Dictionaries and Loops

Looping through dictionary values is very similar to looping through lists. The main difference is that we have to use the `items()`, `values()` or `keys()` method to get a list of key-value pairs, values, or keys respectively. We can then use the key-value pairs to access the values in the dictionary.

What are we going to do?

We're going to take votes in from users, store them in a dictionary, tally up the results (in another dictionary), and then print out the results.

Why is this important?

Looping over keys to get certain values or just looping over a dictionary is an essential skill to have.

Steps

1. Let's observe the code what we in the code.

- In the code here you can observe a couple of things. We have a couple of dictionaries.
  - We're going to loop through `fav_fruit_voters` and tally up the votes in `voting_results`.
  - We're also going to list who has voted.

```
fav_fruit_voters = {
    "daniel": "apple",
    "jessica": "apple",
    "michael": "banana",
    "john": "banana",
    "jessie": "apple",
    "jim": "orange",
    "jenny": "apple",
    "jason": "orange",
    "joseph": "banana",
    "james": "orange",
    "mary": "apple",
    "melody": "banana",
}

voting_results = {
    "banana": 0,
    "apple": 0,
    "orange": 0
}

# tally up code here.
```

- Let's add the code to loop through the voters here. Observe that we're using the `keys()` method to get a list of keys, and loop through them.

```
# ... dictionaries code here ...

print("Voters:")
for voter in fav_fruit_voters.keys():
    print(F"- {voter}")
```

- Here's what we get for an output

```
$ python fruit_census.py
Voters:
- daniel
- jessica
- michael
- john
- jessie
- jim
- jenny
- jason
- joseph
- james
- mary
- melody
```

2. Let's loop through the values in the dictionary using `values()`

- So what where we're going to do is loop through the `fav_fruit_voters` dictionary and tally up the votes in the `voting_results` dictionary.
    - To do this we're going to use the method `values()` on the dictionary, our knowledge of looping lists and, accessing dictionaries.
- Let's take a look at the code.

```
# ... dictionary code here

# list the voters
print("Voters:")
for voter in fav_fruit_voters.keys():
    print(F"- {voter}")

# tally up code here.
for vote in fav_fruit_voters.values():
    if vote == "banana":
        voting_results["banana"] += 1
    elif vote == "apple":
        voting_results["apple"] += 1
    elif vote == "orange":
        voting_results["orange"] += 1
```

```
print("Voting Results:")
print(voting_results)
```

- You can see here that we're looping through the values in the `fav_fruit_voters` dictionary and checking if the value is equal to a certain string. If it is we're going to add one to the value in the `voting_results` dictionary.
    - We're also printing out the `voting_results` dictionary. Next, we'll print out the results in a nicer way.
- Here's the output of the application.

```
$ python fruit_census.py
Voters:
- daniel
- jessica
- michael
- john
- jessie
- jim
- jenny
- jason
- joseph
- james
- mary
- melody
Voting Results:
{'banana': 4, 'apple': 5, 'orange': 3}
```

- You can see that we get the correct number of votes for each fruit if we go count them!
    - This is fantastic but we can make this look even nicer! Let's loop through the `voting_results` dictionary with `items()` so that we can loop through the dictionary and have access to the key-value pairs.

3. Looping through the voting_results dictionary with `items()`

- The great thing about looping through dictionaries is that you can get some context from looping over the entire dictionary and having access to the key-value pair.
    - This uses a technique that we haven't taken a long look at yet which is multiple assignments using the comma `,`.
- Let's take a look a the code.

```
# ... dictionary code omitted here ...
voting_results = {
    "banana": 0,
    "apple": 0,
    "orange": 0
}
```

```
# ... listing voters code omitted here ...
# ... tallying up code omitted here ...

print("Voting Results:")
for fruit, votes in voting_results.items():
    print(F"- The fruit: {fruit} has {votes} votes")
```

- You can see that `voting_results.items()` returns a list of tuples.
    - A tuple is a data type that is similar to a list except it is immutable (can't be changed).
    - We can use multiple assignments to assign the key to `fruit` and the value to `votes`.
    - You can see the benefit of this because now we can see both the key and the value in our loop!
- Let's take a look at the output of the application.

```
$ python fruit_census.py
Voters:
- daniel
- jessica
- michael
- john
- jessie
- jim
- jenny
- jason
- joseph
- james
- mary
- melody
Voting Results:
- The fruit: banana has 4 votes
- The fruit: apple has 5 votes
- The fruit: orange has 3 votes
```

4. Let's Refactor the tallying code to be more elegant.

- Let's take another look at the code for tallying the votes. You see here that we're using a lot of `if` statements to check if the vote is equal to a certain string.
    - Why don't we use our knowledge of reassigning dictionary values to make this code more elegant?
- The existing code looks like the code below.

```
# tally up code here.
for vote in fav_fruit_voters.values():
    if vote == "banana":
        voting_results["banana"] += 1
    elif vote == "apple":
        voting_results["apple"] += 1
```

```
    elif vote == "orange":
        voting_results["orange"] += 1
```

- The refactor of this code looks like the code below.

```
# ... dictionary and voters list code omitted ...

# tally up code here.
for vote in fav_fruit_voters.values():
    voting_results[vote] += 1

# ... printing out the results code omitted ...
```

- You can see that we're using the value of the vote to access the key in the `voting_results` dictionary and then adding one to it.
    - This is a much more elegant solution and it's easier to read!
- Note: Refactoring is something you'll want to do when you're writing code. It's a good idea to refactor your code to make it more readable and elegant.
    - This is something that you'll get better at with practice.
- If you run the code you shouldn't see any difference.

5. (Optional) Let's display the results in order using the `sorted` functions

- Sometimes, like in this instance, you'll want to have the results in a certain order.
    - We can use the `sorted` function to sort the dictionary by the key, this is a function that returns a sorted list.
    - We're going to use the `sorted` function and pass in the `voting_results.items()` as an argument. Let's take a look at the code looks like.

```
# ... all code omitted above ...

# print the results
print("Voting Results:")
for fruit, votes in sorted(voting_results.items()):
    print(F"- The fruit: {fruit} has {votes} votes")
```

- Note if you want to take a look at the documentation for the `sorted` function you can take a look at the documentation here.
- Great now you have some tools to be able to use dictionaries and loop through keys, values, and key-value pairs!

Challenge

Use your knowledge looping and user input to get more values from the user and add them to the dictionary.

- Note that you'll need to add some code to check if the value is not in the `voting_results` dictionary and add it if it isn't.

Conclusion

In this example we learned how to loop through dictionaries using the `keys()`, `values()`, and `items()` methods. We also learned how to use the `sorted` function to sort the dictionary by the key.