DICTIONARIES

COMP130 – INTRODUCTION TO COMPUTING

DICKINSON COLLEGE

DICTIONARY (A.K.A MAP)

- A ***dictionary*** is a data structure that ***maps*** a ***key*** to an associated ***value***.
  - Real World Examples:

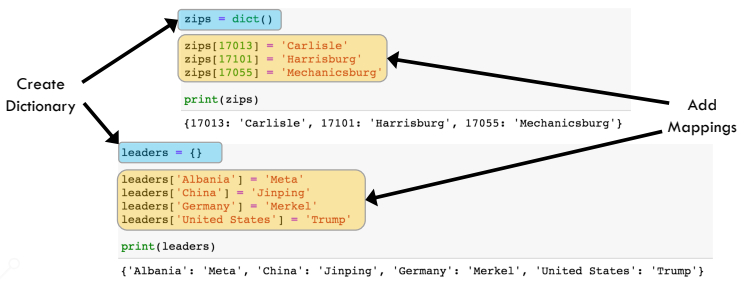
Key		Value
Student ID	→	Student
License Plate	→	Vehicle
Zip Code	→	Location

  - Keys and values come in pairs (***key-value pairs***)
    - The ***key*** uniquely identifies a value.
    - The ***value*** is associated with the key.
      - A value may be associated with multiple keys.
        - E.g. The same location may have multiple zip codes.

17013	→	Carlisle, PA
17015	→	Carlisle, PA

PYTHON DICTIONARIES

- In Python a dictionary is an *object* that efficiently maps keys to values.
  - Dictionaries are created using the `dict()` constructor or the `{ }` shorthand.
  - Mappings are added using the `[ ]` notation.



DICTIONARY LOOKUPS

- A dictionary ***lookup*** uses a *key* to find the associated *value*.
  - Lookups can use either the `get` method in the `dict` class or the `[ ]` shorthand.

```
print(zips.get(17055))
print(zips[17013])
```

```
Mechanicsburg
Carlisle
```

```
print(leaders.get('Albania'))
print(leaders['China'])
```

```
Meta
Jinping
```

  - Looking up a key that is not in the dictionary generates a `KeyError`:

```
print(zips[90201])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-112-310a0376e3a0> in <module>
----> 1 print(zips[90201])

KeyError: 90201
```

GETTING KEYS AND VALUES

- The dictionary provides access to the keys and values separately.

```
zip_keys = zips.keys()
print(zip_keys)
```

```
dict_keys([17013, 17101, 17055])
```

```
zip_vals = zips.values()
print(zip_vals)
```

```
dict_values(['Carlisle', 'Harrisburg', 'Mechanicsburg'])
```

- These can be used to check if a key or value is (or is not) in the map.
  - Useful for preventing `KeyErorr` errors.

```
zip_keys = zips.keys()
present = 17013 in zip_keys
print(present)
absent = 17013 not in zip_keys
print(absent)
```

```
True
False
```

```
zip = 90001
if zip in zips.keys():
    print(zips[zip])
else:
    print("unknown")
```

```
unknown
```

DICTIONARIES OF COUNTERS

- Dictionaries provide a convenient means to maintain a collection of counters.
  - The key is the category for the counts (e.g. by state)
  - The value is the count (e.g. number of sales).

```
sales = ['PA', 'WA', 'FL', 'PA', 'NY', 'PA', 'VA', 'NY', 'PA']

sales_by_state = dict()

for state in sales:
    if state not in sales_by_state:
        sales_by_state[state] = 1
    else:
        sales_by_state[state] += 1

print(sales_by_state)
```

First time a state is found, use state as a key and map it to the value 1.

Each additional time a state is found, add one to the value to which the state maps

```
{'PA': 4, 'WA': 1, 'FL': 1, 'NY': 2, 'VA': 1}
```

ITERATING OVER DICTIONARIES

- Iteration over the keys or values of a dictionary allows application of the *map*, *filter* and *reduce* patterns to its elements.
  - The examples below *map* the `print` function to the elements.

```
for leader in leaders.values():
    print(leader)
```

Iterate over each *value* in the `leaders` dictionary.

Use the *value* to map, filter or reduce.

```
Meta
Jinping
Merkel
Trump
```

```
for zip_key in zips.keys():
    zip_value = zips[zip_key]
    print(str(zip_key) + ' : ' + zip_value)
```

Iterate over each *key* in the `zips` dictionary.

Get the *value* for the current *key*.

Use the *key and/or the value* to map, filter or reduce

```
17013 : Carlisle
17101 : Harrisburg
17055 : Mechanicsburg
```

REVERSE LOOKUPS

- The filter pattern can be applied to do a ***reverse lookup*** – i.e given a value, find a key that maps to that value.

```
def find_country(leaders, leader):
    for country in leaders.keys():
        if leaders[country] == leader:
            return country

    return None

print(find_country(leaders, 'Merkel'))
print(find_country(leaders, 'Portugal'))
```

Iterate over each *key* (`country`) in the `leaders` dictionary.

Filter the elements to find the *key* (`country`) that maps to the desired *value* (`leader`).

Checked all of the *keys* (`countries`) and didn't find the desired *value* (`leader`).

```
Germany
None
```

THE BUILT-IN SORTED FUNCTION

- The ***sorted*** function will sort a sequence into numeric or lexicographical order.
  - The sorted function is fruitful and does not change the order of the original sequence.

Without Sorting

```
for leader in leaders.values():
    print(leader)
```

```
Meta
Jinping
Merkel
Trump
```

With Sorting

```
sorted_leaders = sorted(leaders.values())
for leader in sorted_leaders:
    print(leader)
```

```
Jinping
Merkel
Meta
Trump
```

OBJECTS AS DICTIONARY VALUES

- When a *key needs to map to more than one value*, the key can be mapped to an object (e.g. a `List` or even another `dict`).

The key `Carlisle` maps to the `List` object `[17013, 17055]`

```
city_zips = dict()

city_zips['Carlisle'] = [17013, 17015]
city_zips['Mechanicsburg'] = [17055, 17050]
city_zips['Harrisburg'] = [17101, 17102, 17103, 17014]

print(city_zips['Carlisle'])
```

```
[17013, 17015]
```

- An object in the dictionary can be manipulated using its methods.

The `List` object mapped to by the key `Harrisburg` is modified using the `List` methods `append` and `extend`.

```
city_zips['Harrisburg'].append(17105)
city_zips['Harrisburg'].extend([17106, 17107, 17018, 17109, 17110])
print(city_zips['Harrisburg'])
```

```
[17101, 17102, 17103, 17014, 17105, 17106, 17107, 17018, 17109, 17110]
```

DEBUGGING/TESTING STRATEGIES

- **Scaffold the Code:**
  - **Print values**
  - **Print examples, summaries or types**
    - Total number of lines, first item, last item, etc...
    - Type of values, lists, keys, etc…
  - **Write self checks**
    - Guardians as sanity checks (positive, not zero, etc)
    - Automated tests to check boundary cases and specific values
  - **Format the output**
    - Use a tabular format
    - Use indentation (\t) to indicate input inside functions
    - Use dividers (*******) to breakup output and make it easier to read
- **Scale down the input**
  - Create dummy data files that have the same format as the full data set.
  - Create smaller data files with just a few lines from the full data set.

EXAM #3

- Wednesday 12/11
- 50 minutes / written (pencil & eraser)
- Closed book / Closed Notes
  - Allowed 1 sheet (single sided) of hand written notes.
  - Documentation for modules/classes will be provided as needed.
- Focus is on material from Class25 through Class35-37 HW & Labs 08-11
- Expect:
  - Vocabulary, Definitions, Ideas, Concepts: Multiple choice, T/F, matching, short answer, etc…
  - Code: Rearrange statements, predict output, state/stack diagrams, write/modify short bits of code.
- Study:
  - Slides / Example Notebooks - Boldface, Italics, Highlighted, Terms Used Repeatedly
  - Homework/Lab Notebooks & Solutions - Practice!
  - Textbook as needed