Week 9

Lecture:

- Dictionaries
- Dictionary Operations & Methods

Lecture: Dictionaries, Dictionary operations & methods *

*Adapted from Chapter 12 of *Think Python: How to Think Like a Computer Scientist, Second Edition*

- Strings, lists, and tuples are sequence types - the items in the collection are ordered from left to right and they use integers as indices to access the values they contain.
- **Dictionary** in Python is a collection of data values.
  - They map (associate) a **key** with a **value**.
  - In other languages, they are called *associative arrays*.

Dictionaries - Example

- A dictionary to translate English words to Spanish. In this example, both keys and values will be strings.

To create dictionary- **Method One:**

```
eng2sp = {}
eng2sp["one"] = "uno"
eng2sp["two"] = "dos"
```

- Start with an empty dictionary named eng2sp.
- Then add new **key:value** pairs to the dictionary.

Dictionaries - Printing / len()

- Print the current value of the dictionary:

```
print(eng2sp)
# {'one': 'uno', 'two': 'dos'}
```

- key:value pairs are separated by commas.
- Each pair contains a key and value separated by a colon.

- len() function returns the number of key:value pairs:

```
len(eng2sp)                    # 2
```

Dictionaries

- To create dictionary - **Method Two:**

```
eng2sp = {"one":"uno", "two":"dos", "three":"tres"}
```

- Create list and provide initial list of key:value pairs.
- The order of the pairs does not matter – values are accessed with keys, not with indices. Using a key to look up the corresponding value:

```
print(eng2sp["two"])  # dos
```

Dictionary – del statement

- The del statement removes a key:value pair.
- E.g., dictionary contains fruits and number of each fruit in stock:

```
stock = {"apples": 430, "bananas": 312, "pears": 217}
```

- If all pears are bought - remove the entry from the dictionary:

```
del stock["pears"]
print(stock)
# {'apples': 430, 'bananas': 312}
```

Dictionaries Are Mutable

- *Dictionaries* are like *lists* – they are both **mutable**. To change a value:

```
stock["pears"] = 0
print(stock)
# {'apples': 430, 'bananas': 312, 'pears': 0}
```

- A new shipment of bananas could be handled like this:

```
stock["bananas"] += 200
print(stock)
# {'apples': 430, 'bananas': 512, 'pears': 0}
```

Dictionaries Key / Value Rules

Dictionary Keys:

- Must be unique
- Of *immutable* data type such as Strings, Integers and tuples

Dictionary Values:

- May contain repeating values
- Can be of any type

SELF-CHECK: Question 1&2

Question 1 - What is printed?

```
mydict = {"cat":12, "dog":6, "elephant":23}
print(mydict["dog"])
```

A) True B) 6 C) dog : 6 D) Error

Question 2 - What is printed?

```
mydict = {"cat":12, "dog":6, "elephant":23}
mydict["mouse"] = mydict["cat"] + mydict["dog"]
print(mydict["mouse"])
```

A) 12 B) 0 C) 18 D) Error – no mouse key

Dictionary methods

Method	Parameters	Description
keys	None	Returns a view of the keys in the dictionary
values	None	Returns a view of the values in the dictionary
items	None	Returns a view of the key-value pairs in the dictionary
get	Key	Returns the value associated with key; None otherwise
get	key, alt	Returns the value associated with key; alt otherwise

Dictionary method: keys()

keys() returns a **view** of its underlying keys. Iterating over the view:

```
eng2sp = {"one":"uno", "two":"dos", "three":"tres"}
for k in eng2sp.keys():
    print("Got key", k, "maps to value", eng2sp[k])
```

Output:

Got key one maps to value uno

Got key two maps to value dos

Got key three maps to value tres

Dictionary method: keys()

- We can turn the view into a list by using the list conversion function.

```
ks = list(eng2sp.keys())
print(ks)
#['one', 'two', 'three’]
```

- We can omit the keys method call in the for loop — iterating over a dictionary implicitly iterates over its keys:

```
for k in eng2sp:
    print("Got key", k)
```

Dictionary method: values()

- The values method is similar; it returns a view which can be turned into a list:

```
list(eng2sp.values())
#['uno', 'dos', 'tres']
```

Dictionary method: items()

- Method items() returns a list of tuples, one tuple for each key:value

```
list(eng2sp.items())
#[('one', 'uno'), ('two', 'dos'), ('three', 'tres')]
```

- Tuples are useful for getting both the key and the value while looping:

```
for (k,v) in eng2sp.items():
    print("Got",k,"that maps to",v)
```

Output:

```
Got one that maps to uno
Got two that maps to dos
Got three that maps to tres
```

Dictionary operators: in/not in

- **in** and **not in** can test if a key is in the dictionary:

```
"one" in eng2sp     # True
"six" in eng2sp     # False
"tres" in eng2sp
# False 'in' tests keys, not values
```

- Useful, since looking up a non-existent key in a dictionary causes an error.

Self-Check: Question 3 - What is printed?

```
mydict = {"cat":12, "dog":6, "elephant":23, "bear":20}
keylist = list(mydict.keys())
keylist.sort()
print(keylist[3])
```

A) cat B) dog C) elephant D) bear

Dictionary method: get()

- The get() method accesses the value associated with a key.
- Similar to the [ ] operator. However, get() will NOT cause an error if the key is not present - will return None.

```
stock = {'apples': 430, 'bananas': 312, 'pears': 217}

print(stock.get("apples"))              # 430

print(stock.get("cherries"))            # None
```

- An optional second parameter provides an alternative return value where the key is not present. E.g., as “cherries” is not a key, return 0 (instead of None).

```
print(stock.get("cherries", 0))         # 0
```

Self-Check: Question 4 - What is printed?

```
mydict = {"cat":12, "dog":6, "elephant":23, "bear":20}
answer = mydict.get("cat") + mydict.get("dog")
print(answer)
```

A) 126
B) 18
C) catdog
D) Error, + is not a valid on dictionaries

Self-Check: Questions 5 & 6

- **Question 5** - What is printed – True or False?

```
mydict = {"cat":12, "dog":6, "elephant":23, "bear":20}
print("dog" in mydict)
```

- **Question 6** - What is printed – True or False?

```
mydict = {"cat":12, "dog":6, "elephant":23, "bear":20}
print(23 in mydict)
```

Self-Check: Question 7 - What is printed?

```
total = 0
mydict = {"cat":12, "dog":6, "elephant":23, "bear":20}
for akey in mydict:
    if len(akey) > 3:
        total = total + mydict[akey]
print(total)
```

A) 18 B) 43 C) 0 D) 61

Dictionaries - Aliasing and copying

- As with lists, dictionaries are mutable - be aware of aliasing. Whenever two variables refer to the same object, changes to one affect the other.
- To modify a dictionary and keep a copy of the original, use copy().

```
opposites = {"up": "down", "right": "wrong", "yes": "no"}
alias = opposites
copy = opposites.copy()
```

- alias and opposites refer to the same object. If we modify alias, opposites is also changed. However, if we modify copy, opposites is unchanged.

Python f-string and dictionaries

- We can work with dictionaries in f-strings.

```
user = {'name': 'John Doe', 'occupation': 'gardener'}
print(f"{user['name']} is a {user['occupation']}")
```

- The example evaluates a dictionary in an f-string.

Output:

```
John Doe is a gardener
```

Summary - Python Dictionary

- A collection of data values.
- They map (associate) a **key** with a **value**.
- Dictionary methods:
  - keys() - Returns a view of the keys in the dictionary
  - values() - Returns a view of the values in the dictionary
  - items() - Returns a view of the key-value pairs in the dictionary
  - get() - Returns the value associated with key
- Dictionaries are mutable - Aliasing and copying