Chapter 6: Associated Arrays and Hashing

CSCI-UA 0002 – Introduction to Computer Programming
Mr. Joel Kemp
Oh no!

- We misread the clipboard with the scores!

<table>
<thead>
<tr>
<th>Name</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Johnny</td>
<td>8.65</td>
</tr>
<tr>
<td>Juan</td>
<td>9.12</td>
</tr>
<tr>
<td>Joseph</td>
<td>8.45</td>
</tr>
<tr>
<td>Stacey</td>
<td>7.81</td>
</tr>
<tr>
<td>Nideen</td>
<td>8.05</td>
</tr>
<tr>
<td>Zack</td>
<td>7.21</td>
</tr>
<tr>
<td>Aaron</td>
<td>8.31</td>
</tr>
</tbody>
</table>

The judge’s official ID badge was covering up the names.

There are two pieces of information on each line: a name and a number (the surfer’s score).
The Revised File

- The judges recorded the names of the contestants next to the scores!

- This means that our code won’t work!

Refer to:
threeHighestFailure.py
How can we fix this?

<table>
<thead>
<tr>
<th>Name</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Johnny</td>
<td>8.45</td>
</tr>
<tr>
<td>Juan</td>
<td>9.12</td>
</tr>
<tr>
<td>Joseph</td>
<td>8.45</td>
</tr>
<tr>
<td>Stacey</td>
<td>7.81</td>
</tr>
<tr>
<td>Aideen</td>
<td>8.05</td>
</tr>
<tr>
<td>Zack</td>
<td>7.21</td>
</tr>
<tr>
<td>Aaron</td>
<td>8.31</td>
</tr>
</tbody>
</table>
What if...

• ...we could **split** each row into two strings?
• Then we could convert the string for the score!

<table>
<thead>
<tr>
<th>Name</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Johnny</td>
<td>8.65</td>
</tr>
<tr>
<td>Juan</td>
<td>9.12</td>
</tr>
<tr>
<td>Joseph</td>
<td>8.45</td>
</tr>
<tr>
<td>Stacey</td>
<td>7.81</td>
</tr>
<tr>
<td>Aileen</td>
<td>8.05</td>
</tr>
<tr>
<td>Zack</td>
<td>7.21</td>
</tr>
<tr>
<td>Aaron</td>
<td>8.31</td>
</tr>
</tbody>
</table>

*The for loop shredder™*

- Each line contains a name and a number, as a string.
- To isolate the score, you need to cut each line in two.
split()

• What does it do?
  – Returns a list of words in the given string

• Example:
  
  names = “Joel Bob Yogi”
  print(names.split())

• How does it work?
  – Looks for whitespace to separate/delimit words.
    • You can override this to look for another delimiter by passing that character as an argument to `split()`
      names = “Joel,Bob,Yogi”
      print(names.split(“,”))

• Once we split a string, we get a bunch of separate strings.
  – So how do we store the resulting strings?

• Solutions:
  – **Multiple assignment!**
    • *Multiple variables* to hold all of the parts.
  – **Use a list!**
Splitting

• Let’s take a simple example:

```python
rock_band = "Al Carl Mike Brian"
```

A single variable is assigned...

...a single string, which contains four words.

• If we say `rock_bank.split()`, we can use 4 variables to hold all of the strings.

```python
(rhythm, lead, vocals, bass) = rock_band.split()
```

The left side of the assignment operator lists the variables to assign values to.

The right side of the assignment operator contains the call to the `split()` method.

How does this work?
Multiple Assignment

\[ \text{rock\_band} = \text{"Al Carl Mike Brian"} \]

\[ (\text{rhythm}, \text{lead}, \text{vocals}, \text{bass}) = \text{rock\_band}.\text{split}() \]

- Al
- Carl
- Mike
- Brian

Multiple variables... each with its own stringed value.
Exercise

• Now that we know how to split strings, let’s fix our problem with finding the top three scores!

• Here’s the code so far:

```python
file_handle = open("results.txt")
scores = []

for line in file_handle:
    line_float = float(line)
    scores.append(line_float)
file_handle.close()

scores.sort()
scores.reverse()

for i in range(0, 3):
    print(str(i+1) + ".", scores[i])
```

• Use `split()` and multiple assignment to fix it.

Refer to:
threeHighestMA.py

Refer to:
threeHighestList.py
And the winners are...

**Surf-A-Thon**

1. 9.12
2. 8.65
3. 8.45

You know the winning scores, but the winning surfers remain a mystery...
Hmm, what about the names?

- We managed to compute the top three scores
  - But what were the names associated with those scores?

How might we remember the names and scores for each surfer in the contest?
Would this work?

• What if we had two arrays, `scores` and `names`?
  – When we sort `scores`, could we just sort `names`?

Refer to: `sortNames.py`
Association lost!

- The associations between the surfer names and the scores are lost.
  - Names and scores are next to each other in the file.
  - In two separate arrays, the association is severed!
  - Sorting one array has no effect on the other!
- Can we force this to work?
The `index()` function for lists could help...

- `index()`: Looks for an item and returns its index value.
- Example:
  ```python
  position = scores.index(highest_score)
  ```

Here’s the algorithm:

- Make sure to keep the original order of scores and names.
  - Create a separate list for the sorted scores.
    - You can copy the scores list using `list()`
      ```python
      scores_copy = list(scores)
      ```
    - Find the indices of the top three scores using `index()`
    - Use those indices on the names list to get the names associated with those scores!

Refer to: `indexScores.py`
That sucked!

• That solution was pretty messy!
  – We needed to keep an extra copy of the scores
    • That allowed us to retain the original associations.
  – Loop-stravaganza!
    • The loops were meant to keep the logic simple.
    • Imagine if we jammed everything into as few loops as possible.

• Is there a better way of keeping the associations between data?
Indeed!

**Array**
A variable that allows data to enter at one end of a collection and leave at the other end, supporting a first-in, first-out mechanism

**Linked list**
A variable that contains data arranged as a matrix of multiple dimensions (but typically, only two)

**Queue**
A variable that has exactly two columns and (potentially) many rows of data

**Hash**
A variable with multiple indexed slots for holding data

**Set**
A variable that creates a chain of data where one data item points to another data item, which itself points to another data item, and another, and so on and so forth

**Multi-dimensional array**
A variable that contains a collection of unique data items

---

Here’s the one you need.
Hash

• Known as a dictionary in Python!
• Allows us to maintain associations between data!
• **Main point:** let’s associate a **key** with a piece of data (**value**)!
Dictionaries in action!

Put the key inside the square brackets...

scores[8.45] = 'Joseph'

...and put the value to the right of the assignment operator.

A new row of data is added to the hash. Note the association.

Note how the act of assigning a value to a key CREATES the hash entry (assuming it’s not already there). In Python, there’s no explicit “append()” method for hashes as there is for arrays.

Once you have a hash created, you can use the trusty for loop to iterate over each of the rows:
Dictionary Iteration

Take each of the keys in the hash in turn...

```python
for key in scores.keys():
    print(scores[key] + ' had a score of ' + str(name_part))
```

...and display a custom message using the data in each row of the hash.

The “keys()” built-in method returns an array of the keys of the hash. There are no extra points for guessing what the “values()” method does.

When referring to a value associated with a key, use square brackets (just like you did with array data).

Another hash method, called `items()`, returns each key-value pair in turn, and can be used with the for loop, too:

```python
for score, surfer in scores.items():
    print(surfer + ' had a score of ' + str(score))
```

Refer to: `dictionaryIteration.py`
Exercise

• Write a Rolodex program that allows a user to input the names and phone numbers of their contacts.
  – An input of “done” ends the input and prints the currently stored names and phone numbers, one line at a time.

Refer to:
rolodex.py
Additional Dictionary Notes

• **No `append()` method!**
  – To add data to the hash, you have to set up the key, value association manually using the format:
    ```python
    some_list[key] = value
    ```

• **What can be used as a key?**
  – Pretty much anything!
    • But, stick with numbers and strings!
    • Otherwise, it can get messy!

• **There’s no `sort()` method!**
  – The creator of the language didn’t bother to create it!
There is **no** `sort()` method for a hash.

- We can’t do `some_hash.sort()`!

But there is a general (not specifically for hashes) `sorted()` method!

Use the “`sorted()`” function to sort the keys of the “scores” hash.

```python
for each_score in sorted(scores.keys(), reverse = True):
    print('Surfer ' + scores[each_score] + ' scored ' + each_score)
```

The `sorted()` method works on the left side of the hash! i.e. `sorted()` sorts the keys, not the values!

- You *can* sort by the values, but it’s not easy.

Refer to: `threeHighestWithNames.py`
Time to party!

Fantastic! You’ve identified the top 3 surfers. Time to hit those waves, dude!

Surf-A-Thon

1. Juan  9.12
2. Johnny  8.65
3. Joseph  8.45