Python Data Structures

Dr. Emily Hill

The 7 Programming Basics

	Concept	Example from Math
1.	Variables	$x = 5$ *hellothere* = "howdy"
2.	Math & Logic	$5 * 7 + a - 3 / b \% 4$ a is 5 AND $x < 7$ OR *degree* ≥ 98
3.	Input/Output (IO)	print "Hello World"

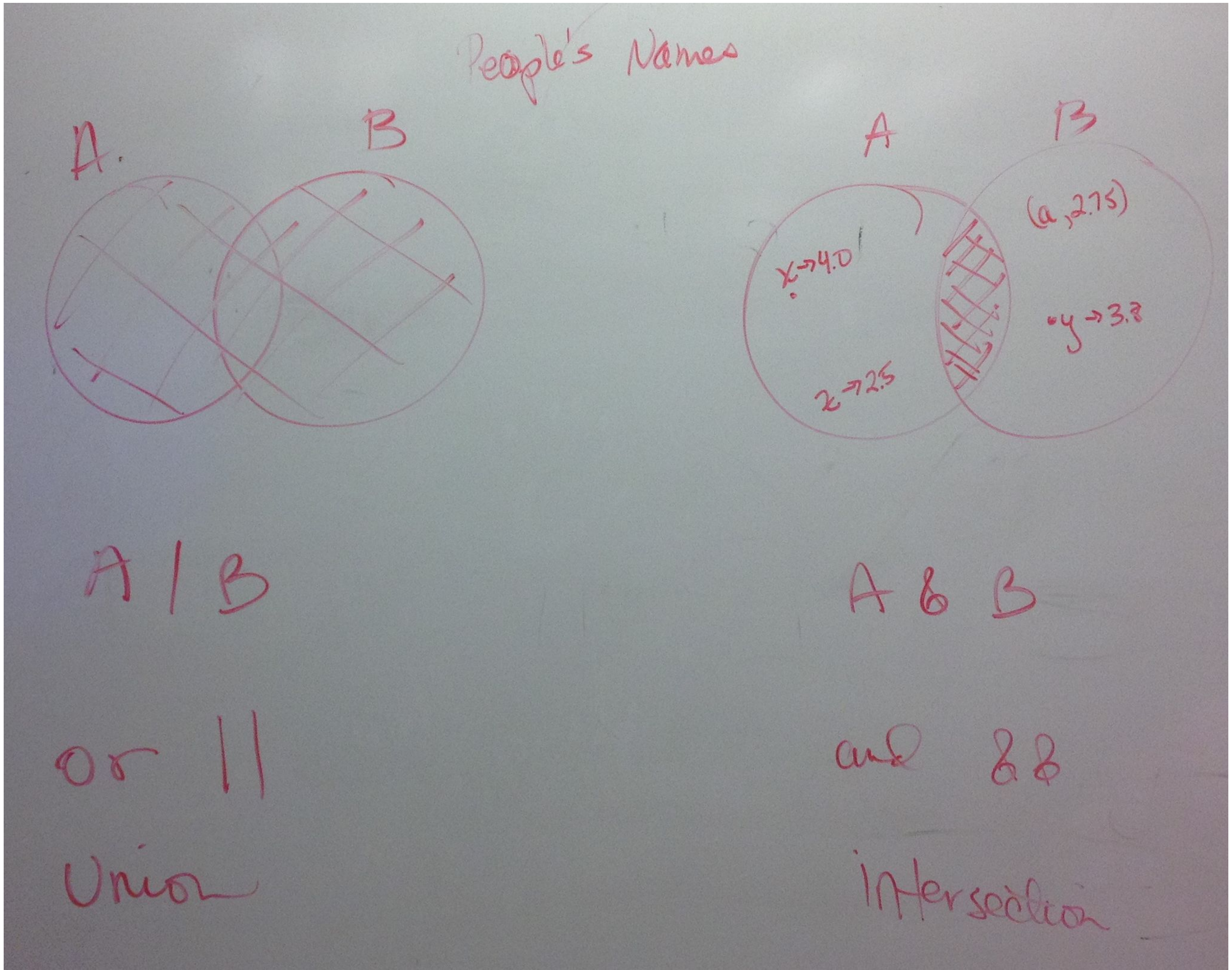
4.	Conditionals	**if** ($x == f(x)$)
		then print "x is 0 or 1"
		else print "x is not 0 or 1"

5.	Loops	**foreach** x **in** (*array*) print x
6.	Functions	$f(x) = x^2$
7.	Lists	*array* = 1:5 *array* = 1, 4, 7, 8, a, b, c, d

Lists & Tuples & Sets

- Tuples are an immutable list, use () instead of []
  - Like lists & strings support * in & slicing [:]
  - Built-in functions: min, max, len
- Sets can only hold unique values (use {})
  - Support len & in
  - Add, remove, discard, pop, clear
  - Union (|), intersection (&), difference (-)
  - Subset (<, <=) & superset (>, >=)
- Can convert between with built-in list, tuple, and set functions

Sets


People's Names
A
B
A
B
(a, 2.75)
x → 4.0
z → 2.5
y → 3.8
A | B
A & B
or ||
and &&
Union
intersection

Dictionaries

Dr. Emily Hill

How Does **in** Work for Lists?

- Example: guess **in** prevGuesses, where prevGuesses is a list object
  - For each element in list, checks if element equals (==) guess
- In the worst case, how many elements does **in** have to check?

Faster Lookups

- In my phone’s contacts app, if I wanted to know my friend’s phone number, …
  - Would I search through an ordered list of phone numbers?
  - No, I would look up my friend and find the phone number **associated** with my friend
- This type of data structure is known as a **dictionary** in Python
  - Maps a **key** to a **value**
  - Contacts’ key: “Friend’s name”, value: phone number

Examples of Dictionaries

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

Examples of Dictionaries

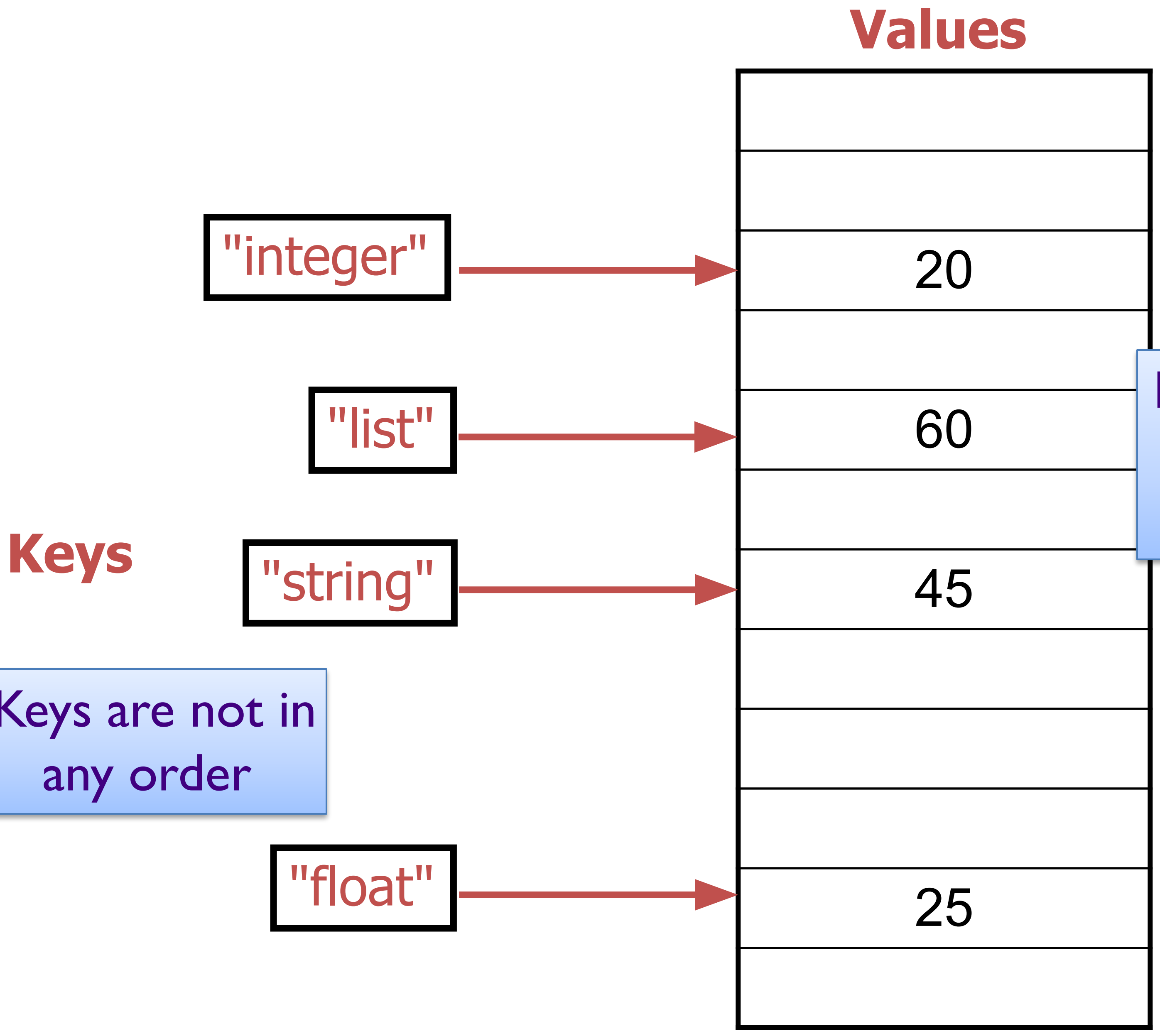
Dictionary	**Keys**	**Values**
Dictionary	Word	Definition
Textbook's index	Keyword	Page number
Cookbook	Food type	Recipes
URL (Uniform Resource Locator)	URL	Web page

- 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

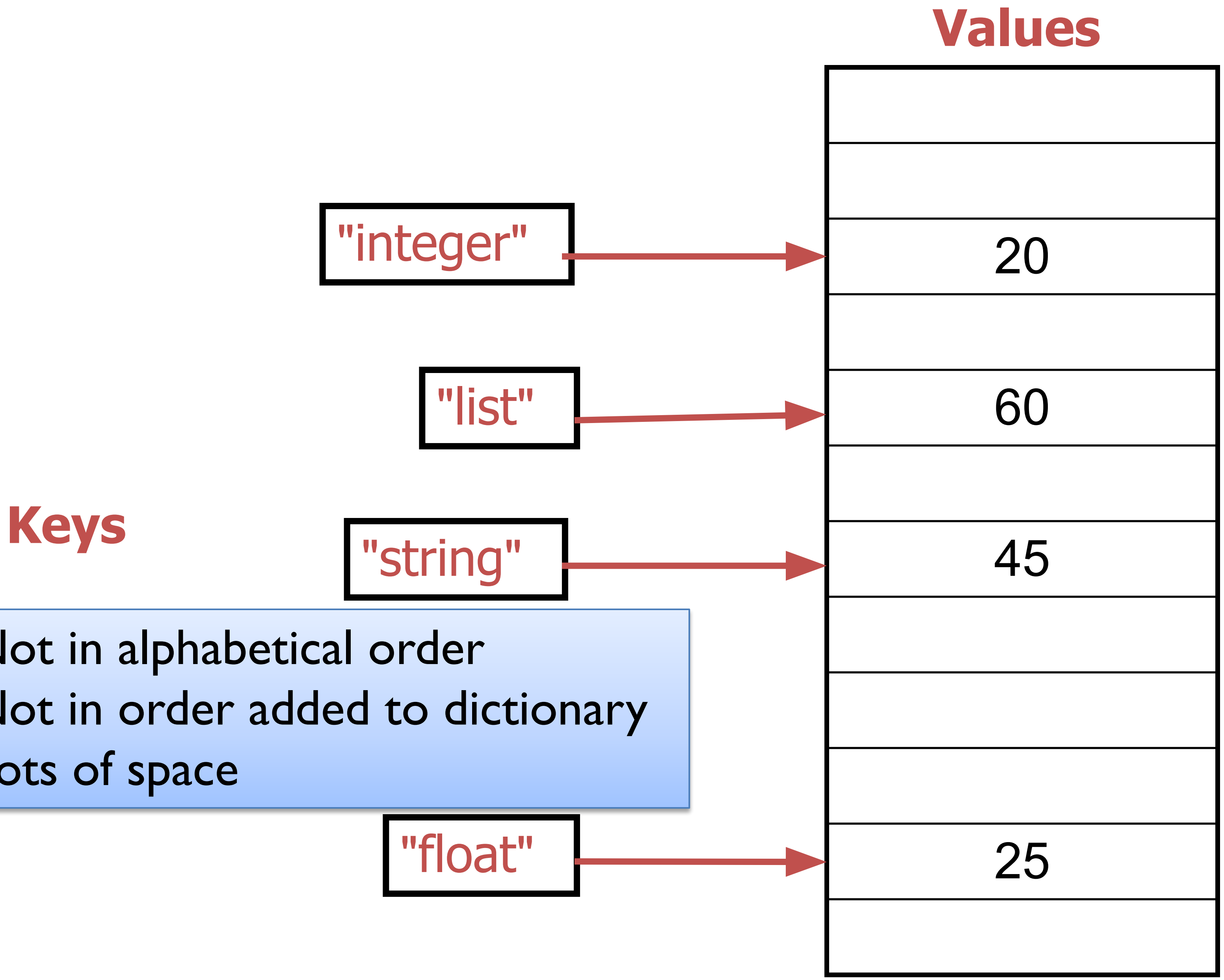
Example: Textbook's Index



Lots of empty space to add new values

Keys are not in any order

Textbook's Index



- Not in alphabetical order
- Not in order added to dictionary
- Lots of space

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?

Why Dictionaries?

- Another way to store data
- Allow fast lookup of data
  - Requires unique keys
    - Data may not have a natural mapping

Pros	Cons
Fast lookup (*much* faster than lists if a lot of elements)	Requires a lot of space, unique keys

Creating Dictionaries in Python

Syntax:

{<key>:<value>, …, <key>:<value>}

```
empty = {}
ascii = { 'a':97, 'b':98, 'c':99, …, '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

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 Keys

- Syntax:

  <dictionary>[<key>]

- Examples:

```
ascii['z']

contacts['friendname']
```

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

Accessing Values Using **get** Method

- <dict>.get(*x* [,*default*])
  - Returns <dict>[*x*] if *x* is a key; Otherwise, returns None (or default value)

```
ascii.get('z')

directory.get('friendname')
```

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

Accessing Values

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

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

Know mapping exists before trying to access

- Or handle if returns default

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

No phone number exists

Review: Special Value **None**

- Special value we can use
  - E.g., Return value from function when there is an error
- Similar to **null** in Java
- If you execute:

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

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

Examples using **None**

```
# returns the lowercase letter translated by the key.
# If letter is not a lowercase letter, returns None
def translateLetter( letter, key ):
    if letter < 'a' or letter > 'z':
        return None
    #As usual …
```

```
# example use
encLetter = translateLetter(char, key)
if encLetter is None:
    print("Error in message: ", char)
```

Inserting Key-Value Pairs

- Syntax:

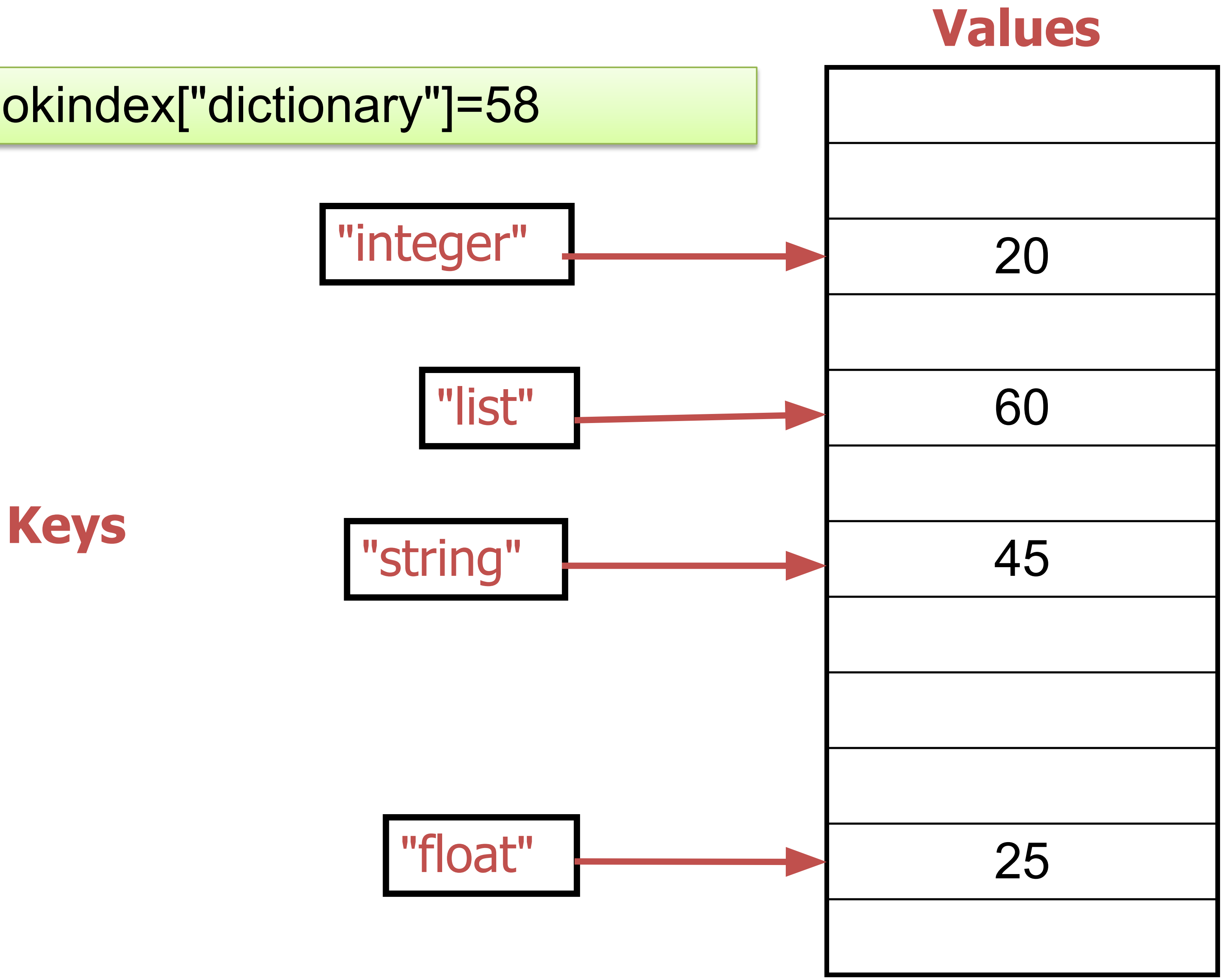
  <dictionary>[<key>] = <value>

- ascii['a'] = 97
  - Creates new mapping of 'a' □ 97

ascii_dictionary.py

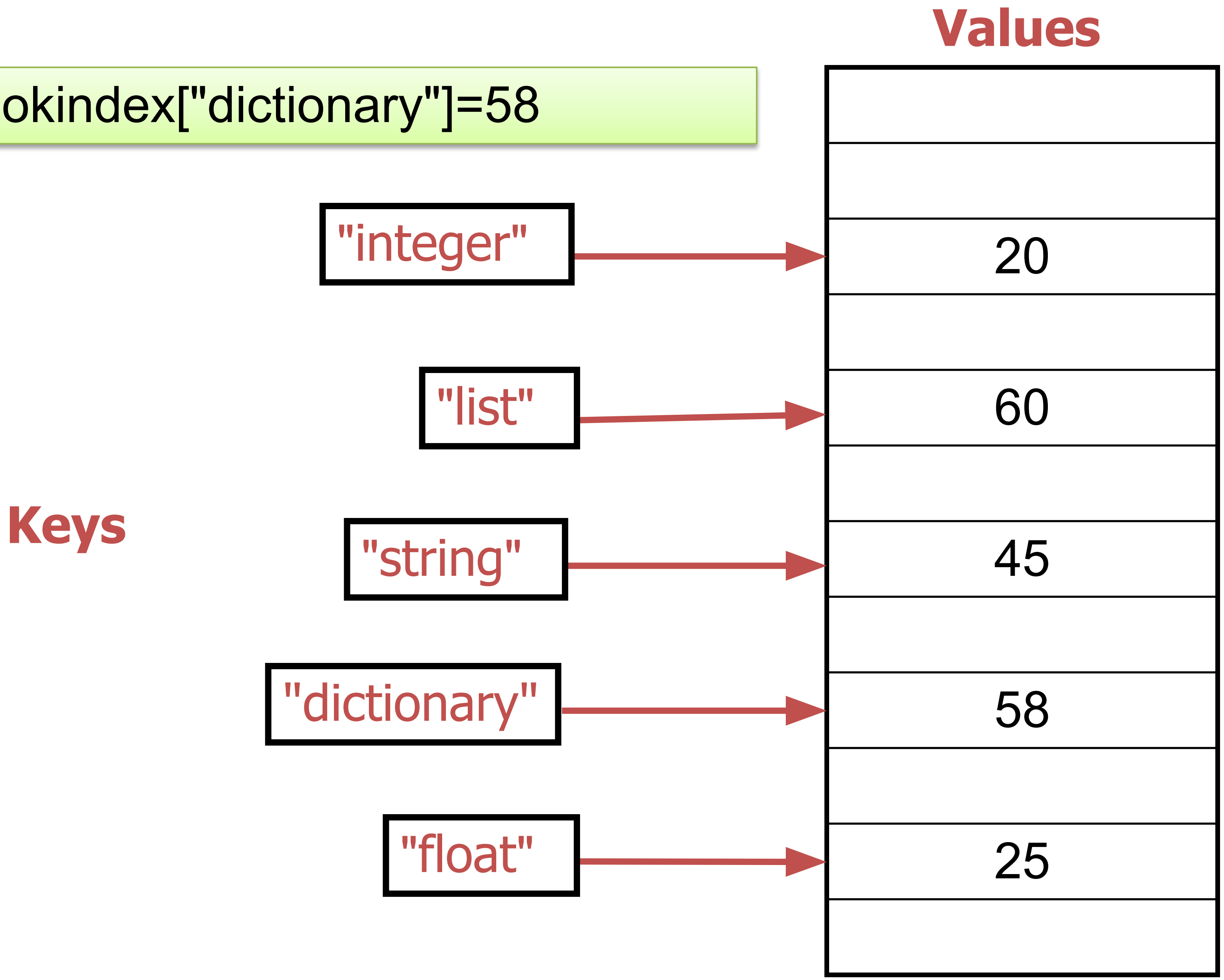
Textbook's Index

```
bookindex["dictionary"]=58
```



Textbook's Index

```
bookindex["dictionary"]=58
```



Adding/Modifying Key-Value Pairs

- Syntax:

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

- directory[‘registrar’] = 3025
  - Adds mapping for ‘registrar’ to 3025

  OR
  - Modifies old entry if it existed to 3025

Using Dictionaries

using_dictionary.py

Methods keys() and values()

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

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

Discussion

- Compare lists and dictionaries
  - What are their properties?
  - How are they similar?
  - How are they different?
  - When do you use one or the other?

Lists vs. Dictionaries

Lists	Dictionaries
integer *positions* (0, …) to any type of value	Map immutable *keys* (int, float, string) to any type of value
Ordered	Unordered
Slower to find a value (**in**)	Fast to find a value (use key)
Fast to print in order	Slower to print in order (by key)
Only as big as you make it	Takes up a lot of space (so can add elements in the middle)