Tuples, Dictionaries, and Sets

APRIL 6

Tuples in Python

- Tuples are closely related to lists in Python
- The one obvious syntactic difference is that round brackets are used to define tuples rather than square brackets.
- **Example:** `T = (0, -90, "hello")`
- Elements of a tuple can be accessed just like elements of a list
- **Example:** `T[0]`
- **Example:** `T[0:2]`

Main difference

- Tuples are *immutable* versions of lists.
- Tuples do not come with any of the methods that modify lists in place: insert, remove, pop, etc.

- So why use tuples??

1. **Faster than lists**. Use if you know that you only have to scan, not modify.
2. Can be used (instead of lists) in situations that require immutable objects (e.g., dictionaries).

Easy to go from lists to tuples (and back)

- **Example:**

```
>>> L = [3, 4, "hello"]
>>> T = tuple(L)
>>> T
(3, 4, 'hello')
>>> LL = list(T)
>>> LL
[3, 4, 'hello']
>>>
```

Dictionaries

- *Dictionary* is a Python data structure that consists of key-value pairs.
- **Example**:

  `D = {"to": 10, "be": 20, "it": 31, "go": 20}`
- Here the *keys* are "to", "be", "it", and "go"
- The *values* are 10, 20, 31, and 20
- The dictionary D is a function that associates a value to each key.
- Keys in a dictionary have to be distinct, i.e., no duplicate keys.

Accessing items in a dictionary

- **Example**:

  D = {"to": 10, "be": 20, "it": 31, "go": 20}

  D["to"] evalutes to 10

- Typical way of accessing a dictionary is by using a key inside square brackets as a way to get to the associated value.
- A dictionary cannot be accessed using the values – only via the keys.

Modifying a dictionary

- **Example:**

```
>>> D = {"to": 10, "be": 20, "it": 31, "go": 20}
>>> D["to"] = 25
>>> D
{'go': 20, 'to': 25, 'it': 31, 'be': 20}
>>> D["hello"] = 100
>>> D
{'go': 20, 'to': 25, 'hello': 100, 'it': 31, 'be': 20}
```

- The value associated with a key can be modified by an assignment.
- A new key-value pair can also be added to the dictionary by an assignment.

Deleting items from a dictionary

- **Example**:

```
>>> D = {"to": 10, "be": 20, "it": 31, "go": 20}
>>> del D["be"]
>>> D
{'go': 20, 'to': 10, 'it': 31}
```

- To clear all values in a dictionary, use D.clear(). After a dictionary is "cleared" its value is {}

A few dictionary functions

1. `D.keys()` returns a list with all the keys in `D`
2. `D.values()` returns a list with all the values in `D`
3. `D.items()` returns a list of key-value pairs (as tuples)
4. `key in D`, `key not in D` evaluate to boolean values depending on whether key is in `D`
5. `D.pop(key)` removes the key-value pair corresponding to `key` and returns the value
6. `D.popitem()` removes and returns an arbitrary key-value pair from `D`
7. `D.update(uD)` updates `D` using the key-value pairs in `uD`

Dictionaries: why use them?

- Search in lists is slow – takes time proportional to the size of the list, in the worst case.
- Search in dictionaries is extremely fast.
- Use dictionaries in applications where searching by key values is done repeatedly.

- **Problem**: Process a text file and create dictionaries of 1-letter, 2-letter, and 3-letter words. The keys are the words and the frequencies are the corresponding values.