Topic 6: Lists, Sets, Tuples, Dictionaries

Goals: By the end of this week, we will discuss…

- *what are lists, sets, tuples, and dictionaries (data structures)*
- *how to iterate over the data structures*
- *storing and passing lists with functions*

Acknowledgements: These class notes build on the content of my previous courses as well as the work of R. Jordan Crouser, and Jeffrey S. Castrucci.

Recall: Stings

One way to think about a **string** is as a **list/collection** of **characters**:

```
          name = "Smith College"
     ≈ ['S','m','i','t','h',' ','C','o','l','l','e','g','e']
         0   1   2   3   4   5   6   7   8   9   10  11  12
```

```
>>> name = "Smith College"
>>> name[2]              # Index of a letter.
'i'
>>> name[-4]
'l'
>>> name[:5]             # Substring / slicing
'Smith'
>>> name[2:]
'ith College'
>>> name[1:5]            #up to but not including 5
'mith'
```

- Strings are collections of characters, defined using "quotes".

- Lists are collections of objects, defined using [square brackets].

- Just about anything can go in a list, for example....

```
>>> [1, 2, 3, 4, 5, 6]        #list of integers
>>> [1.2, 3.5, 0.7, 7.8] #list of floats
>>> ["dog", "cat", "pig"]     #list of strings
```

- Lists can be indexed...just like strings

```
>>> animals = ["dog", "cat", "pig"]
>>> animals[1]
"cat"
>>> animals[-2]
"cat"
```

- Python allow for lists with mixed types, for example...

```
>>> [1, "cat", 7.8]
```

Other programming languages do not allow this, so use with extreme caution.

Naming convention

- Remember: it's always a a good idea variable names to be descriptive
- Because lists contain collections of things, we'll generally label them with a **plural noun**, for example....
- Just about anything can go in a list, for example....

```
numbers = [1, 2, 3, 4, 5, 6]
names = ["Jordan", "Ray", "Maisie"]
prices = [1.23, 3.55, 0.75, 7.80]
```

Iterating through items **in** a list

```
names = ["Jordan", "Ray", "Maisie"]
for name in names:
     print(name)
```

Checking membership **in** a list

```
names = ["Jordan", "Ray", "Maisie"]
new_name = input("Enter a student's name: ")

if new_name in names:
     print("They are in the class.")
else:
     print("Hmm, I don't know them.")
```

Exercise: Friends

- Create a list of your friends and assign it to a new variable.
- Then depending on you user either print each name in ALL CAPS or lower case letters.
- If ALL CAPS, include an exclamation mark at the end (use a loop).

Overwriting an item in a list

- If we want to overwrite an item in a list, we can use indexing combined with the = operator:

```
>>> animals = ['cat', 'dog', 'pig']
>>> animals[2] = 'rabbit'
>>> print(animals)
['cat', 'dog', 'rabbit']
```

Question: What happens when we try to do this with a string?

Answer: A TypeError

```
>>> animal = 'bat'
>>> animal[1] = 'd'
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    animal[1] = 'd'
TypeError: 'str' object does not support item assignment
```

Mutable vs. Immutable

- strings are immutable (which means we cannot change them in memory, we have to overwrite them completely)
- lists defined with […] are mutable (which means we can change them in memory)
- if we want an immutable lists, we can define them with (…) instead, for example:

```
>>> animals = ('cat', 'dog', 'pig')     #immutable
>>> animals[1]
'dog'
>>> animals[1] = 'bag'
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    animals[1] = 'bag'
TypeError: 'tuple' object does not support item assignment
```

List Operators

`.append()` - If you want to add a new item to the end of a list.
`.insert()` - If you want to add a new item into a list at a specific position.
`.remove()` - If you want to remove an item from a list, but if you try to remove an item that isn't in the list, the interpreter will throw a ValueError.
`.copy()` - If you want to copy the list.

For example:

```
>>> animals = ['cat', 'dog', 'pig', 'frog']  #mutable
>>> animals.append('turtle')
>>> animals.insert(3, 'fish')
>>> animals.remove('cat')
>>> print(animals)
['dog', 'pig', 'fish', 'frog', 'turtle']
>>> backup_animals = animals.copy()

>>> animals.remove('whale')
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    animals.remove('whale')
ValueError: list.remove(x): x not in list
```

- It is good practice to check if an element is in a list before removing it.

An important note about copying a list:

- Usually when we want to copy a string or a number, we just say something like: x2 = x1
- Copying a list this way, both the original and the copy point to the same spot in memory
- This can cause some unexpected behavior… remember when we said lists were mutable?

```
>>> animals = ['cat', 'dog', 'pig', 'frog']
>>> animals2 = animals
>>> animals.remove('dog')
>>> print(animals2)
['cat', 'pig', 'frog']             #Oops it was deleted from both.
```

List Operators (Continued)

`.count(..)` - If you want to count how many times an item appears in the list.

`.reverse()` - If you want to reverse the list.

`.sort()` - If you want to sort the list.

For example:

```
>>> animals = ['cat', 'dog', 'pig', 'frog', 'dog', 'pig']
>>> animals.count('dog')
2
>>> animals.reverse()
>>> print(animals)
['pig', 'dog', 'frog', 'pig', 'dog', 'cat']
>>> animals.sort()
>>> print(animals)
['cat', 'dog', 'dog', 'frog', 'pig', 'pig']
```

Exercise: Friends (Part 2)

Instead, write a program that:
- asks the user to input() names one at a time
- adds each new name to a list called friends
- after each new name is added prints the list in alphabetical order

The program should loop until the user types “DONE”

Answer:

```
friends = []
name = input("Enter a friend's name or DONE: ")
while(name != "DONE"):
    friends.append(name)
    friends.sort()
    print(friends)
    name = input("Enter a friend's name or DONE: ")
```

Next: Imagine we want to use the previous exercise to create a contact list. (see Dictionaries)

Lists of Lists

You can put a list inside a list.

For example, here is how I might store our cloths.

```
>>> cloths = [['top', 'blue', 'short sleeve'],
          ['top', 'red', 'graphic telephone'],
          ['bottom', 'blue', 'fashion jeans']]
>>> print(cloths)
[['top', 'blue', 'short sleeve'], ['top', 'red', 'graphic
telephone'], ['bottom', 'blue', 'fashion jeans']]
>>> print(cloths[1])
['top', 'red', 'graphic telephone']
>>> print(cloths[1][0])
top
>>> cloths.append(['dress', 'black', 'cocktail'])
>>> print(cloths)
[['top', 'blue', 'short sleeve'], ['top', 'red', 'graphic
telephone'], ['bottom', 'blue', 'fashion jeans'], ['dress', 'black',
'cocktail']]
```

Advanced Topic (Optional)

You can define lists using a loop.

```
>>> n = 3
>>> grids = [[0]*n for row in range(n)]
>>> grids
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

- This is called a list comprehension,
- It is a shorthand way to define a list according to a rule.

Dictionaries

- Today we learn how to store and retrieve elements with dictionaries.

Recap: Exercise: Friends (Part 2)

Write a program that: (a) asks the user to input() names one at a time, (b) adds each new name to a list called friends, and (c) after each new name is added prints the list in alphabetical order. The program should loop until the user types “DONE”

Answer:

```
friends = []
name = input("Enter a friend's name or DONE: ")
while(name != "DONE"):
    friends.append(name)
    friends.sort()
    print(friends)
    name = input("Enter a friend's name or DONE: ")
```

- Imagine we want to use the previous exercise to create a contact list.

Could do it with multiple lists:

```
def friendBook():
    instruction = "ADD"
    friends = []
    numbers = []

    while (instruction != "DONE"):
        # Get information about new contact
        friends.append(input("Name? "))
        numbers.append(input("Number? "))

        #Ask for next instruction.
        instruction = input("ADD or DONE?")
    print("Friends:", friends)
    print("Numbers:", numbers)
```

BUT....

- This is very annoying if you want to access the data.

```
    print(friends[0])
    print(numbers[0])
```

- Worse to modify the data.

```
    friends.remove('John')
    numbers.remove('413-555-2936')
```

Motivation: Each name should “map” to the corresponding number:

“Suzy” -> “413-286-3712”
“Alison” -> “972-272-2782”
“Clio” -> “291-288-2897”

That way, we could access the number using the name:

```
contacts["Suzy"]
```
“413-286-3712”

Dictionaries

- **lists** were **ordered** sets of objects, and we accessed their contents via position (index)
- **dictionaries** are **unordered** sets, and we can access their contents via **keys**
  - declare them using {…} <- “curly braces” like this...

```
contacts = {}
```

For example:

```
def friendDictionary():
    instruction = "ADD"
    contacts = {}

    while (instruction != "DONE"):
        # Get information about new contact
        new_friend = input("Name? ")
        new_number = input("Number? ")

        #Add contact to dictionary
        contacts[new_friend] = new_number

        #Ask for next instruction.
        instruction = input("ADD or DONE?")
```

What happens when we iterate over a dictionary?

```
    for thing in contacts:
        print(thing)
```

... not exactly what we hoped.

Question: How are these dictionaries different that language dictionaries (in books or online)?

Exercise: Course Dictionary

Consider the courses you are taking this term.
Create a list to store the course codes, and another list to store the course titles.
Use a loop to add each course to your new course dictionary.
Hint: key is the course code, value is the course title.

Dictionary Methods

`.keys()` - If you want to get a list of the keys in a dictionary.
`.values()` - If you want a list of the values in a dictionary.
`.items()` - If you want a list of the (key, value) pairs in a dictionary.
`.pop()` - If you want to remove an item from the dictionary.
`.copy()` - If you want to copy the dictionary (same as lists).
`.zip()` - Combine two lists into a dictionary.

For example...

```
#contacts is my dictionary made above
>>> print("Keys:", contacts.keys())
Keys: dict_keys(['Miranda', 'Kris', 'Jeffrey', 'Oliver', 'Leah', 'Fatima'])
>>> print("Values:", contacts.values())
Values: dict_values(['413-555-6472', '413-555-2349', '413-555-0204', '413-555-6193',
'413-555-9328', '413-555-0385'])
>>> for key, value in contacts.items():
        print(key, value)
Miranda 413-555-6472
Kris 413-555-2349
Jeffrey 413-555-0204
Oliver 413-555-6193
Leah 413-555-9328
Fatima 413-555-0385
```

`.zip()`

If you want to combine two lists into one dictionary, use a comprehension and the zip(…) function:

```
initial_names = ['Miranda', 'Kris', 'Fatima']
initial_numbers = ['413-555-6472', '413-555-2349', '413-555-0385']
contacts = {name:number for name, number in zip(initial_names,
initial_numbers)}
```

Recap

- strings: immutable ordered collections of characters
- lists: mutable ordered collections of objects
- dictionaries: mutable unordered collections of objects

Passing “by reference”

What does this mean when we pass a list / dictionary as input to a function?

```
def modifyFriends(my_dict):
    my_dict['Kris'] = '413-444-6472'
    my_dict.pop('Oliver')
    my_dict['Shelley'] = '413-555-1010'
    return my_dict

def main():
    contacts = dictionaryOperations()
    new_contacts = modifyFriends(contacts)
    print()
    print(contacts)
    print(new_contacts)
```

This results in contacts and new_contacts being the same.

Exercise: Course Dictionary Con't

Add to the program you wrote above, to allow for changes during the add/drop period. Loop until the user enters 'DONE' and allow for additions and removals from your course dictionary.

```
code = ['CSC111', 'FRN363', 'ARH278', 'ENG327', 'ESS975']
title = ['Introduction to Computer Science Through Programming',
         'Crossing the Divide: Love, Ambition, and the Exploration of
Social Difference',
         'Race and Gender in the History of Photography',
         'Robin Hood: Legendary Outlaw',
         'Yoga Hatha Yoga I']
course_dictionary = {key:value for key, value in zip(code, title)}
print(course_dictionary)
response = input("(A)dd, (R)emove, or (D)one:")
while (response.upper() != 'D'):
    if (response.upper() == 'A'):
        new_code = input("Code? ")
        new_title = input("Title? ")
        course_dictionary[new_code] = new_title
    elif (response.upper() == 'R'):
        print(course_dictionary.keys())
        del_code = input("Code to remove? ")
        course_dictionary.pop(del_code)
    print("Your courses are:")
    for key, value in course_dictionary.items():
        print(key, value)
    response = input("(A)dd, (R)emove, or (D)one:")
```

Learning Reflection

Take 3-5 minutes,

1. What constructs/concepts am I most comfortable with?
2. What constructs/concepts am I most confused/fuzzy about?
3. What do I wish I had done differently in this course?