

dictionaries.

Week 8 | Lecture 2 (8.2)

if nothing else, write `#cleancode`

This Week's Content

- **Lecture 8.1**
 - tuples and sets
 - Reading: 11
- **Lecture 8.2**
 - dictionaries
 - Reading: 11
- **Lecture 8.3**
 - Review for Midterm 2 #jeopardy

Dictionaries

- A dictionary (type `dict`) is an unordered data structure similar to how `sets` are unordered.
- Dictionaries contain references to objects as **key: value** pairs.
- Each key in the dictionary is associated with a value.
- Dictionaries are mutable, entries can be added, modified or removed.

key: **value**

"name": "Pam"

1: [1, 9, 4]

2.35: { 'bee', 'ant' }

(3, 6, 'hi'): False

True: [4, 'tree']

None: 'car'

Dictionaries

Set { 'ford', 'tesla', 'BMW' }

- The general syntax of dict data type is as follows:

```
{key1: val1, key2: val2, ..., keyN: valN}
```

```
{ 'name' : 'Pam', 1: [1, 9, 4], None: 'car' }
```

- Dictionaries are created using curly braces { } around **key: value** pairs of literals and/or variables.

Dictionaries

■ Keys:

- Must be **immutable**.
- int, float, str, tuple, and NoneType, bool. (No lists or sets)

■ Values:

- Can be anything.
- int, float, str, tuple, and NoneType, bool, dict, list, and set.
- Many other 3rd party datasets.

key: **value**

"name": "Pam"

1: [1, 9, 4]

2.35: { 'bee', 'ant' }

(3, 6, 'hi'): False

True: [4, 'tree']

None: 'car'

Dictionaries

- Let's create some dictionaries.

**Open your
notebook**

Click Link:

1. Dictionaries

Dictionary Operations

- Given: `grades = { 'Tina' : 'A+' }`
- **Indexing Operation:**
 - Retrieves the value associated with a key.
 - We index with keys not position like lists & tuples.
- **Add/Modify Entry :**
 - Adds an entry if the entry does not exist, otherwise it modifies the existing entry.
- **Delete Operation:**
 - Removes the key and it's value from a dictionary.
- **In Operator:**
 - Tests for existence of key in the dictionary (it does not check the values).

Dictionaries are **mutable** so entries can be added, modified, and removed.

```
>>> grades[ 'Tina' ]  
'A+'
```

```
>>> grades[ 'John' ] = 'B+'  
>>> grades  
{ 'Tina' : 'A+' , 'John' : 'B+' }
```

```
>>> del grades[ 'Tina' ]  
>>> grades  
{ 'John' : 'B+' }
```

```
>>> if 'John' in grades:  
    print(grades)  
{ 'John' : 'B+' }
```

Dictionary Operations

- Given: `grades = { 'Tina' : 'A+' }`
- **Indexing Operation:**
 - Retrieves the value associated with a key.
 - We index with keys not position like lists & tuples.
- **Add/Modify Entry :**
 - Adds an entry if the entry does not exist, otherwise it modifies the existing entry.
- **Delete Operation:**
 - Removes the key and it's value from a dictionary.
- **In Operator:**
 - Tests for existence of key in the dictionary (it does not check the values).

**Open your
notebook**

Click Link:
**2. Dictionaries
Operations**

Dictionary Methods

- Dictionaries are objects and just like some of the other object we have seen, there are associated methods that are only valid for `dict` types.
- `dict.clear()`
- `dict.keys()`
- `dict.values()`
- `dict.items()`
- `dict.pop(args)`
- `dict.get(args)`
- `dict.update(args)`

**Open your
notebook**

Click Link:
**3. Dictionaries
Methods**

Breakout Session 1

- Complete the exercises in the notebook.

**Open your
notebook**

Click Link:

4. Breakout Session 1

Iterating

- Introducing out 6th iterable!
- A for loop can be used to iterate over a dictionary, with the loop variable being set to the key of an entry in each iteration.
- The **ordering** in which the keys are iterated over is not necessarily the order in which the elements were inserted into the dictionary.

```
friends = {"Bob": 32, "Jane": 42}
```

Keys

```
>>> for key in friends:
        print(key)
```

or friends.keys()

Defaults to List-like object of keys.

"Bob"
"Jane"

Values

```
>>> for value in friends.values():
        print(value)
```

List-like object of values.

32
42

Keys and Values

```
>>> for item in friends.items():
        print(item)
```

List-like object of key-value tuples.

("Bob", 32)
("Jane", 42)

Iterating

- A for loop can be used to iterate over a dictionary, with the loop variable being set to the key of an entry in each iteration.
- The ordering in which the keys are iterated over is not necessarily the order in which the elements were inserted into the dictionary.

**Open your
notebook**

Click Link:
5. Iterating

Breakout Session 2

- **#feelinthebern** with Bernie Sanders.



**Open your
notebook**

Click Link:

6. Breakout Session 2

Inverting Dictionaries

- Reversing a dictionary is not the same as reversing a list; it entails inverting or switching the dictionary's key and value parts.

```
eng2spa = {"two": "dos", "one": "uno"}
```

```
spa2eng = {"dos": "two", "uno": "one"}
```

**Open your
notebook**

Click Link:
**7. Inverting
Dictionaries**

Dictionaries **as** Data Structures

- Nested dictionaries also serve as a simple but powerful data structure.
- A data structure is a logical and coherent organization of data.
- Actually, container objects like lists and dictionaries are already a form of a data structure.
- But, nesting such containers provides a programmer with much more flexibility in the way that the data can be organized.

**Open your
notebook**

Click Link:

**8. Dictionaries as
Data Structures**

Lecture Recap

Practice

- A container of `key:value` pairs.
- Accessing an element via its key.
- Dictionary methods.
- Iterating over dictionaries.
- Testing membership: `in`
- Dictionaries as data structures.
- See Chapter 11 of the Gries textbook.

dictionaries.

Week 8 | Lecture 2 (8.2)

if nothing else, write `#cleancode`