

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.

```
zips = dict()
zips[17013] = 'Carlisle'
zips[17101] = 'Harrisburg'
zips[17055] = 'Mechanicsburg'

print(zips)
(17013: 'Carlisle', 17101: 'Harrisburg', 17055: 'Mechanicsburg')

leaders = {}

leaders['Albania'] = 'Meta'
leaders['China'] = 'Jinping'
leaders['Germany'] = 'Merkel'
leaders['United States'] = 'Trump'

print(leaders)
{'Albania': 'Meta', 'China': 'Jinping', 'Germany': 'Merkel', 'United States': 'Trump'}
```

Create Dictionary

Add Mappings

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
<ipython-input-112-310a0376e3a0> in <module>
----> 1 print(zips[90201])
```

KeyError: 90201

Traceback (most recent call last)

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'])
```

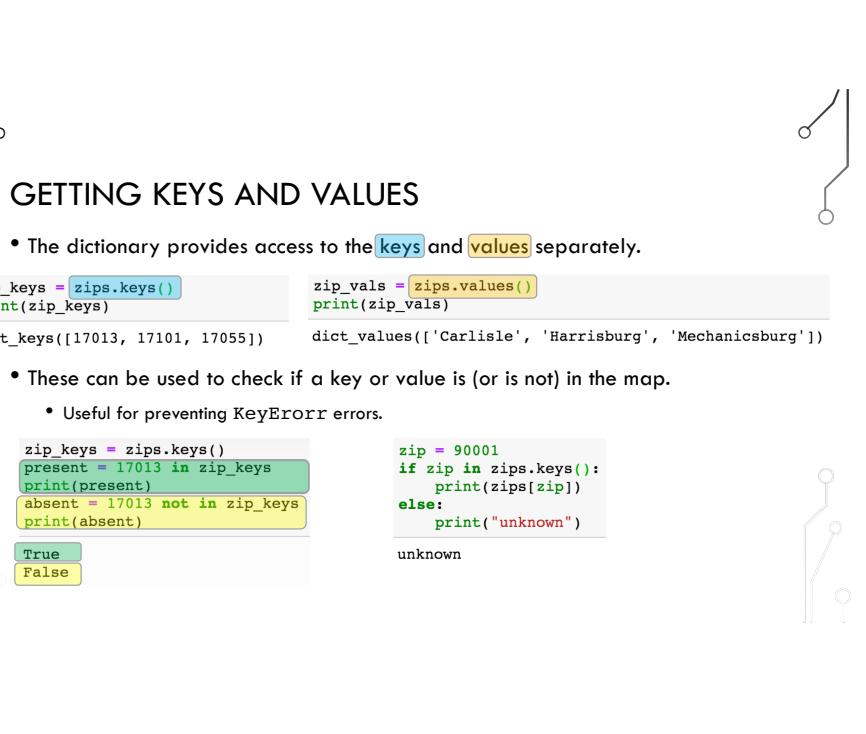
- These can be used to check if a key or value is (or is not) in the map.

- Useful for preventing `KeyError` errors.

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

```
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)
```

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

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

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.  
  
Meta  
Jinping  
Merkel  
Trump
```

Use the value to map, filter or reduce.

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

Get the value for the current key.

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

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
```

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

```
return None
```

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

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

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).

```
city_zips = dict()
```

```
city_zips['Carlisle'] = [17013, 17015]
city_zips['Mechanicsburg'] = [17055, 17050]
city_zips['Harrisburg'] = [17101, 17102, 17103, 17104]
```

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

```
[17013, 17015]
```

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

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

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

```
[17101, 17102, 17103, 17104, 17105, 17106, 17107, 17108, 17109, 17110]
```

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