Lecture 21

Beyond Sequences: The Python Dictionary

24 Germinal, Year CCXXX

Song of the day: ***Ain't Nobody Know*** *by Gen Hoshino (2019).*

Part 1: *A Sample Problem, for Motivation*

Let's start with a simple problem to illustrate the usefulness of the topic we're covering next. Let's say we have a **file** that contains information for a large number of video games:

```
"","Name","Platform","Year_of_Release","Genre","Publisher","NA_players","EU_players","JP_players","Other_play
"1","Wii Sports","Wii","2006","Sports","Nintendo",41.36,28.96,3.77,8.45,82.53,76,51,"8",322,"Nintendo","E"
"2","Super Mario Bros.","NES","1985","Platform","Nintendo",29.08,3.58,6.81,0.77,40.24,NA,NA,"",NA,"",""
"3","Mario Kart Wii","Wii","2008","Racing","Nintendo",15.68,12.76,3.79,3.29,35.52,82,73,"8.3",709,"Nintendo",
"4","Wii Sports Resort","Wii","2009","Sports","Nintendo",15.61,10.93,3.28,2.95,32.77,80,73,"8",192,"Nintendo"
"5","Pokemon Red/Pokemon Blue","GB","1996","Role-Playing","Nintendo",11.27,8.89,10.22,1,31.37,NA,NA,"",NA,"",
"6","Tetris","GB","1989","Puzzle","Nintendo",23.2,2.26,4.22,0.58,30.26,NA,NA,"",NA,"",""
"7","New Super Mario Bros.","DS","2006","Platform","Nintendo",11.28,9.14,6.5,2.88,29.8,89,65,"8.5",431,"Ninte
"8","Wii Play","Wii","2006","Misc","Nintendo",13.96,9.18,2.93,2.84,28.92,58,41,"6.6",129,"Nintendo","E"
"9","New Super Mario Bros. Wii","Wii","2009","Platform","Nintendo",14.44,6.94,4.7,2.24,28.32,87,80,"8.4",594,
"10","Duck Hunt","NES","1984","Shooter","Nintendo",26.93,0.63,0.28,0.47,28.31,NA,NA,"",NA,"",""
```

Figure 1*: First ten lines of* ***video_game.csv*** *(****Source****).*

Write a function called `extract_data()` that accepts the file's address as its only parameter, and returns a list of each of these video games' information (as lists). The information we want to include will be:

- Name
- Platform
- Year of release
- Genre
- Publisher
- Developer

So, if our file only contained the first ten lines, our output would be:

```
[
    ['Wii Sports', 'Wii', '2006', 'Sports', 'Nintendo', 'Nintendo'],
    ['Super Mario Bros.', 'NES', '1985', 'Platform', 'Nintendo', ''],
    ['Mario Kart Wii', 'Wii', '2008', 'Racing', 'Nintendo', 'Nintendo'],
    ['Wii Sports Resort', 'Wii', '2009', 'Sports', 'Nintendo', 'Nintendo'],
    ['Pokemon Red/Pokemon Blue', 'GB', '1996', 'Role-Playing', 'Nintendo', ''],
    ['Tetris', 'GB', '1989', 'Puzzle', 'Nintendo', ''],
    ['New Super Mario Bros.', 'DS', '2006', 'Platform', 'Nintendo', 'Nintendo'],
    ['Wii Play', 'Wii', '2006', 'Misc', 'Nintendo', 'Nintendo'],
    ['New Super Mario Bros. Wii', 'Wii', '2009', 'Platform', 'Nintendo', 'Nintendo'],
```

```
    ['Duck Hunt', 'NES', '1984', 'Shooter', 'Nintendo', '']
]
```

The program would, then, look like **this**:

```
NAME_IDX, PLATFORM_IDX, YR_IDX, GENRE_IDX, PUB_IDX, DEV_IDX = 1, 2, 3, 4, 5, 15


def extract_data(filepath):
    data = []

    # Step 1: Attempt to safely open the file pointed to by filepath
    try:
        data_file = open(filepath, 'r')
    except FileNotFoundError:
        # Step 2: If the file doesn't exist, return our empty list
        return data

    # Step 3: If it does exist, iterate through each line, ignoring the header
    for line in data_file.readlines()[1:]:
        # Step 4: Prepare our list for data extraction
        datum_list = line.strip().replace('"', '').split(',')

        # Step 5: Create our data list
        datum = [
            datum_list[NAME_IDX],
            datum_list[PLATFORM_IDX],
            datum_list[YR_IDX],
            datum_list[GENRE_IDX],
            datum_list[PUB_IDX],
            datum_list[DEV_IDX]
        ]

        # Step 6: Append this data list to our main list
        data.append(datum)

    # Step 7: Close our file
    data_file.close()

    # Step 8: Return our whole data list
    return data
```

Okay, great. Let's see how big this list would end up being if we used the entirety of `video_game.csv` :

```
def main():
    filepath = "video_game.csv"
    data = extract_data(filepath)
    print(len(data))

main()
```

Ouput:

```
16719
```

Oof. That's quite a lot. But hey, we accomplished our task! The problem really arises when, for example, you show this to your boss, and they ask you something like:

> "Cool, can you just show me the data for 'Custom Robo'? The GameCube one?"

Or maybe something like:

> "Can you show me if the data for the first Dishonored game includes a valid developer field?"

Neither of these tasks are impossible with our current knowledge—they would just be tedious, and unnecessarily so.

The first question could be accomplished with a `for`-loop, for example:

```
NAME_IDX = 0

def retrieve_data(data, game_name):
    for datum in data:
        if datum[NAME_IDX] == game_name:
            return datum

    return []
```

And the second one would be even worse:

```
NAME_IDX, DEV_IDX = 0, 6
INVALID_ENTRY = ""

def does_include_developer_field(data, game_name):
    for datum in data:
        if datum[NAME_IDX] == game_name:
            if datum[DEV_IDX] != INVALID_ENTRY:
                return True
            else:
                return False

    return False
```

They might not look *awfully* complicated, but your boss's questions were, quite frankly, not very complex. The main inefficiency here is that we're completely reliant on **each game's index** within our data list. Since we don't know the index for each and every game, we *have* to use a `for`-loop to find it. And if that game doesn't even exist within our list, our `for`-loop search was literally a waste of time.

Surely, you may say to yourself, there must be a better way to store data that doesn't constrain us to having to base everything by numerical indices.

Right? Right you are! Say hello to the Python dictionary.

Part 2: *The Python Dictionary*

If you were to show a hierarchy chart of the types of objects we've encountered so far in Python, it might look a bit like this:


types

***Figure 2**: Python's data type hierarchy...thus far. (Courtesy of Prof. Reeves)*

What do we notice on the container side of this hierarchy? So far, every single one of them are *sequential*—that is, they hold data in a specific order. This makes sense if you consider the fact that we can index at all. If we can index a list at, say, position 2 (e.g. `lst[2]`), it's because that element comes after the element in position 1 (`lst[1]`) and before the element in position 3 (`lst[3]`).

This has the obvious advantage of being able to organize and access data that follows a certain order, such as ranges of numbers, or the titles of books from the same series. But as we saw above, when the order of the elements being contained doesn't especially matter, finding those elements quickly without having to resort to looping through the whole list is virtually impossible. You would have to know which index matches every single element to be able to quickly extract it. And I don't know about you, but I can't remember the contents of 16,719 elements.

What I'm trying to get at is that it would be nice if we could, instead, answer our boss's questions by doing something like this, instead:

```
>>> data["Custom Robo"]
['GC', '2004', 'Role-Playing', 'Nintendo', 'Noise Inc.']

>>> data["Dishonored"][DEV_IDX] != ""
True
```

Both of these are one-liners, unlike our solutions with lists. In other words, is there we could save data by any means that are not numbers? That's what dictionaries are for.

***Figure 3**: Python's container hierarchy. Now with dictionaries! (Courtesy of Prof. Reeves)*

A dictionary is actually pretty similar to what its literal equivalent. We have a word that we want to look up, and a definition associated to that word. This word-definition connection is referred to as a ***key-value pair***, where the word would be the key, and the definition would be the value.

Let's say we had a dictionary only containing the data for the games "Custom Robo" and "Dishonored". How would we go about creating it? The general structure of a dictionary is as follows:

```
your_dictionary = {key_1: value_1, key_2: value_2, ..., key_x: value_X}
```

That is, your key-value pairs are structures using a colon `:` (e.g. `key_17: value_17`), and they are separated by a comma `,`. Note here that, while I used numbers to differentiate between key-value pairs, **dictionaries are not sequences** (i.e. they don't actually have an order; that's their whole point). So, with this in mind, let's create our first dictionary. There are two ways to go about this:

```
import pprint  # for an output that's easier to read

# Method 1: All at once
data = {"Custom Robo": ['GC', '2004', 'Role-Playing', 'Nintendo', 'Noise Inc.'], "Dishonored": ['X360', '201
pprint.pprint(data)
```

Output:

```
{
    'Custom Robo': ['GC', '2004', 'Role-Playing', 'Nintendo', 'Noise Inc.'],
    'Dishonored': ['X360', '2012', 'Action', 'Bethesda Softworks', 'Arkane Studios']
}
```

You can think of this structure as saying the following:

> The key `'Custom Robo'` maps to the value `['GC', '2004', 'Role-Playing', 'Nintendo', 'Noise Inc.']` and the key of `'Dishonored'` maps to the value `['X360', '2012', 'Action', 'Bethesda Softworks', 'Arkane Studios']`.

Just like the word "tired" maps to the definition "(adjective) in need of sleep or rest; weary." in an actual dictionary.

The second way of creating a dictionary is by adding one entry at a time:

```
import pprint  # for an output that's easier to read

data = {}  # an empty dictionary

data["Dishonored"] = ['GC', '2004', 'Role-Playing', 'Nintendo', 'Noise Inc.']
data["Custom Robo"] = ['GC', '2004', 'Role-Playing', 'Nintendo', 'Noise Inc.']

pprint.pprint(data)
```

Output:

```
{
    'Custom Robo': ['GC', '2004', 'Role-Playing', 'Nintendo', 'Noise Inc.'],
    'Dishonored': ['X360', '2012', 'Action', 'Bethesda Softworks', 'Arkane Studios']
}
```

Same thing, essentially, just different ways to approach the same thing. Both are equally valid, but one way will almost always be more convenient than the other, depending on your situation.

There are two very important restrictions on dictionaries that we must keep in mind:

- ***Keys must be immutable***: If you want to access data that's being arranged by a key, what would be the use if anybody can just go ahead and change that key? It would literally be as if anybody could change the key that fits into the lock on your front door whenever they wanted. We don't want that.

```
>>> key = ["Pichu", "Pikachu", "Raichu"]

>>> value = "Electric"

>>> pokedex = {key: value}
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: unhashable type: 'list'

>>> key = tuple(key)  # turning our key into something immutable

>>> pokedex = {key: value}  # no errors!
```

```
>>> pokedex
{('Pichu', 'Pikachu', 'Raichu'): 'Electric'}
```

- ***A specific key can only exist once in a dictionary***: This also makes sense if you think about it this way. What if we had a dictionary of city coordinates. It would make absolutely no sense for they key of 'Angoulême, FR' to have two sets of coordinates.

```
>>> cities = {}

>>> city = "Angoulême, FR"
>>> coordinates = (45.18333, 0.71667)

>>> cities[city] = coordinates

>>> cities["Angoulême, FR"]
(45.18333, 0.71667)

>>> cities["Angoulême, FR"] = (45.650002, 0.160000)  # it makes no sense for a city to have two locations, so

>>> cities["Angoulême, FR"]
(45.650002, 0.16)
```

That's right; if you attempt to add a key-value pair using a pre-existing key, it will replace the previous value with the most recent assignment's value. This can be both a useful feature in dictionaries, so it would be great to add a simple `if` -statement as a safeguard:

```
if "Angoulême, FR" not in cities:
    # if the string "Angoulême, FR" doesn't yet exist as a key inside the cities dictionary,
    # then perform these steps
```

If you ever need to remove a specific key-value pair for whatever reason, we would use the keyword `del` :

```
actions = {"w": "up_key", "a": "left_key", "s": "down_key", "d": "right_key", " ": "space_bar"}
print("Before deletion: {}".format(actions))

del actions[" "]
print("After deletion: {}".format(actions))
```

Output:

```
Before deletion: {'w': 'up_key', 'a': 'left_key', 's': 'down_key', 'd': 'right_key', ' ': 'space_bar'}
After deletion: {'w': 'up_key', 'a': 'left_key', 's': 'down_key', 'd': 'right_key'}
```

Finally, you can determine the number of key-value pairs that your dictionary has by using our good ol' `len` function:

```
actions = {"w": "up_key", "a": "left_key", "s": "down_key", "d": "right_key", " ": "space_bar"}

print(len(actions))
```

Output:

```
5
```

Awesome. So now we know how to:

- Create an empty dictionary (e.g. `your_dict = {}`), as well as a non-empty dictionary (e.g. `your_dict = {"age": 28}`)
- Add new key-value pairs to existing dictionaries (e.g. `your_dict["favourite_colour"] = "PANTONE 18-4140 TCX"`)
- Update a pre-existing key-value pair in a dictionary (e.g. `your_dict["age"] = 29`)
- Delete a key-value pair in a dictionary (e.g. `del year_dict["age"]`)
- Check whether a certain key already exists inside a dictionary (e.g. `"favourite_colour" in your_dict`)
- Check how many key-value pairs exist in a dictionary (e.g. `len(your_dict)`)

So how can we change our `extract_data()` from earlier so that, instead of giving us a list of games in form of a list of lists, we instead received a dictionary that looked like this:

```
import pprint


def main():
    filepath = ARGUMENTS[DATA_IDX]
    data = extract_data_to_dict(filepath)
    pprint.pprint(data)


main()
```

Output:

```
{
    'Duck Hunt': ['NES', '1984', 'Shooter', 'Nintendo', ''],
    'Mario Kart Wii': ['Wii', '2008', 'Racing', 'Nintendo', 'Nintendo'],
    'New Super Mario Bros.': ['DS', '2006', 'Platform', 'Nintendo', 'Nintendo'],
    'New Super Mario Bros. Wii': ['Wii', '2009' 'Platform', 'Nintendo', 'Nintendo'],
    'Pokemon Red/Pokemon Blue': ['GB', '1996', 'Role-Playing', 'Nintendo', ''],
    'Super Mario Bros.': ['NES', '1985', 'Platform', 'Nintendo', ''],
    'Tetris': ['GB', '1989', 'Puzzle', 'Nintendo', ''],
    'Wii Play': ['Wii', '2006', 'Misc', 'Nintendo', 'Nintendo'],
    'Wii Sports': ['Wii', '2006', 'Sports', 'Nintendo', 'Nintendo'],
    'Wii Sports Resort': ['Wii', '2009', 'Sports', 'Nintendo', 'Nintendo']
}
```

The code is actually almost identical to our earlier function. The key (no pun intended) differences are that instead of initializing an empty list, we initialize an empty dictionary. The file opening process stays exactly the same:

```
def extract_data_to_dict(filepath):
    data = {}  # EMPTY DICTIONARY!

    # Step 1: Attempt to safely open the file pointed to by filepath
    try:
        data_file = open(filepath, 'r')
```

```
    except FileNotFoundError:
        # Step 2: If the file doesn't exist, return our empty list
        return data
```

Assuming that our file does open, we will now need to iterate and prepare each line for data extraction. This, too, stays exactly the same:

```
def extract_data_to_dict(filepath):
    data = {}

    # Step 1: Attempt to safely open the file pointed to by filepath
    try:
        data_file = open(filepath, 'r')
    except FileNotFoundError:
        # Step 2: If the file doesn't exist, return our empty list
        return data

    # Step 3: If it does exist, iterate through each line, ignoring the header
    for line in data_file.readlines()[1:]:
        # Step 4: Prepare our list for data extraction
        datum_list = line.strip().replace('"', '').split(',')
```

Now for the new stuff. For each line of this file, I want to create a new dictionary entry; a new key-value pair. As you can see in our example above, the keys seem to be the name of the video games, while the values seem to be the rest of the information that we had already extracted before. So let's isolate those two things properly:

```
NAME_IDX, PLATFORM_IDX, YR_IDX, GENRE_IDX, PUB_IDX, DEV_IDX = 1, 2, 3, 4, 5, 15


def extract_data_to_dict(filepath):
    data = {}


    # Step 1: Attempt to safely open the file pointed to by filepath
    try:
        data_file = open(filepath, 'r')
    except FileNotFoundError:
        # Step 2: If the file doesn't exist, return our empty list
        return data

    # Step 3: If it does exist, iterate through each line, ignoring the header
    for line in data_file.readlines()[1:]:
        # Step 4: Prepare our list for data extraction
        datum_list = line.strip().replace('"', '').split(',')

        # Step 5: Isolate the object that will serve as a key, and the object that will serve as the value
        key = datum_list[NAME_IDX]
        value = [
            datum_list[PLATFORM_IDX],
            datum_list[YR_IDX],
            datum_list[GENRE_IDX],
            datum_list[PUB_IDX],
            datum_list[DEV_IDX]
        ]
```

Finally, we add this entry into our dictionary, in case it doesn't already exist (data often has repeats, so you might want to save time and computational energy this way). This step is not strictly necessary; you may actually *want* your program to update data that has the same key. It depends on the instructions you get, or on the purpose that you're trying to achieve:

```
NAME_IDX, PLATFORM_IDX, YR_IDX, GENRE_IDX, PUB_IDX, DEV_IDX = 1, 2, 3, 4, 5, 15


def extract_data_to_dict(filepath):
    data = {}

    # Step 1: Attempt to safely open the file pointed to by filepath
    try:
        data_file = open(filepath, 'r')
    except FileNotFoundError:
        # Step 2: If the file doesn't exist, return our empty list
        return data

    # Step 3: If it does exist, iterate through each line, ignoring the header
    for line in data_file.readlines()[1:]:
        # Step 4: Prepare our list for data extraction
        datum_list = line.strip().replace('"', '').split(',')

        # Step 5: Add dictionary entry to our data list
        key = datum_list[NAME_IDX]
        value = [
            datum_list[PLATFORM_IDX],
            datum_list[YR_IDX],
            datum_list[GENRE_IDX],
            datum_list[PUB_IDX],
            datum_list[DEV_IDX]
        ]

        # Step 6: If this entry doesn't already exist in our dictionary, add it
        if key not in data:
            data[key] = value

    # Step 7: Close our file
    data_file.close()

    # Step 8: Return our whole data list
    return data
```

Don't forget to close your file and return your dictionary! **Here's** the full program.