ITEC 1150 Week 8 Chapter 5

THE DICTIONARY DATA STRUCTURE

Agenda

- This week, our focus will be on a new data type, **Dictionaries**.
- Before getting into dictionaries though, we'll briefly look at another type, **Tuples**, and some methods on strings that make working with user input easier.

Tuples

- A **Tuple** is exactly like a list, with two important differences:
  - It uses parenthesis instead of square brackets:

```
1. grades = [99, 98, 97] # list
2. grades = (99, 98, 97) # tuple
```

  - A tuple is *immutable*, which means it cannot be changed after it is created.

- We won't need to use tuples in our labs (although you can if you like), but it's important to know what they are so you don't accidentally create one without realizing it, and because some library methods will return tuples instead of lists.
- Just remember that you can treat a tuple like a list for looping and printing.

String Case Functions

- Here are some basic functions that you will need in the labs this week. We'll look at them (and other functions) in more detail next week.

```
1. string = 'ExAmPlE sEntENCE.'
2. print("Upper:\t" + string.upper())
3. print("Lower:\t" + string.lower())
4. print("Cap:\t" + string.capitalize())
5. print("Title:\t" + string.title())
```

- These can come in handy when comparing user input to data in your program and for making your output look better.

```
Upper:  EXAMPLE SENTENCE.
Lower:  example sentence.
Cap:    Example sentence.
Title:  Example Sentence.
```

Dictionaries

Our second data structure

- Last week we looked at Lists, which are the simplest and most used data structure.
- Sometimes, we need more capabilities from a data structure, such as the ability to create a relationship between pieces of data.
- **Dictionaries** provide just that capability.

From Lists to Dictionaries

- A list is fine when all the elements in the list have the same meaning, such as a list of grades, a list of book prices, and so on.
- A list *isn't* fine when you have pieces of data with different meanings, such as:
  - Student and grade
  - Book title and price
  - Year and yearly average rainfall
- For these situations, a dictionary is a good choice.
- As we'll see, combining lists with dictionaries is even more powerful.

Dictionary Syntax

```
1. student_grades = {
2.     'Zaid Zahrah': 99,
3.     'Heather Ferguson': 88,
4.     'Naveed Sanaa': 77,
5.     'Glenn Ferguson': 66
6. }
```

- Dictionaries are made up of **key : value** pairs
- The **key** identifies the **value** in the dictionary.
- Keys and values can be any data type and can even be data structures.

Accessing Dictionary Values

THERE ARE MULTIPLE METHODS, WITH DIFFERENT REASONS TO USE THEM

Accessing Dictionary Values

- Using the **key** inside brackets [ ] is the simplest way…

```
1. student_grades = {
2.     'Zaid Zahrah': 99,
3.     'Heather Ferguson': 88,
4.     'Naveed Sanaa': 77,
5.     'Glenn Ferguson': 66
6. }
7. student = 'Zaid Zahrah'
8. print(f'{student} scored {student_grades[student]} on the last quiz.')
```

- Line 8 in the code shows how we can access the value of the dictionary using the key.

Output:

```
Zaid Zahrah scored 99 on the last quiz.
```

Accessing Dictionary Values, cont.

- …but it has drawbacks:

```
1. student = 'Joe Schmoe'
2. print(f'{student} scored {student_grades[student]} on the last quiz.')
```

- Trying to access a key that doesn't exist gives us an error:

```
Traceback (most recent call last):
  File "/Users/egranse/Desktop/temp.py", line 9, in <module>
    print(f'{student} scored {student_grades[student]} on the last quiz.')
                             ~~~~~~~~~~~~~~~^^^^^^^^^
KeyError: 'Joe Schmoe'
```

Accessing Dictionary Values, cont.

- One method to avoid key errors is to check if the key exists in the dictionary:

```
1. student = 'Joe Schmoe'
2. if student in student_grades:
3.     print(f'{student} scored {student_grades[student]} on the last quiz.')
4. else:
5.     print(f"No score found for {student}.")
```

Output:

```
No score found for
Joe Schmoe
```

The .get() Method

- Using the method .get(key, default_value) allows us to specify a value that will be returned if the key doesn't exist:

```
1. student = 'Joe Schmoe'
2. print(f'Most recent quiz score for {student}: {student_grades.get(student, 'N/A')}')
3. student = 'Zaid Zahrah'
4. print(f'Most recent quiz score for {student}: {student_grades.get(student, 'N/A')}')
```

- Because 'Joe Schmoe' isn't a key in the student_grades dictionary, using .get(student, 'N/A') returns 'N/A'.

Output:

```
Most recent quiz score for Joe Schmoe: N/A

Most recent quiz score for Zaid Zahrah: 99
```

Keys, Values, and Items

The examples we've seen so far assume we know the key we want. What if we don't know the key, or we want to perform an operation on all entries in the dictionary? That's where these methods come in handy:

- `keys()`
- `values()`
- `items()`

Looping Through a Dictionary

- The default iterator for dictionaries iterates over the keys:

```
1. for student in student_grades:
2.     print(student, '\t', student_grades[student])
```

- Using the `keys()` method

```
1. for student in student_grades.keys():
2.     print(student, '\t', student_grades[student])
```

- Both give the same output.

Output:

```
Zaid Zarah          99
Heather Ferguson    88
Naveed Sanaa        77
Glenn Ferguson      66
```

Unpacking Keys and Values At The Same Time

- The `items()` method gives us an iterable tuple with both the key and value:

```
1. for student, grade in student_grades.items():
2.     print(student, '\t', grade)
```

- The output is still the same, but you get two variables to work with in your program. This can be more convenient in many situations.

Output:

```
Zaid Zarah          99
Heather Ferguson    88
Naveed Sanaa        77
Glenn Ferguson      66
```

Loop Through Values Only

- the `values()` method is more obvious in its difference:

```
1. for grade in student_grades.values():
2. print(grade)
```

- The output includes values only, not keys.

Output:

99

88

77

66

Adding and Removing Dictionary Values

THERE ARE MULTIPLE METHODS, WITH DIFFERENT REASONS TO USE THEM

Adding or Editing Dictionary Data

- Adding a new student grade:

```
1. student_grades['Mandy Rice'] = 100
```

- Modifying a student grade (changing the value stored for a key):

```
1. student_grades['Glenn Ferguson'] = 85
```

- They're exactly the same! This means that a key can exist exactly once in a dictionary.

Deleting an Item

```
1. del student_grades['Glenn Ferguson']
2. del student_grades['Joe Schmoe']
```

- Trying to delete `Joe Schmoe` will cause an error because that key doesn't exist.
- We need to check if the key exists before deleting:

```
1. student = 'Joe Schmoe'
2. if student in student_grades:
3.     del student_grades[student]
4.     print(f"{student} deleted from the class list.")
5. else:
6.     print(f"{student} was not present in the class list.")
```

The .pop() And .clear() Methods

- The `.pop()` method will remove a key and return the matching value. We can provide a default to prevent errors:

```
1. student = 'Joe Schmoe'
2. deleted_grade = student_grades.pop(student, 'N/A')
3. print(f"{student} with grade {deleted_grade} is no longer in the class list.")
```

- The `.clear()` method will remove *all* keys and values from the dictionary. This can't be undone!

```
1. student_grades.clear()
2. print("Student grades:", student_grades)
```

Scenarios & Examples

SOME PRACTICAL WAYS IN WHICH TO USE DICTIONARIES

Dictionary Examples

```
1.  # you can have strings as both keys and values
2.  countries = {'CA': 'Canada', 'US': 'United States', 'MX': 'Mexico', 'GB': 'Great Britain'}
3.
4.  # you can have numbers as keys, strings as values
5.  numbers = {1: 'One', 2: 'Two', 3: 'Three', 4: 'Four', 5: 'Five'}
6.
7.  # you can have strings as keys, mixed types as values
8.  book = {'name': 'Automate the Boring Stuff', 'year': 2018, 'price': 18.99 }
9.
10. # how to initialize a variable as an empty dictionary
11. book_catalog = {}
```

String Functions With User Input

- Line 4 shows us adjusting the user input to title case so it will match the case of the keys (which are the movie names). If we didn't do this, the user would have to capitalize exactly as we had it in our map.

```
1. movies = {'The Apartment': 'Billy Wilder', 'The Maltese Falcon': 'John Houston',
   'Casablanca': 'Michael Curtiz'}
2.
3. movie = input("Enter a movie name to see the director: ")
4. movie = movie.title() # convert the input to title case to match keys
5. if movie in movies:
6.     print(f"The director was {movies[movie]}")
7. else:
8.     print(f"{movie} not found.")
```

Multiple Dictionaries With Common Keys

- Our examples so far have looked at simple key/value pairs, where we kept pairs of data together, for example, a name and a grade.
- Sometimes though, we have lots of regular data. Think of a book—it has a title, author, price, publication date, ISBN, and so forth. Holding multiple book keys in a dictionary wouldn't work.
- Instead, we can create a dictionary per book:

```
1. book_1 = { 'title': 'Python Primer', 'author': 'Alice Smith', 'price': 34.95 }
2. book_2 = { 'title': 'Java Jumpstart', 'author': 'Hassan Hassan', 'price': 91.25}
3. book_3 = { 'title': 'Clojure Code', 'author': 'Bill Bower', 'price': 14.50}
```

- Then we can put the book dictionaries into **another** data structure…

A List of Dictionaries

```
1. book_1 = { 'title': 'Python Primer', 'author': 'Alice Smith', 'price': 34.95 }
2. book_2 = { 'title': 'Java Jumpstart', 'author': 'Hassan Hassan', 'price': 91.25}
3. book_3 = { 'title': 'Clojure Code', 'author': 'Bill Bower', 'price': 14.50}
4. book_list = [book_1, book_2, book_3]
5.
6. for book in book_list:
7.     print(f"{book['title']} was written by {book['author']} and costs ${book[price']}.")
```

- Each one of those dictionaries is now a 'record' of an individual book.

A Dictionary Of Dictionaries: A Basic Database

- Even more powerfully, can combine our book 'records' with another dictionary to get a book by its title and then get all the book information:

```
1.  library = {}
2.  library['Python Primer'] = { 'title': 'Python Primer', 'author': 'Alice Smith', 'price': 34.95 }
3.  library['Java Jumpstart'] = { 'title': 'Java Jumpstart', 'author': 'Hassan Hassan', 'price': 91.25}
4.  library['Clojure Code'] = { 'title': 'Clojure Code', 'author': 'Bill Bower', 'price': 14.50}
5.
6.  title = input("Enter a book title: ").title()
7.  if title in books:
8.      print(books[title])
9.  else:
10.     print("Book not found.")
```