Welcome!

We'll get started shortly …

CS 49 Section

Week 11

Surajit A Bose

Agenda

- Logistics and check-ins
- Review of lecture concepts
  - Dictionaries
  - Dictionary practice
- Section Problem: Find Grandchildren

Logistics

How to get hold of me / get help

- The class forum. Feel free not only to ask, but also to answer questions!
- Surajit's office hours:
  - ~~Fridays 12 noon–1p, directly after section~~ (Today is the last day!)
  - By appointment on Zoom
- ~~Lane's office hours~~
- Canvas inbox or Pronto inbox for Lane
- Canvas inbox (preferred) or Pronto for Surajit
- Sina's support section, Fridays 1p–2p on Zoom
- Email bosesurajit@fhda.edu, 24 hr turnaround
- ~~Online or in-person tutoring via the STEM center (Room 4213)~~
- The section GitHub repo has lecture and section slides and solutions

Lecture Review: Dictionaries

Introducing Dictionaries

- One container type in Python is dictionaries (`dict`)
- A dictionary enables very fast lookups of a **`value`** via an associated **`key`**
- For example, if the college needs to look up info about a student:
  - Looking up via the CWID will return information about the student's name, enrollment status, major, GPA, etc.
  - The CWID is the key, the student info is the value
- Thus, a dictionary is a collection of **`key : value`** pairs
- Conceptual examples:
  - Country as key, capital as value ('Canada' : 'Ottawa')
  - Hex value as key, color name as value (0xb70b2f : 'cardinal ')

Creating and Modifying Dictionaries

- To create an empty dictionary: `{}` or `dict()`

```
my_dict = {}
```

- To create a new non-empty dictionary:

```
state_caps = {'Oregon' : 'Salem', 'Idaho': 'Boise',
    'Hawaii' : 'Honolulu', 'Alaska' : 'Juneau'}
```

- To add a key : value pair to the dictionary: `dict[key] = value`

```
state_caps['Maryland'] = 'Annapolis'
state_caps['Kentucky'] = 'Lexington'
```

- Oops! To update an existing value, use the same syntax:

```
state_caps['Kentucky'] = 'Frankfort'
```

- This will replace the existing value for that key with the new value

Accessing Data in Dictionaries

- To access a value, use the key:

  ```
  md_cap = state_caps['Maryland']
  # md_cap now has the value 'Annapolis'
  ```

- Trying to access the value for a non-existent key results in an error:

  ```
  ca_cap = state_caps['California']    # KeyError
  ```

- To see if a particular key is in a dictionary, use `in`:

  ```
  print('California' in state_caps)    # prints False
  print('Hawaii' in state_caps)        # prints True
  ```

- To see if a particular value is in a dictionary, use `in dict.values()`:

  ```
  print('Boise' in state_caps.values()) # prints True
  ```

Constraints on Dictionary Keys

- Keys must be unique. Cannot have:

```
mixed_caps = {'Georgia' : 'Tbilisi',
    'Georgia' : 'Atlanta'}    # 'Tbilisi' is replaced
```

- Keys must be immutable. Recall:
  - Atomic types (`int`, `Boolean`, `float`) are immutable
  - Of the container types, `str` is immutable
- Lists are mutable, so a `list` cannot be used as a key
- These constraints do not apply to values
  - Two different keys can have the same associated value
  - Values can be mutable types like lists or even other dictionaries

Iterating over Dictionaries

- To iterate over all key : value pairs: `dict.items()`

```
for state, city in state_caps.items():
    print(f'The capital of {state} is {city}'}
```

- To iterate over just the keys:

```
for state in state_caps:
    print(state)
```

- To iterate over just the values, use `dict.values()`:

```
for city in state_caps.values():
    print(city)
```

- Likewise, there is also a `dict.keys()`

Dictionaries vs Lists

- We've seen that data types can be:
  - atomic (`int`, `float`, `Boolean`, `None`) or container (`str`, `list`)
  - immutable (`str`, any atomic type) or mutable (`list`)
- `dict` is a mutable container type, like `list`
- Data types can also be ordered or unordered
  - `list` and `str` are ordered: data is consecutive by index
  - `list` and `str` indices are always integers
- `dict` is unordered: the sequence is arbitrary
- `dict` keys can be any immutable value (including integers, which could be non-consecutive)

Any Questions?

Dictionaries Practice

https://codeinplace.stanford.edu/foothill-cs49/ide/p/sy6eC9jGvP11LL9qp8dV

Dictionaries: Key-Value Pairs

- I have three dogs. Rover is a lab, Renly a mutt, and Spot a pug. Create a new dictionary **breeds** with the names as keys, the breeds as values.

Dictionaries: Key-Value Pairs

- I have three dogs. Rover is a lab, Spot a mutt, and Max a pug. Create a new dictionary **breeds** with the names as keys, the breeds as values.

```
breeds = {
    'Rover' : 'lab',
    'Spot' : 'mutt',
    'Max' : 'pug'
}
```

Dictionaries: Key-Value Pairs

- I have three dogs. Rover is a lab, Spot a mutt, and Max a pug. Create a new dictionary **breeds** with the names as keys, the breeds as values.

```
breeds = {
    'Rover' : 'lab',
    'Spot' : 'mutt',
    'Max' : 'pug'
}
```

- Fetch Rover's breed and store it in a variable called **rover_breed**

Dictionaries: Key-Value Pairs

- I have three dogs. Rover is a lab, Spot a mutt, and Max a pug. Create a new dictionary **breeds** with the names as keys, the breeds as values.

```
breeds = {
    'Rover' : 'lab',
    'Spot' : 'mutt',
    'Max' : 'pug'
}
```

- Fetch Rover's breed and store it in a variable called **rover_breed**

```
rover_breed = breeds['Rover']
```

Dictionaries: Key-Value Pairs

- I have a new dog! Her name is Fifi. She is also a mutt. Add Fifi to the **breeds** dictionary.

Dictionaries: Key-Value Pairs

- I have a new dog! Her name is Fifi. She is also a mutt. Add Fifi to the **breeds** dictionary.

```
    breeds['Fifi'] = 'mutt'
```

Dictionaries: Key-Value Pairs

- I have a new dog! Her name is Fifi. She is also a mutt. Add Fifi to the **breeds** dictionary.

```
breeds['Fifi'] = 'mutt'
```

- Oops! Fifi is very upset. She says she is not a mutt, she is a poodle!

Dictionaries: Key-Value Pairs

- I have a new dog! Her name is Fifi. She is also a mutt. Add Fifi to the **breeds** dictionary.

```
    breeds['Fifi'] = 'mutt'
```

- Oops! Fifi is very upset. She says she is not a mutt, she is a poodle!

```
    breeds['Fifi'] = 'poodle'
```

Dictionaries: Key-Value Pairs

- I have a new dog! Her name is Fifi. She is also a mutt. Add Fifi to the **breeds** dictionary.

```
breeds['Fifi'] = 'mutt'
```

- Oops! Fifi is very upset. She says she is not a mutt, she is a poodle!

```
breeds['Fifi'] = 'poodle'
```

- Hm, I can't remember whether Max is in the dictionary. Can you check?

Dictionaries: Key-Value Pairs

- I have a new dog! Her name is Fifi. She is also a mutt. Add Fifi to the **breeds** dictionary.

```
    breeds['Fifi'] = 'mutt'
```

- Oops! Fifi is very upset. She says she is not a mutt, she is a poodle!

```
    breeds['Fifi'] = 'poodle'
```

- Hm, I can't remember whether Max is in the dictionary. Can you check?

```
    print('Max' in breeds)     # prints True
```

Dictionaries: Key-Value Pairs

- I have a new dog! Her name is Fifi. She is also a mutt. Add Fifi to the **breeds** dictionary.

```
breeds['Fifi'] = 'mutt'
```

- Oops! Fifi is very upset. She says she is not a mutt, she is a poodle!

```
breeds['Fifi'] = 'poodle'
```

- Hm, I can't remember whether Max is in the dictionary. Can you check?

```
print('Max' in breeds)    # prints True
```

- Print out all the key-value pairs.

Dictionaries: Key-Value Pairs

- I have a new dog! Her name is Fifi. She is also a mutt. Add Fifi to the **breeds** dictionary.

```
breeds['Fifi'] = 'mutt'
```

- Oops! Fifi is very upset. She says she is not a mutt, she is a poodle!

```
breeds['Fifi'] = 'poodle'
```

- Hm, I can't remember whether Max is in the dictionary. Can you check?

```
print('Max' in breeds)    # prints True
```

- Print out all the key-value pairs.

```
for dog, breed in breeds.items():
    print(f'{dog} is a {breed}')
```

Section problem: Find Grandchildren

https://codeinplace.stanford.edu/cs49-w24/ide/a/findgrandchildren

Find Grandchildren

- Given a `dict` with parents as keys, and lists of their children as values:

```
parents_dict = { 'Khaled' : ['Chibundu', 'Jesmyn'],
    'Daniel' : ['Khaled', 'Eve'],
    'Jesmyn' : ['Frank'],
    'Eve': ['Grace'] }
```

- Create a `dict` with grandparents as keys, and grandchildren as values

Find Grandchildren

- Given a **dict** with parents as keys, and lists of their children as values:

```
parents_dict = { 'Khaled' : ['Chibundu', 'Jesmyn'],
     'Daniel' : ['Khaled', 'Eve'],
     'Jesmyn' : ['Frank'],
     'Eve': ['Grace'] }
```

- Create a **dict** with grandparents as keys, and grandchildren as values
- Only actual grandparents should be keys — i.e., no empty lists for values

```
parents_dict = {
     'Jesmyn' : [],
     'Eve': [] }
```

Find Grandchildren

- Given a dict with parents as keys, and lists of their children as values:

```
parents_dict = { 'Khaled' : ['Chibundu', 'Jesmyn'],
    'Daniel' : ['Khaled', 'Eve'],
    'Jesmyn' : ['Frank'],
    'Eve': ['Grace'] }
```

- Create a dict with grandparents as keys, and grandchildren as values
- How should we identify the needed key : value pairs?

Find Grandchildren

- Given a **dict** with parents as keys, and lists of their children as values:

```
parents_dict = { 'Khaled' : ['Chibundu', 'Jesmyn'],
    'Daniel' : ['Khaled', 'Eve'],
    'Jesmyn' : ['Frank'],
    'Eve': ['Grace'] }
```

- Create a **dict** with grandparents as keys, and grandchildren as values
- How should we identify the needed key : value pairs?
- For keys, consider:
  - Is Khaled a grandparent? How do we know?
  - How about Daniel? Jesmyn? Eve?

Find Grandchildren

- Given a **dict** with parents as keys, and lists of their children as values:

```
    parents_dict = { 'Khaled' : ['Chibundu', 'Jesmyn'],
        'Daniel' : ['Khaled', 'Eve'],
        'Jesmyn' : ['Frank'],
        'Eve': ['Grace'] }
```

- Create a **dict** with grandparents as keys, and grandchildren as values
- How should we identify the needed key : value pairs?
- For keys, consider:
    - Is Khaled a grandparent? How do we know?
    - How about Daniel? Jesmyn? Eve?
- A parent **X** is a grandparent if any of their children is also a parent

Find Grandchildren

- Given a `dict` with parents as keys, and lists of their children as values:

```
parents_dict = { 'Khaled' : ['Chibundu', 'Jesmyn'],
    'Daniel' : ['Khaled', 'Eve'],
    'Jesmyn' : ['Frank'],
    'Eve': ['Grace'] }
```

- Create a `dict` with grandparents as keys, and grandchildren as values
- How should we identify the needed key : value pairs?
- A key **`X`** in **`parents_dict`** should be a key in **`grandparents_dict`** if any of the list elements in the value **`parents_dict[X]`** is itself a key in **`parents_dict`**

Find Grandchildren

- Given a **dict** with parents as keys, and lists of their children as values:

```
parents_dict = { 'Khaled' : ['Chibundu', 'Jesmyn'],
    'Daniel' : ['Khaled', 'Eve'],
    'Jesmyn' : ['Frank'],
    'Eve': ['Grace'] }
```

- Create a **dict** with grandparents as keys, and grandchildren as values
- How should we identify the needed key : value pairs?
- For values, consider:
  - Who are Khaled's grandchildren?
  - How about Daniel's?

Find Grandchildren

- Given a **`dict`** with parents as keys, and lists of their children as values:

```
parents_dict = { 'Khaled' : ['Chibundu', 'Jesmyn'],
      'Daniel' : ['Khaled', 'Eve'],
      'Jesmyn' : ['Frank'],
      'Eve': ['Grace'] }
```

- Create a **`dict`** with grandparents as keys, and grandchildren as values
- How should we identify the needed key : value pairs?
- For values, consider:
  - Who are Khaled's grandchildren?
  - How about Daniel's?
- **X** 's grandchildren are *all* the children of *each* of their children

Find Grandchildren

- Given a **dict** with parents as keys, and lists of their children as values:

```
parents_dict = { 'Khaled' : ['Chibundu', 'Jesmyn'],
    'Daniel' : ['Khaled', 'Eve'],
    'Jesmyn' : ['Frank'],
    'Eve': ['Grace'] }
```

- Create a **dict** with grandparents as keys, and grandchildren as values
- How should we identify the needed key : value pairs?
- To construct the value at **grandparents_dict[X]**:
  - Identify the children in **parents_dict[X]** who are themselves keys in **parents_dict**
  - Combine their values in **parents_dict** into a single list

Find Grandchildren

- Given a **dict** with parents as keys, and lists of their children as values:

```
parents_dict = { 'Khaled' : ['Chibundu', 'Jesmyn'],
    'Daniel' : ['Khaled', 'Eve'],
    'Jesmyn' : ['Frank'],
    'Eve': ['Grace'] }
```

- Create a **dict** with grandparents as keys, and grandchildren as values
- Desired output:

```
grandparents_dict = { 'Khaled' : ['Frank'],
    'Daniel' : ['Chibundu', 'Jesmyn', 'Grace'] }
```

Questions Before We Begin?

Fun stuff

Find Grandchildren

- Given a `dict` with parents as keys, and lists of their children as values:

```
parents_dict = { 'Khaled' : ['Chibundu', 'Jesmyn'],
    'Daniel' : ['Khaled', 'Eve'],
    'Jesmyn' : ['Frank'],
    'Eve': ['Grace'] }
```

- Just for kicks: what will the following print out?

```
print(parents_dict[parents_dict['Khaled'][1]][0])
```

Find Grandchildren

- Given a `dict` with parents as keys, and lists of their children as values:

```
parents_dict = { 'Khaled' : ['Chibundu', 'Jesmyn'],
    'Daniel' : ['Khaled', 'Eve'],
    'Jesmyn' : ['Frank'],
    'Eve': ['Grace'] }
```

- Just for kicks: what will the following print out?

```
print(parents_dict[parents_dict['Khaled'][1]][0])
# Output: Frank
```

Find Grandchildren

- Given a `dict` with parents as keys, and lists of their children as values:

```
parents_dict = { 'Khaled' : ['Chibundu', 'Jesmyn'],
     'Daniel' : ['Khaled', 'Eve'],
     'Jesmyn' : ['Frank'],
     'Eve': ['Grace'] }
```

- Just for kicks: what will the following print out?

```
print(parents_dict[parents_dict['Khaled'][1]][0])
```

- `parents_dict['Khaled']` is `['Chibundu', 'Jesmyn']`
- `parents_dict['Khaled'][1]` is `'Jesmyn'`
- `parents_dict[parents_dict['Khaled'][1]]` is `['Frank']`
- `parents_dict[parents_dict['Khaled'][1]][0]` is `'Frank'`

What's Next?

More Python at FHDA

- De Anza's Python Sequence:
  - CIS 40, Introduction to Programming in Python
  - CIS 41A, Python Programming
  - CIS 9, Introduction to Data Science
  - CIS 41B, Advanced Python Programming
- Foothill's Python Sequence:
  - CS 3A, Object Oriented Programming Methodologies in Python
  - CS 3B, Intermediate Software Design in Python
  - CS 3C, Advanced Data Structures and Algorithms in Python
  - CS 8A, Introduction to Data Science
  - CS 48A, Data Visualization

That's all, folks!