Dictionaries, Tuples, and Files

Advanced data structures in `Python`

Alwin Tareen

Dictionaries have no Sequential Structure

- Data which is placed in a list will, by default, have an implicit sequential structure imposed on it. This means that the data has both content, and a positional value.
- There are many instances in which the positional nature of a list is unsuitable for the data.
- In other words, we can have collections of data in which there is no sequential relationship between the elements.
- `Python` has a special data structure which allows you to index your data elements in a much less restrictive manner. It is called a **dictionary**.

The Definition of a Dictionary

- A dictionary can be described as a kind of mapping between a lookup element called a **key**, and the data which corresponds to it, called a **value**.
- A dictionary consists of a series of these key-value pairs, and they are not sequential in nature.
- In order to obtain a value from a dictionary, you must specify its associated `key`.
- One restriction on a dictionary is that a key must be of an immutable data type. Values, however, can be of any data type.
- `Python`'s data type name for a dictionary is: `dict`.

Declaring a Dictionary

- We can declare an empty dictionary by using a pair of curly braces:

```
sports = {}
```

- Alternatively, we can specify the key-value pairs of the dictionary when we first create it.
- Each of the key-value pairs are separated by a comma. Within each pair, the key is separated from the value by a colon.

```
fruit = {"pear":2, "banana":3, "orange":5}
```

Dictionary Lookups

- Note that it doesn't matter how the particular key-value pairs are ordered. The values in a dictionary are accessed with keys, not numerical indexes, so the ordering should be of no concern.
- In order to look up a value in a dictionary, use square bracket notation with its associated key in the following manner:

```
snack = fruit["orange"]
```

- The key `"orange"` yields the value `5`.

Dictionary Operations

`len()`

The `len()` function returns the number of key-value pairs that are present in the dictionary.

```
total = len(fruit)
```

`in`

This indicates whether an item appears as a key in a dictionary.

```
result = "banana" in fruit
outcome = "kiwi" in fruit
```

Dictionary Operations

`del`

The `del` statement removes an entire key-value pair from the dictionary. You only need to specify the key with square bracket notation in the following manner:

```
del fruit["pear"]
```

Adding a key-value pair to the dictionary

This can be accomplished by using square bracket notation with the key, and placing the value on the right hand side of the assignment statement.

```
fruit["mango"] = 8
```

Dictionary Operations

`sorted(x)`

This takes a dictionary as an argument, and returns a list consisting of the keys that appear in the dictionary, in sorted order.

```
ranked = sorted(fruit)
print(ranked) # ["banana", "orange", "pear"]
```

Dictionary Methods

The syntax of a method call is as follows:

```
dictionaryname.methodname(arguments)
```

keys()

This method returns a list consisting of all the keys that appear in the dictionary.

```
produce = fruit.keys()
print(produce) # ["pear", "banana", "orange"]
```

copy()

This method produces an exact reproduction of the dictionary.

```
citrus = fruit.copy()
```

Dictionary Methods

`values()`

This method returns a list consisting of all the values that appear in the dictionary.

```
quantity = fruit.values()
print(quantity) # [2, 3, 5]
```

`items()`

This method returns a list consisting of all the key-value pairs, with each pair contained in a tuple.

```
inventory = fruit.items()
print(inventory)
# [("pear", 2), ("banana", 3), ("orange", 5)]
```

Dictionary Methods

`get(x, y)`

- This method take two arguments: a data item `x` which corresponds to the key, and a default value `y`.
- The `get()` method will check to see if the argument `x` appears as a key in the dictionary.
- If so, then it will return the dictionary value associated with that key. If not, then it will return the default value `y`.
- Usually, programmers assign `0` to `y`.

  `fruit.get("orange", 0)` $\rightarrow 5$

  `fruit.get("watermelon", 0)` $\rightarrow 0$
- Using `get()` is a much safer method of retrieval, because you avoid `KeyErrors`.

Looping Through a Dictionary

- It is possible to use a dictionary data structure with a `for` loop.
- The `for` loop's iteration variable progresses through the keys of the dictionary.
- In the following example, the iteration variable is called `key`.

```
for key in fruit:
    print(key)
```

Looping Through a Dictionary

- If you wanted to access the values in the dictionary, you could perform a lookup using the key with square bracket notation:

```
for key in fruit:
    print(fruit[key])
```

- An alternate way to perform this action is to loop across the list that is generated by the `values()` method.

```
for value in fruit.values():
    print(value)
```

Tuples

The Definition of a Tuple

- A tuple is a sequence of values, much like a list.
- The items stored in a tuple can be of any data type, and they are indexed by integers.
- Unlike a list, once a tuple has been initialized, it cannot be changed. Tuples are immutable.
- You can use the `len()` function on a tuple, as well as the `count()` and `index()` methods.
- However, you cannot use any method that attempts to alter the tuple, like `sort()` or `reverse()`.

The Definition of a Tuple

- There are some situations where an immutable data sequence such as a tuple is desirable.
- For instance, tuples can serve as keys in dictionaries whereas lists cannot.
- Also, tuples have faster element access when compared to lists.
- Tuples are enclosed in parentheses, as in the following:

```
drinks = ("tea", "coffee", "juice")
```

- Note that to create a tuple with a single element, you must include the final comma, or `Python` thinks that it's just a string:

```
soda = ("cola",)
```

Tuple Operations

- Once created, the elements in a tuple can be accessed using square bracket notation:

  `drinks[0]` → `"tea"`

- Slicing can also be used:

  `drinks[:2]` → `("tea", "coffee")`

- Remember that tuples cannot be modified. The following is an error:

  `drinks[0] = "milk"`

- We can use the `len()` function with a tuple:

  `len(drinks)` → `3`

- A `for` loop can be used to iterate over the elements in a tuple:

```
for item in drinks:
    print(item)
```

Comparing Tuples

- The comparison operators($<$, $>$, etc.) work with tuples.
- `Python` starts by comparing the first element from each sequence.
- If those elements are equal, then it goes on to the next element, and so on, until it finds elements that differ. Then, it makes a determination based on those.

```
(8, 5, 17, 500) < (8, 5, 23, 19) # True
```

Tricks with Tuples

Multiple assignment

- In `Python`, we can have a tuple on the left hand side of the assignment statement. This means that you can assign more than one variable at a time.

```
results = [98, 17]
(total, count) = results
```

Tuples as dictionary keys

- Since tuples are immutable, we can use them as keys in a dictionary.

```
res = {("Smith", "Alice"):92, ("Jones", "Bob"):89}
```

Using a Sorting Function with Tuples

- Tuples can't use the `sort()` function, since `sort()`'s behaviour causes the data structure to be altered. Recall that we can't alter a tuple.
- However, tuples can use the `sorted()` function, because `sorted()` returns a new list sequence.

```
drinks = ("tea", "coffee", "juice")
beverages = sorted(drinks)
print(beverages)
```

File Handling

Using Files in Python

- In this section, we explore ways of storing data in a persistent manner.
- In other words, we can store data in files, which remain in secondary memory even when the computer's power is turned off.

Opening Files

The `open()` function

- Before we can perform any action on a file, such as reading or writing to it, we must first open the file.
- The file that you want to open must be in the same directory as your `Python` program.
- If we have a file called `mailbox.txt`, then the `open()` function would work as follows:

```
fhand = open("mailbox.txt")
```

- Note that the file name must be enclosed by quotation marks.
- `fhand` represents the file **handle**, not the file itself. It acts as a data pipe, from which the file's data can be read.

A Text File's Format

- A text file can be regarded as a sequence of lines.
- Each line in the file is terminated with the newline character: `\n`
- Essentially, the newline character separates the text in the file into individual lines.

Reading Files

- We can use the file handle `fhand` along with a `for` loop to read the lines in a text file.

```
fhand = open("mailbox.txt")
for line in fhand:
    print(line)
```

- Note that the output from this program looks awkward, because there is an extra blank line in between every line of text.

Fixing the Extra Blank Lines

- Recall that every line of text in a file ends in a newline, and then the `print()` function adds another newline, resulting in a double spacing effect.
- A simple and common way to overcome this is to use the `rstrip()` method. It removes all the whitespace characters from the right side of a string.

```
fhand = open("mailbox.txt")
for line in fhand:
    line = line.rstrip()
    print(line)
```

Searching Through a File

- It is a common pattern to read through a file, ignore most of the lines, and to only process those lines of text that meet a particular condition.
- For example, consider the case where we want to read the file `mailbox.txt`, and only display the lines that begin with `"From:"`
- We could use the method `startswith()` as follows:

```
fhand = open("mailbox.txt")
for line in fhand:
    line = line.rstrip()
    if line.startswith("From:"):
        print(line)
```

Writing to a File

- If you intend to write to a file, then you must use the `open()` function with `"w"` as the second parameter:

```
fout = open("datadump.txt", "w")
```

- If the file `datadump.txt` does not exist in the current directory, then it will be created automatically.
- However, if `datadump.txt` is already present, then it will automatically be erased and written over. Be careful!

Writing to a File

The `write()` method

- The `write()` method places string data into a file.
- It returns the number of characters that have been written.

```
fout = open("datadump.txt", "w")
fout.write("Here is the lunch menu.\n")
```

- Note that we must indicate the end of the line by inserting the newline character, `\n`

```
food = "pizza, cheeseburgers, soda\n"
fout.write(food)
```

Closing a File

The `close()` method

- When we have finished writing to the file, we must call the `close()` method on the file handle, as follows:

```
fout = open("datadump.txt", "w")
fout.write("Here is the lunch menu.\n")
food = "pizza, cheeseburgers, soda\n"
fout.write(food)
fout.close()
```

Dictionaries, Tuples and Files: End of Notes