

Code in Place 2024

Stanford CS106A

Section - Week 6

Lists & Dictionaries

David Tsai, Code in Place

Today's Agenda



1. Check-In

How are we all doing?



2. Course Announcements

Diagnostic, Final Project



3. Concepts Review

Lists, Dictionaries, Mutability



4. Practice Problems

"Index Game", "List Practice",
"Heads Up"



Check-In

Welcome to our final live Section!

- What was your favorite problem, section, topic, or memory from your time in the course?

Course Announcements

Code in Place Diagnostic

- 50-minute self-assessment exam covering topics up through Week 5.
- Open-book.
- You will get feedback, but not be judged for pass/fail.
- Check the diagnostic instructions page for deadline.
- *Clicking the “Start Diagnostic” blue button will start the timer!*

Course Announcements

Final Project

- Start thinking about ideas for a program you might want to build.
 - Open ended, anything you can imagine.
 - Can be something utilitarian, or something creative and fun.
 - An animation or game on the console or canvas.
 - An app you can use in your personal life for family/work/study/etc.
- CIP 2020 Showcase:
 - <https://compedu.stanford.edu/codeinplace/public/>
- CIP 2021 Showcase:
 - <https://codeinplace.stanford.edu/2021/showcase/>

Please share your submitted project in our Section Forum, so we can all admire your wonderful creation!



Concepts Review

Intro to Data Structures: Lists & Dictionaries



- **Data Structure**

- Special container for organizing, processing, retrieving, and storing data.
- Examples: list, tuple, dictionary, set, tree, stack, queue, graph

- **List**

- An ordered collection of values.
- `games = ["Elden Ring", "Zelda", "Diablo 4", "Genshin Impact"]`

- **Dictionary**

- An unordered collection of key/value pairs.
- `movie_scores = {"Godfather": 97, "Avatar": 81, "Morbius": 15}`

Lists



- The elements of a list are **indexed, starting from 0**.

```
superheroes = ["Batman", "Superman", "Spider-Man", "Iron Man"]
```

Index	0	1	2	3
Element	"Batman"	"Superman"	"Spider-Man"	"Iron Man"

- Accessing individual elements:

```
>>> superheroes[0]  
"Batman"
```

```
>>> superheroes[2]  
"Spider-Man"
```

```
>>> superheroes[-1]          # Count in reverse.  
"Iron Man"
```


Let's Practice

"Rose"	"Donna"	"Martha"	"Amy"	"Clara"	"Bill"	"Yasmin"
--------	---------	----------	-------	---------	--------	----------

1. What index is "Rose"?
2. What index is "Clara"?
3. What index is "Clara"? (answer with a negative index)
4. What is the length of the list (the # of elements)?

(any "Doctor Who" fans, btw?)

Lists: Basics



- List creation
 - `empty_list = []`
 - `letters = ["a", "b", "c", "d", "e"]`
- Assigning new values to elements
 - `letters[3] = "x"` # letters -> ["a", "b", "c", "x", "e"]
- Length of list
 - `len(letters)` # 5
- Check if an element is in a list using the Python keyword `"in"`.
 - `if "b" in letters` # True
 - `if "z" in letters` # False

Lists: Other Useful Functions



- `list.append(elem)`
 - Add element to end of list.
- `list.pop()`
 - Remove element from end of list. Returns the element that was removed.
- `list.pop(index)`
 - Remove element from list at specified index.
- `list.remove(elem)`
 - Remove first occurrence of an element.
- `del list[index]`
 - Remove an element from a list at specified index. Doesn't return anything.
- `list1.extend(list2)`
 - Add all elements from list2 to the end of list1.
- ... [and many other useful functions in the Python Docs!](#)

Lists: How to Loop



- How do you loop over the following list?

```
letters = ["a", "b", "c", "d", "e"]
```

- **Method 1:** Using regular “for” loop with `range()`.

```
for i in range(len(letters)):          # len(letters) is 5
    print(letters[i])
```

- **Method 2:** Using “for-each” loop pattern.

```
for elem in letters:
    print(elem)
```

Dictionaries



- Dictionaries are similar to lists. They associate a key with a value.
 - Keys must be unique. Keys must be immutable types.

```
secret_identities = {"Batman": "Bruce Wayne", "Superman": "Clark Kent",  
                    "Spider-Man": "Peter Parker", "Iron Man": "Tony Stark"}
```

- In Python, you can also create a dictionary with the following formatting:

```
secret_identities = {  
    "Batman": "Bruce Wayne",  
    "Superman": "Clark Kent",  
    "Spider-Man": "Peter Parker",  
    "Iron Man": "Tony Stark"  
}
```

Dictionaries: Basics



```
empty_dict = {}  
ages = {"Chris": 33, "Julie": 22, "Mehran": 50}
```

- Using a key to access its associated value (**Method 1**).
 - `ages["Chris"]` # 33
 - `ages["Santa Claus"]` # `KeyError` (Program will crash.)
- Using a key to access its associated value (**Method 2**).
 - `ages.get("Chris")` # 33
 - `ages.get("Santa Claus")` # `None` (They keyword "`None`" is for null values.)
- Set a value to a key
 - `ages["Bronya"] = 25` # Will create a new key/value pair
 - `ages["Mehran"] = 18` # Will overwrite the existing value of 50
- Check if a key is in the dictionary using the Python keyword "`in`".
 - `if "Julie" in ages` # True
 - `if "Santa Claus" in ages` # False

Dictionaries: Other Useful Functions



- `len(dict)`
 - Returns the number of key/value pairs in the dictionary.
- `dict.pop(key)`
 - Removes key/value pair with the given key. Returns value from that pair.
- `del dict[key]`
 - Removes key/value pair with the given key. Doesn't return anything.
- `dict.keys()`
 - Returns something similar to a range of keys in the dictionary.
- `dict.values()`
 - Returns something similar to a range of values in the dictionary.
- ...[and many other useful functions in the Python Docs!](#)

Dictionaries: How to Loop



- **Method 1:** for-each `key`

```
for key in my_dict.keys():  
    value = my_dict[key]  
    print(key, value)
```

`.keys()` is optional. Can just use “my_dict”.

- **Method 2:** for-each `value`

```
for value in my_dict.values():  
    print(value)
```

- **Method 3:** for-each tuple of (`key`, `value`)

```
for key, value in my_dict.items():  
    print(key)  
    print(value)
```


Bonus Review

Mutability



- Different data types have different behaviors when **passed as parameters** to another function.

Types that are “immutable”	Types that are “mutable”
int, float, bool, string	list, dictionary, canvas
If passed as a parameter for a function: The original variable value you passed in is <u>not</u> changed when function is done.	If passed as a parameter for a function: The original variable value you passed in <u>is</u> changed when function is done.

Mutability: Examples



- **Strings are immutable:**

```
def main():  
    message = "Hello"  
    print(message)           # "Hello"  
    depart(message)  
    print(message)          # "Hello"  
  
def depart(message):         # Pass in string as a parameter.  
    message = "Goodbye"
```

- **Canvas is mutable:**

```
def main():  
    my_canvas = Canvas(300, 300)  
    draw_circle(my_canvas)   # Canvas is a mutable type.  
  
def draw_circle(canvas):     # Pass in a Canvas as a parameter.  
    canvas.create_oval(0, 0, 50, 50, "blue")  # The Canvas WILL be modified.
```

Mutability: Lists Are Mutable



- When you pass a list as a parameter, you are passing a reference to the actual list.
 - A reference is like getting a URL to the list.
 - Example: If I give you a URL to an editable Google Doc file, you can access the file using the URL and even make changes to it.
 - In a function, changes to values in list persist after the function ends.

```
def add_ten(num_list):  
    for i in range(len(num_list)):  
        num_list[i] += 10
```

```
def main():  
    values = [6, 7, 8, 9]  
    add_ten(values)           # Pass in a list as a parameter.  
    print(values)            # Output: [16, 17, 18, 19]
```

Mutability: Dictionaries Are Mutable



- Keys must be immutable types (e.g. `int`, `float`, `bool`, `string`).
 - Keys cannot be changed “in place”.
 - If you want to change a key, you must remove the key/value pair from dictionary and then add a key/value pair with new key.
- Values can be mutable or immutable types.
 - Values can be changed “in place”.
- Dictionaries are mutable.
 - Changes made to a dictionary in a function persist after the function is done.

```
def have_birthday(dict, name):  
    dict[name] += 1
```

```
def main():  
    ages = {"Chris": 33, "Julie": 22, "Mehran": 50}  
    print(ages)                    # Mehran's value is 50.  
    have_birthday(ages, "Mehran")  
    print(ages)                    # Mehran's value is 51.
```

Section Exercise: “Heads Up” Game



Take words from a text file and use them to recreate the “Heads Up” game!

Input

A text file (.txt) containing the following lines:*

Karel
For Loop
While Loop
If Statement
Else



Output

The word to guess: Karel
(Press “Enter” to get the next word.)

The word to guess: For Loop
(Press “Enter” to get the next word.)

The word to guess: While Loop
(Press “Enter” to get the next word.)

Useful random Library Functions for Lists



- **Randomly shuffling the items in a list:**

- `random.shuffle(a_list)`

```
fruits_list = ["apple", "banana", "cherry"]  
random.shuffle(fruits_list)  
print(fruits_list)                # Print the shuffled list
```

- **Returning a random item from a list:**

- `random.choice(a_list)`

```
random_fruit = random.choice(fruits_list)
```

How to Read from a Text File



- **Method 1:** Using the open-close pattern:

```
my_file = open("mydata.txt")
for line in my_file:
    line = line.strip()    # Gets rid of newline ("\n") character at end of line.
    print(line)
my_file.close()          # Close the file to free up program memory.
```

- **Method 2:** Using the with-as pattern:
 - At the end of the `with` block, the file is automatically closed.

```
with open("mydata.txt") as my_file:
    for line in my_file:
        line = line.strip()
        print(line)
```


Thank You

Congratulations! You've made it through CIP 2024!



- I had such a great time being your Section Leader!
- Feel free to connect with me on LinkedIn.
 - <https://www.linkedin.com/in/dwctsai/>
 - Would appreciate Endorsements for any Skills in my profile.

Please look forward to the remainder of the course!