Lecture 23

Dictionary Methods and Iteration

1 Floréal, Year CCXXX

Song of the day: ***Les Jolies Choses*** *by Polo & Pan (2021).*

Part 1: *Review*

Alright, let's do a simple dictionary problem to review how they work. Let's pretend we have a short CSV file that saves our preferences in various fields:

```
Field,Preference
Colour,Bleu de France
Movie,Liz and the Blue Bird
Composer,Debussy
Writer,Sanaë Lemoine
Programming Language,Swift
City,Kyoto
```

Figure 1*: Contents of file* ***preferences.csv****. Feel free to change it to your own preferences.*

Write a function, `extract_preferences()`, that does just that: attempts to open the CSV file, and creates a dictionary that saves a key-value pair for each line. The zeroth element of each line should be used as the key, and the first should be used as the value:

```
from sys import argv
from pprint import pprint

FILEPATH_IDX = 1
FILEPATH = argv[FILEPATH_IDX]


def main():
    preferences = extract_preferences(filepath=FILEPATH)
    pprint(preferences)
```

```
main()
```

Output:

```
{'City': 'Kyoto',
 'Colour': 'Bleu de France',
 'Composer': 'Debussy',
 'Movie': 'Liz and the Blue Bird',
 'Programming Language': 'Swift',
 'Writer': 'Sanaë Lemoine'}
```

You may assume that all data in the file, if opened, will be valid. Your function must accept the file's address as it's single parameter:

```
KEY_IDX = 0
VALUE_IDX = 1


def extract_preferences(filepath):
    """
    Extracts user preferences from a CSV file. Uses zeroth element in each line as
    Assumes all lines are valid entries. If file not found, returns an empty dict

    :param filepath: Address of CSV file in string form
    :return: A dictionary of preferences
    """
    preferences = {}

    # Step 1: Attempt to safely open the file under read mode
    try:
        file = open(filepath, 'r')
    except FileNotFoundError:
        # Step 2: If file doesn't exist, return an empty dictionary
        print("WARNING: File '{}' not found. Returning empty dictionary.".format(
        return preferences

    # Step 3: Iterate through every line in the file EXCEPT for the zeroth one (th
    for line in file.readlines()[1:]:
        # Step 4: Prep the file for data extraction
        line = line.strip().split(',')

        # Step 5: Extract data from line
        key, value = line[KEY_IDX], line[VALUE_IDX]

        # Step 6: Create new key-value pair
        preferences[key] = value
```

```
    # Step 7: Close file
    file.close()

    # Step 8: Return dictionary of preferences
    return preferences
```

*Figure 2: An **implementation** of* `extract_preferences()` *.*

Part 2: *Safe key lookup*

There are a couple of ways of safely getting the value corresponding to a key. We saw the `if` version earlier:

```
key = "band"

if key in preferences:
    value = preferences[key]
    print("This user's favourite {} is '{}'.".format(key, value))
else:
    print("This user does not have a favourite {}.".format(key))
```

This is fine, but there is actually a way of doing something similar in one line using ***conditional expressions***. This is my personal favourite way of doing this:

```
key = "band"
value = preferences[key] if key in preferences else None
```

The second line reads as such in English:

> Inside the parameter `value` , save the value corresponding to the key `key` in the dictionary `preferences` **if** `key` exists **in** `preferences` . If not (`else`), save the value of `None` instead.

You need not necessarily use `None` as an alternative value. For example:

```
key = "band"
value = preferences[key] if key in preferences else None

message = "This user's favourite {} is '{}'.".format(key, value) if value else "Th
print(message)
```

That is,

> If `value` is not `None`, save `"This user's favourite {} is '{}'.".format(key, value)` inside the variable `message`. If `value` is `None`, save the value `"This user does not have a favourite {}.".format(key)` instead.

The reason why I personally like this method is because it is not **dictionary exclusive**; you can use conditional expressions just about anywhere in Python. I'll be using them a lot from now on, so make sure to get used to them!

A third way of doing this is by using the dictionary method `get()`. Get takes two parameters:

1. `key`: The key you are trying to search up.
2. `alt`: The value returned if the key does not exist in the dictionary. This is an optional variable, with a default value of `None`.

```
key = "band"
alternative = "NA"
value = preferences.get(key, alternative)

# With a specified alternative value
message = "This user's favourite {} is '{}'.".format(key, value) if value != "NA"
print(message)


# Using the default alternative value
value = preferences.get(key)
message = "This user's favourite {} is '{}'.".format(key, value) if value else "Th
print(message)
```

This is probably the simplest way to safely extracting values from a dictionary, but again, the `get()` method only exists for Python dictionaries, and not anywhere else.

Part 2: Updating dictionaries

What is the user updated their preferences, changing some originals and adding new preferences?

```
latest_update = {"Colour": "Royal Blue", "Band": "The 1975", "Movie": "The French
```

What we ***cannot*** do is this:

```
preferences = preferences + latest_update
```

Output:

```
Traceback (most recent call last):
  File "Preferences.py", line 54, in <module>
    main()
  File "Preferences.py", line 50, in main
    preferences = preferences + latest_update
TypeError: unsupported operand type(s) for +: 'dict' and 'dict'
```

Instead, use the `update()` method:

```
preferences.update(latest_update)

pprint(preferences)
```

Output:

```
{'Band': 'The 1975',
 'City': 'Kyoto',
 'Colour': 'Royal Blue',
 'Composer': 'Debussy',
 'Movie': 'The French Dispatch',
 'Programming Language': 'Swift',
 'Writer': 'Sanaë Lemoine'}
```

So, this is basically the only safe way of quickly "adding" dictionaries together.

Part 3: *Dictionary views*

I mentioned that dictionaries are non-sequential containers, and therefore cannot be iterated through the same way lists, ranges, strings, and tuples can be:

```
for key_value in preferences:
    print(key_value)
```

Output:

```
Colour
Movie
Composer
Writer
Programming Language
City
```

While the code doesn't crash, we're clearly not getting the full key-value pair information; we only get back the keys—and not in any particular order. So how do we specify which information we want to iterate through? We use something called **views**.

```
keys = preferences.keys()
pprint(keys)

values = preferences.values()
pprint(values)

key_values = preferences.items()
pprint(key_values)
```

Output:

```
dict_keys([
    'Colour',
    'Movie',
    'Composer',
    'Writer',
    'Programming Language',
    'City'
])
dict_values([
    'Bleu de France',
    'Liz and the Blue Bird',
    'Debussy',
    'Sanaë Lemoine',
    'Swift',
    'Kyoto'
])
dict_items([
    ('Colour', 'Bleu de France'),
    ('Movie', 'Liz and the Blue Bird'),
    ('Composer', 'Debussy'),
    ('Writer', 'Sanaë Lemoine'),
    ('Programming Language', 'Swift'),
    ('City', 'Kyoto')
])
```

These `dict_keys()`, `dict_values()`, and `dict_items()` objects are `view` objects. They are containers of the sequence family that can be iterated in a similar fashion to lists:

```
key_values = preferences.items()

 for key_value in key_values:
    print(key_value)
```

Output:

```
('Colour', 'Bleu de France')
('Movie', 'Liz and the Blue Bird')
('Composer', 'Debussy')
('Writer', 'Sanaë Lemoine')
('Programming Language', 'Swift')
('City', 'Kyoto')
```

However, we cannot index a view object in the same way we can index a list, or a tuple:

```
Traceback (most recent call last):
  File "Preferences.py", line 57, in <module>
    main()
  File "Preferences.py", line 53, in main
    print(key_values[0])
TypeError: 'dict_items' object is not subscriptable
```

So we must **explicitely cast them to a list or tuple in order to do this**:

```
key_values = preferences.items()
key_values = tuple(key_values)

for index in range(len(key_values)):
    print(key_values[index])
```

Output:

```
('Colour', 'Bleu de France')
('Movie', 'Liz and the Blue Bird')
('Composer', 'Debussy')
('Writer', 'Sanaë Lemoine')
('Programming Language', 'Swift')
('City', 'Kyoto')
```

Part 4: *Dictionary traversal*

Let's use views in this sample problem: let's say we had the video game file from Monday:

```
"","Name","Platform","Year_of_Release","Genre","Publisher","NA_players","EU_player
"1","Wii Sports","Wii","2006","Sports","Nintendo",41.36,28.96,3.77,8.45,82.53,76,5
"2","Super Mario Bros.","NES","1985","Platform","Nintendo",29.08,3.58,6.81,0.77,40
"3","Mario Kart Wii","Wii","2008","Racing","Nintendo",15.68,12.76,3.79,3.29,35.52,
"4","Wii Sports Resort","Wii","2009","Sports","Nintendo",15.61,10.93,3.28,2.95,32.
"5","Pokemon Red/Pokemon Blue","GB","1996","Role-Playing","Nintendo",11.27,8.89,10
"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,
"8","Wii Play","Wii","2006","Misc","Nintendo",13.96,9.18,2.93,2.84,28.92,58,41,"6.
"9","New Super Mario Bros. Wii","Wii","2009","Platform","Nintendo",14.44,6.94,4.7,
"10","Duck Hunt","NES","1984","Shooter","Nintendo",26.93,0.63,0.28,0.47,28.31,NA,N
```

***Figure 3**: The first eleven lines of **video_game.csv**.*

Using the `extract_data_into_list()` function from **Monday's class**, how would we write a function to get the names of all the games from a specific console? For instance, if your boss wanted the names of all the **NES** games, you would be able to do this:

```
def main():
    game_data = extract_data_to_dict(filepath="video_game.csv")
    games = get_all_games_of_console(data=game_data, console_name="NES")

    pprint(games)

main()
```

Output:

```
['Super Mario Bros.',
 'Duck Hunt',
 'Super Mario Bros. 3',
 'Super Mario Bros. 2',
 'The Legend of Zelda',
 'Zelda II: The Adventure of Link',
 'Teenage Mutant Ninja Turtles',
 'Excitebike',
 'Golf',
 'Dragon Warrior III',
 'Kung Fu',
```

'Baseball',
'Dragon Warrior IV',
'World Class Track Meet',
"Mike Tyson's Punch-Out!!",
'Metroid',
'Super Mario Bros.: The Lost Levels',
'Dragon Warrior II',
'Dragon Warrior',
'Ice Hockey',
'Pro Wrestling',
'Mario Bros.',
'Teenage Mutant Ninja Turtles II: The Arcade Game',
'Pro Yakyuu Family Stadium',
'Tennis',
'Volleyball',
'R.C. Pro-Am',
'Mahjong',
'Soccer',
'Rad Racer',
'Pinball',
'Kid Icarus',
"Kirby's Adventure",
'Yoshi',
"Disney's DuckTales",
"Ghosts 'n Goblins",
'Donkey Kong Classics',
'Xevious',
'F1 Race',
'Mega Man 2',
'Ninja Hattori Kun: Ninja wa Shuugyou Degogiru no Maki',
'Ice Climber',
'Nintendo World Cup',
'4 Nin uchi Mahjong',
"Pro Yakyuu Family Stadium '87",
'Teenage Mutant Ninja Turtles III: The Manhattan Project',
'Gradius',
'Gyromite',
"Hogan's Alley",
'Dragon Ball: Daimaou Fukkatsu',
'Gegege no Kitarou 2: Youkai Gundan no Chousen',
'Castlevania',
'TwinBee',
'Ganbare Goemon! Karakuri Douchuu',
"Disney's Chip 'n Dale: Rescue Rangers",
"Pro Yakyuu Family Stadium '88",
'Mega Man 3',
'Doraemon',
'Commando',
'Donkey Kong Jr.',
'Lode Runner',
'Famicom Jump: Eiyuu Retsuden',

```
 'Popeye',
 'Tag Team Match M.U.S.C.L.E.',
 'Adventure Island',
 'Bomberman',
 '1942',
 'NES Open Tournament Golf',
 'Tetris 2',
 'Star Soldier',
 "Castlevania II: Simon's Quest",
 'Mega Man 4',
 'Balloon Fight',
 "Castlevania III: Dracula's Curse",
 'Final Fantasy',
 'Clu Clu Land',
 'Mega Man',
 'Mega Man 5',
 'Mappy',
 'The Mysterious Murasame Castle',
 'Mega Man 6',
 'Wrecking Crew',
 'Bomberman II',
 'Jurassic Park',
 'Tetris 2 + Bombliss',
 "Famista '91",
 "Famista '92",
 'Final Fantasy I & II',
 'Teenage Mutant Ninja Turtles: Tournament Fighters',
 'Adventures of Lolo']
```

I personally would approach this by using the `items()` method to extract both the key (i.e. the name of the video game) and the value (i.e. the data that contains the console of the game) all at once:

```
def get_all_games_of_console(data, console_name):
    games = []

    # Step 1: To iterate through both keys and values, we use the items() dictiona
    for key_value in data.items():
        # Step 2: Since we know items() returns a tuple of the form (KEY,VALUE), v
        game_name, game_data = key_value

        # Step 3: Append game name if and only if this game's console name is equa
        if game_data[0] == console_name and game_name not in games:
            games.append(game_name)

    # Step 4: Return list
    return games
```

Here's the finished implementation.