Data Structures

CSCI 145 – Fall 2025

October 15, 2025

Outline

- **Tuples**
- Dictionaries
  - Basics
  - Functions + Methods
  - Loops
- Sets
- Additional Examples and Exercises

Tuples

- Like a **list**, but immutable
- Defined using parentheses, e.g.:
  - point3d = (10, 20, 30)
  - burgundy = (102, 9, 51)
  - Weird syntax for a tuple with one element: numbers = (1,)
- Indexed like a list or a string
- **You can’t update a tuple!**

Tuples Use Case: Return Multiple Values

Returning a Tuple

```
def area_and_circumference(radius):
    area = math.pi * radius ** 2
    circ = 2 * math.pi * radius
    return (area, circ)
```

You can leave out the parentheses:

```
def area_and_circumference(radius):
    area = math.pi * radius ** 2
    circ = 2 * math.pi * radius
    return area, circ
```

Calling the Function

```
small_area, small_circ = area_and_circumference(1)
print(small_area)
print(small_circ)
```

Output

```
3.14
6.28
```

Outline

- Tuples
- **Dictionaries**
  - **Basics**
  - Functions + Methods
  - Loops
- Sets
- Additional Examples and Exercises

Dictionaries

Dictionaries are a data structure that holds **keys** and **values**

my_dict:

keys	Moon	Mars	Jupiter
values	144	47	10

In Python:

```
my_dict = {
    "Moon": 144,
    "Mars": 47,
    "Jupiter": 10
}
```

Look Up a Key in a Dictionary

You can use [] to access the value of a key in your dictionary

my_dict:

keys	Moon	Mars	Jupiter
values	144	47	10

Code

```
my_dict = {"Moon": 144, "Mars": 47,
           "Jupiter": 10}
print(my_dict["Mars"])
print(my_dict["Jupiter"])
```

Output

```
47
10
```

Check if Something is in a Dictionary

You can use the **in** keyword see if a key is in your dictionary

my_dict:

keys	Moon	Mars	Jupiter
values	144	47	10

Code

```
my_dict = {"Moon": 144, "Mars": 47,
           "Jupiter": 10}
print("Venus" in my_dict)
print("Moon" in my_dict)
```

Output

```
False
True
```

Change a Key's Value in a Dictionary

Use [] with an **existing key** to change the value of a key in a dictionary

my_dict (before):

keys	Moon	Mars	Jupiter
values	144	47	10

my_dict (after):

keys	Moon	Mars	Jupiter
values	144	48	10

Code

```
my_dict = {"Moon": 144, "Mars": 47,
           "Jupiter": 10}
my_dict["Mars"] = 48
print(my_dict)
```

Output

```
{"Moon": 144, "Mars": 48, "Jupiter": 10}
```

Add to a Dictionary

Use [] with a **new key** to add a key-value pair to the dictionary

my_dict (before):

keys	Moon	Mars	Jupiter
values	144	47	10

my_dict (after):

keys	Moon	Mars	Jupiter	Venus
values	144	47	10	30

Code

```
my_dict = {"Moon": 144, "Mars": 47,
           "Jupiter": 10}
my_dict[”Venus"] = 30
print(my_dict)
```

Output

```
{"Moon": 144, "Mars": 47, "Jupiter": 10, ”Saturn": 30}
```

Outline

- Tuples
- **Dictionaries**
  - Basics
  - **Functions + Methods**
  - Loops
- Sets
- Additional Examples and Exercises

Length of a Dictionary

You can get the length of a dictionary using the **len** function

my_dict:

keys	Moon	Mars	Jupiter
values	144	47	10

Code

```
my_dict = {"Moon": 144, "Mars": 47,
           "Jupiter": 10}
print(len(my_dict))
```

Output

```
3
```

Get

You can use the **.get()** method for a lookup with a default value

my_dict:

keys	Moon	Mars	Jupiter
values	144	47	10

Code

```
my_dict = {"Moon": 144, "Mars": 47,
           "Jupiter": 10}
print(my_dict.get("Mars", 0))
print(my_dict.get(”Neptune", 0))
```

Output

```
47
0
```

Dictionary Keys

You can get the keys of a dictionary by using the **.keys()** method

my_dict:

keys	Moon	Mars	Jupiter
values	144	47	10

Code

```
my_dict = {"Moon": 144, "Mars": 47,
           "Jupiter": 10}
print(my_dict.keys())
```

Important Note:

This looks like a list, but you can't index it. You can iterate through it with a loop.

Output

```
dict_keys(['Moon', 'Mars', 'Jupiter'])
```

Dictionary Values

You can get the values of a dictionary by using the **.values()** method

my_dict:

keys	Moon	Mars	Jupiter
values	144	47	10

Code

```
my_dict = {"Moon": 144, "Mars": 47,
           "Jupiter": 10}
print(my_dict.values())
```

Important Note:

This looks like a list, but you can't index it. You can iterate through it with a loop.

Output

```
dict_values([144, 47, 10])
```

Dictionary Key-Value Pairs

You can get the key-value pairs from a dictionary by using the **.items()** method

my_dict:

keys	Moon	Mars	Jupiter
values	144	47	10

Code

```
my_dict = {"Moon": 144, "Mars": 47,
           "Jupiter": 10}
print(my_dict.items())
```

Important Note:

This looks like a list, but you can't index it. You can iterate through it with a loop. The pairs are tuples.

Output

```
dict_items([('Moon', 144), ('Mars', 47), ('Jupiter', 10)])
```

Outline

- Tuples
- **Dictionaries**
  - Basics
  - Functions + Methods
  - **Loops**
- Sets
- Additional Examples and Exercises

Loop Through a Dictionary

```
for key, value in my_dict.items():
    print(key)
    print(value)
    my_dict[key] = value + 1
print(my_dict)
```

Code

```
for key, value in my_dict.items():
    print(key)
    print(value)
    my_dict[key] = value + 1
print(my_dict)
```

Description

Variables

my_dict

keys	Moon	Mars	Jupiter
values	144	47	10

key

value

Output

Code

```
for key, value in my_dict.items():
    print(key)
    print(value)
    my_dict[key] = value + 1
print(my_dict)
```

Description

key, value start off as one key-value pair, the first key-value pair in my_dict.items()

Variables

my_dict

keys	Moon	Mars	Jupiter
values	144	47	10

key: "Moon"

value: 144

Output

Code

```
for key, value in my_dict.items():
    print(key)
    print(value)
    my_dict[key] = value + 1
print(my_dict)
```

Description

print the current key variable

Variables

my_dict

keys	Moon	Mars	Jupiter
values	144	47	10

key: "Moon"

value: 144

Output

Moon

Code

```
for key, value in my_dict.items():
    print(key)
    print(value)
    my_dict[key] = value + 1
print(my_dict)
```

Description

print the current value variable

Variables

my_dict

keys	Moon	Mars	Jupiter
values	144	47	10

key "Moon"

value 144

Output

```
Moon
144
```

Code

```
for key, value in my_dict.items():
    print(key)
    print(value)
    my_dict[key] = value + 1
print(my_dict)
```

Description

Increment the value associated with key in my_dict by 1

Variables

my_dict

keys	Moon	Mars	Jupiter
values	~~144~~ 145	47	10

key: "Moon"

value: 144

Output

```
Moon
144
```

Code

```
for key, value in my_dict.items():
    print(key)
    print(value)
    my_dict[key] = value + 1
print(my_dict)
```

Description

Change key and value to the next key-value pair

Variables

my_dict

keys	Moon	Mars	Jupiter
values	145	47	10

key ~~"Moon"~~ "Mars"

value ~~144~~ 47

Output

```
Moon
144
```

Code

```
for key, value in my_dict.items():
    print(key)
    print(value)
    my_dict[key] = value + 1
print(my_dict)
```

Description

print the current key variable

Variables

my_dict

keys	Moon	Mars	Jupiter
values	145	47	10

key: "Mars"

value: 47

Output

```
Moon
144
Mars
```

Code

```
for key, value in my_dict.items():
    print(key)
    print(value)
    my_dict[key] = value + 1
print(my_dict)
```

Description

print the current value variable

Variables

my_dict

keys	Moon	Mars	Jupiter
values	145	47	10

key "Mars"

value 47

Output

```
Moon
144
Mars
47
```

Code

```
for key, value in my_dict.items():
    print(key)
    print(value)
    my_dict[key] = value + 1
print(my_dict)
```

Description

Increment the value associated with key in my_dict by 1

Variables

my_dict

keys	Moon	Mars	Jupiter
values	145	~~47~~ 48	95

key: "Mars"

value: 47

Output

```
Moon
144
Mars
47
```

Code

```
for key, value in my_dict.items():
    print(key)
    print(value)
    my_dict[key] = value + 1
print(my_dict)
```

Description

Change key and value to the next key-value pair

Variables

my_dict

keys	Moon	Mars	Jupiter
values	145	48	10

key ~~"Mars"~~ "Jupiter"

value ~~47~~ 10

Output

```
Moon
144
Mars
47
```

Code

```
for key, value in my_dict.items():
    print(key)
    print(value)
    my_dict[key] = value + 1
print(my_dict)
```

Description

print the current key variable

Variables

my_dict

keys	Moon	Mars	Jupiter
values	145	48	10

key: "Jupiter"

value: 10

Output

```
Moon
144
Mars
47
Jupiter
```

Code

```
for key, value in my_dict.items():
    print(key)
    print(value)
    my_dict[key] = value + 1
print(my_dict)
```

Description

print the current value variable

Variables

my_dict

keys	Moon	Mars	Jupiter
values	145	48	10

key "Jupiter"

value 10

Output

```
Moon
144
Mars
47
Jupiter
10
```

Code

```
for key, value in my_dict.items():
    print(key)
    print(value)
    my_dict[key] = value + 1
print(my_dict)
```

Description

Increment the value associated with key in my_dict by 1

Variables

my_dict

keys	Moon	Mars	Jupiter
values	145	48	~~10~~ 11

key: "Jupiter"

value: 10

Output

```
Moon
144
Mars
47
Jupiter
10
```

Code

```
for key, value in my_dict.items():
    print(key)
    print(value)
    my_dict[key] = value + 1
print(my_dict)
```

Description

We're done looping! Print the my_dict variable's current value

Variables

my_dict

keys	Moon	Mars	Jupiter
values	145	48	11

key: "Jupiter"

value: 10

Output

```
Moon
144
Mars
47
Jupiter
10
{'Moon': 145, 'Mars': 48, 'Jupiter': 11}
```

Exercise: Character Frequency

```
def character_frequency(word):
    """
    Return a dictionary in which the keys are the specific characters in the word,
    and the values are the frequency of every character.
    Parameters: word (string)
    Returns: frequency (dictionary - str/int)
    """
    frequency = {}
    for char in word:
        frequency[char] += 1
    return frequency


f = character_frequency("hello")
print(f)
```

Description

1. What should this program print, given what the docstring says about the function?
2. There is a runtime error in this code! Identify the line of the runtime error.
3. Identify one way to fix the error
   - If you have time, try to find another way to fix the error (HINT: one uses .get(), the other does not!)

Solution: Character Frequency Bugfix

Using .get()

```
def character_frequency(word):
    frequency = {}
    for char in word:
        frequency[char] = frequency.get(char, 0) + 1
    return frequency
```

Without .get()

```
def character_frequency(word):
    frequency = {}
    for char in word:
        if char not in frequency:
            frequency[char] = 0
        frequency[char] += 1
    return frequency
```

Outline

- Tuples
- Dictionaries
  - Basics
  - Functions + Methods
  - Loops
- **Sets**
- Additional Examples and Exercises

Sets

- Sets are an **unordered collection** of distinct items
- Each item may appear only once in a set
- Creating an empty set:
  - s = set()
  - Common error: don’t use {} (that is an empty dictionary)
- Creating a non-empty set:
  - s = {1, 2}

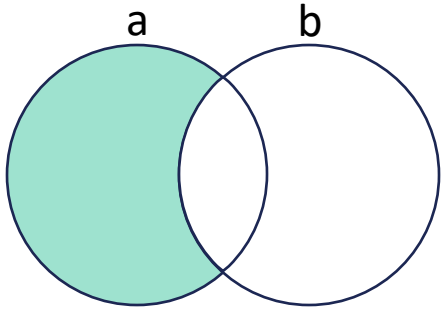
Set Methods

add

```
s = {3}
s.add(1)
print(s) # prints {1, 3}
```

What happens if you add something that's already in the set? **Nothing!**

Difference

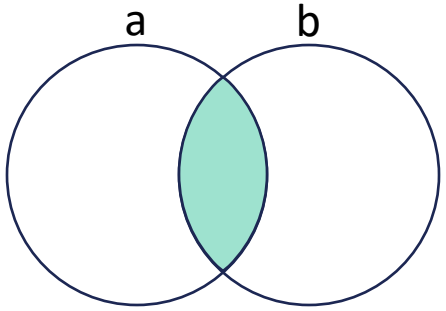


Python syntax:

- a.difference(b)
- a - b

All of the elements from set a that are not in set b

Intersection

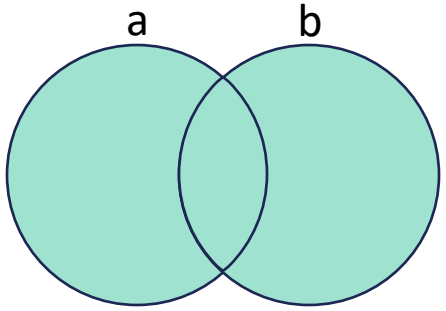


Python syntax:

- a.intersection(b)
- a & b

All of the elements that are in **both** set a and set b

Union



Python syntax:

- a.union(b)
- a | b

All of the elements that are in **either** set a or set b

Check if Something is in a Set

- Use the **in** operator, just like you would for a string, a list, or a dictionary
- One cool thing: this is faster with sets than it is with lists
  - Want to know how this is possible? Take CS 201!

Exercise: Set Operations

We have two sets:

a = {1, 3, 7}

b = {3, 9}

Fill in the blanks in the assignments below

c = a._____(b) → c = {1, 7}

c = a._____(b) → c = {1, 3, 7, 9}

Outline

- Tuples
- Dictionaries
  - Basics
  - Functions + Methods
  - Loops
- Sets
- **Additional Examples and Exercises**