Python - Dictionaries and Sets

Dictionary

- An object that stores a collection of data
- Each element in a dictionary has two parts: a key and a value
- You use a key to locate a specific value

Creating a dictionary

```
variable_name = {key1: value1, key2: value2}

first_dict = {“students”: 3000, “systems”: 30}
```

Retrieving a Value

- Elements in a dictionary are unsorted
- To retrieve an element from a dictionary, use the format `dictionary_name[key]` or `get()` method

```
# Using square brackets
dct1 = {1: "welcome", 2: "to", 3: "ALX"}
print(dct1[3]) # ALX

# Using get() method
dct1 = {1: "welcome", 2: "to", 3: "ALX"}
print(dct1.get(3)) # ALX
```

Test for value

- Test whether a key is in a dictionary using `in` the and `not in` operators. Helps prevent `KeyError` exceptions

```
dct1 = {1: "welcome", 2: "to", 3: "ALX"}
if 3 in dct1:
    print(dict1[3])
```

Adding elements

- Dictionaries are mutable objects
- To add a new key-value pair:

```
dictionary[key] = value
```

- If key exists in the dictionary, the value associated with it will be changed

Deleting elements

- The del() keyword removes the element of the specified key name.
- `del dictionary_name[key]`
- If key not in the dictionary, KeyError exception is raised

Getting the number of elements

- You can use the built in “len” function to get the number of elements in a dictionary

Mixing data types in a dictionary

- Keys must be immutable objects, but their associated values can be of any type of object
- A dictionary can include keys of several different immutable types
- Values stored in a dictionary can be of different types

Creating an empty dictionary

- To create an empty dictionary:
- Use {}
- Use built-in function dict()
- Elements can be added to the dictionary as the program executes
- Use a for loop to iterate over a dictionary

```
for key in dictionary_name:
    statement(s)
```

Dictionary methods

- Clear
- Get
- Items
- Keys
- Pop
- Popitem
- values

Clear method

- Deletes all the elements in a dictionary, leaving the dictionary empty.
- `dictionary_name.clear()`

Get method

- Gets value associated with a specified key from a dictionary
- `dictionary_name.get(key, default)`
- Alternative to [] operator and does not raise KeyError exception

Items method

- Returns all of the dictionaries keys and their associated values.
- Returned in a dictionary view where each element is a tuple with contains a key and its associated value

```
dct = {1: "me", 2: "you"}
dct.items()
# dict_items([(1, 'me'), (2, 'you')])
```

Keys method

- Returns all of a dictionary keys in a dictionary view, which is a type of sequence

```
dct = {1: "me", 2: "you"}
dct.keys()
# dict_keys([1, 2])
```

Pop method

- Returns the value associated with a specified key and removes that key value pair from the dictionary
- If key is not found the method returns a default value

```
dictionay_name.pop(key, default)
```

Popitem method

- Returns a randomly selected key-value pair, and removes it from the dictionary
- The key-value pair is returned as a tuple

```
dictionary_name.popitem()

k,v = dictionary_name.popitem() # assigns the returned key and value to individual variables
```

Dictionary comprehension

- Build a new dictionary by applying an expression to each item in the iterable

```
Syntax: {key: value for var in iterable}
dict = {x: x ** 4 for x in range(5)}
print(dict) # {0: 0, 1: 1, 2: 16, 3: 81, 4: 256}
```

Set

- A set is an object that stores a collection of data in same way as mathematical set
- All items must be unique
- Set is unordered - elements in a set are not stored in any particular order
- Elements can be of different data types

Creating a set using curly braces

```
# set of integers
set1 = {1, 2, 3, 4, 5}

# set of strings
set2 = {"joe", "noah", "amos"}

# set of mixed data type
set3 = {"joe", "noah", (1, ), 2, 3}
```

Creating a set using the set() method

```
# set of integers
set1 = set([1, 2, 3, 4, 5])

# set of strings
set2 = set(["joe", "noah", "amos"])

# set of mixed data type
set3 = set(["joe", "noah", (1, ), 2, 3])
```

Getting the number of elements

- As with lists, tuples, and dictionaries, you can use the “len” function to get the number of elements in a set

Adding and removing elements

- Sets are mutable objects, so you can add items to them and remove items from them.
- You use the add method to add an element to a set

```
set_name.add(element)
```

- To add multiple items, we use update() method
- **`Remove`** and **`discard`** methods remove the specified item from the set

```
set2.remove("joe")
set2.discard("noah")
```

- **`Clear`** method clears all the elements of the set

Iterate over a set

- You can use a **`for loop`** to iterate over all the elements in a set

```
for iter_var in set:
    statement(s)
```

Finding the Union of Sets

- The union of two sets is a set that contains all the elements of both sets.
- In Python, you can call the union method to get the union of two sets

```
set1.union(set2)
```

Finding the Intersection of Sets

- The intersection of two sets is a set that contains only elements that are found in both sets.
- In Python, you can call the intersection method to get the intersection of two sets

```
set1.intersection(set2)
```

Finding the Difference of Sets

- The difference of set1 and set2 are the elements that appear in set1 but do not appear in set2

```
set1.difference(set2)
```

Finding the Symmetric Difference of Sets

- The symmetric difference of two sets is a set that contains the elements that are not shared by the sets.
- Elements that are in one set but not in both

```
set1.symmetric_difference(set2)
```

Finding Subsets and Supersets

- A set containing all of the elements of the other set

```
>>> set1 = set([1, 2, 3, 4])
>>> set2 = set([2, 3])
>>> set2.issubset(set1)
True
>>> set1.issuperset(set2)
True
```

See you at the next session!