  - Keeping track of two lists
    - If remove/add people, need to add/remove from both lists to keep in sync
  - `find` is a relatively expensive operation
    - Has to look through each element: “Are you my element?” until find the match

Towards a Solution

- Representing information in a dictionary
  - Key: Name
  - Value: Class year
- User interaction (given that dictionary)
  - Check if the name is in the dictionary.
  - If so, index their name in the dictionary to get the class year
  - If not, report an error

Algorithm: Parse Data File

```
Name1 2027
Name2 2028
Name3 2026
Name4 2028
Name5 2026
…
```

- Create an empty dictionary
- Read in the file line by line
  - Split the line
  - From the split, get the last name and the year
  - Add a mapping of the last name to the year in the dictionary
    - (*accumulate* the data/mappings in the dictionary)
- for testing only: Display dictionary, in sorted order
- Return dictionary

Looking Ahead

- Lab 8: Due Friday
- No broader issue due this week