

DATA SCIENCE FUNDAMENTALS

LESSON 5

Hay Kranen
Monday October 8th, 2018



TODAY'S PROGRAMME

How are we doing?
Handing in exercises
Assignment #2
Recap
Break
F-Strings
Dicts
Dicts in lists & JSON
Lunch break

HOW ARE WE DOING?

- 1) On a scale from 1 - 10, how well are you doing in class?
- 2) Which part is the hardest?
- 3) If there's one thing you could change during class, what would that be?

Lesson 5

- Dicts
- Dict indexing
- Replacing values in a dict
- Dict methods: keys(), values(), get()
- Lists with dicts with lists with dicts, wow
- Looping over dicts using items()



How are you doing?

It's important for Jonas and me to get a grip on how you are doing. Please enter these three questions so we get a global view on how you're doing.

ASSIGNMENT #2

What your program should do:

Your program **must**:

- Read the [provided csv](#) file and convert the data to a multidimensional list (list with lists)
- Print the contents of the list properly formatted **with the index number** (e.g. '0: Mark Rutte was born in 1967 and a member of the VVD party') and also show how many people there are in the database.
- Ask for four options: remove a person, add a person, save the database to csv or quit the program.
- 'Remove a person' should ask for an index number, and then remove that person from the list.
- 'Add a person' should ask for the same four fields that also exist in the original CSV file and add those to the list.
- 'Save the database' should save the modified list in the same format **in** the original file so that whenever the program is quit and run again the mutations done in the program are properly reflected in the file. **Alternatively**, the list should be saved whenever the user 'removes' or 'adds' a person.
- The program should indefinitely provide the four options again after an action is completed until the user chooses the 'quit the program' option.

The program should run until the user quits

Don't change the file, change the list and save that

Write to the same file that you read from

Number of lines are *just an indication*

RECAP

```
# define the lists
friends = [
    ["Pieter", " "],
    ["Jantje", " "],
    ["Greta", " "]
]

# create a for loop
for friend in friends:
    friend[1] = input("What is your favourite snack " + friend[0] + "? ")
```

```
friends = [
    ["Arjen"],
    ["Emiel"],
    ["Ayco"]
]

for friend in friends:
    snack = input(" What is your favourite snack? ")
    friend.append(snack)
```



```
line = line.split(",")  
player = line[0].strip()  
club = line[1].strip()  
value = line[2].strip()
```

```
line = line.strip().split(",")  
player = line[0]  
club = line[1]  
value = line[2]
```

```
f = open("footballers.csv")  
lines = f.read().splitlines()
```

```
for line in lines:  
    line = line.split(",")  
    player = line[0]  
    club = line[1]  
    value = line[2]
```

```
f = open("footballers.csv")  
lines = f.read().splitlines()  
f.close()
```

```
with open("footballers.csv") as f:  
    lines = f.read().splitlines()
```

```
# -*- coding: utf-8 -*-
"""

Created on Wed Oct  3 12:09:11 2018

@author:
"""

# -*- coding: utf-8 -*-
"""

Created on Wed Oct  3 11:32:14 2018

@author:
"""

#open and manipulate a csv file
newfile = open("paintings2.csv", "w")
csv = open("paintings.csv")
```



```
slice = sentence[start:end]
print(slice)
```

		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

\mathbb{F} -STRINGS

String formatting

```
newentry = newpainting + " is a painting by " + newname + " which was sold for: $ " + editprice
```


DICTIONARIES

[illegible]

```
names = ["Tinus", "Barrie", "Hans"]  
snacks = ["Oreo", "Twix", "Mars"]
```

names	snacks
Tinus	Oreo
Barrie	Twix
Hans	Mars

```
friends = [  
    ["Tinus", "Barrie", "Hans"]  
    ["Oreo", "Twix", "Mars"]  
]
```

```
friends = {  
    "Tinus": "Oreo",  
    "Barrie": "Twix",  
    "Hans": "Mars"  
}
```


Movie facts

Create a program that shows the **title**, **release year**, **duration** (in minutes) and **director** of your favourite movie.


- * Create a new **dictionary** with the **four** keys mentioned above. Year and duration should be integers, title and director strings.
- * Loop over the keys in the dict and create a variable **value** that contains the value associated with the key.
- * Use an **f-string** to format and print the data.

Extend your program

- * When you print the **duration** of the movie, print the number plus minutes (e.g. '117 minutes') instead of just the number
- * Add a new key called **actors** with a **list of actors**, and when you get to this list in your loop, **join** them together as a string

Tips

- * Everything you need for this exercise is in the first two chapters of the **examples-3** Jupyter Notebook.
- * Remember that accessing a key in a dict works the same as with a list
- * Use the **items()** method to loop over keys and values at the same time



```
title: Blade Runner
year: 1982
duration: 117
director: Ridley Scott
```



```
actors: Harrison Ford, Rutger Hauer, Sean Young
```

DICTIONARIES IN LISTS & JSON

```
friends = {  
  "Tinus" : "Oreo",  
  "Barrie" : "Twix",  
  "Hans" : "Mars"  
}
```

```
{  
  "name" : "Tinus",  
  "snack" : "Oreo"  
}  
  
{  
  "name" : "Barrie",  
  "snack" : "Twix"  
}  
  
{  
  "name" : "Hans",  
  "snack" : "Mars"  
}
```

```
friend1 = {  
  "name" : "Tinus",  
  "snack" : "Oreo"  
}  
  
friend2 = {  
  "name" : "Barrie",  
  "snack" : "Twix"  
}  
  
friend3 = {  
  "name" : "Hans",  
  "snack" : "Mars"  
}
```

```
[  
  {  
    "name" : "Tinus",  
    "snack" : "Oreo"  
  },  
  {  
    "name" : "Barrie",  
    "snack" : "Twix"  
  },  
  {  
    "name" : "Hans",  
    "snack" : "Mars"  
  },  
]
```



```
[
  {
    "name" : "Tinus",
    "snack" : "Oreo"
  },
  {
    "name" : "Barrie",
    "snack" : "Twix"
  },
  {
    "name" : "Hans",
    "snack" : "Mars"
  },
]
```

Movie database

Create a program that reads the provided **movies.json** file, asks for a year, and displays all movies that are from that year.

- * Import the **json** library
- * Ask the user for a year
- * Open the provided **movies.json** file and read the contents to a new variable using **json.load()**
- * Loop over the movies and display them only if they are released in the year provided by the user.

Extend your program

- * Provide the user with other filtering options, such as:
 - * Movies longer than a certain duration
 - * Movies where the director is also an actor
 - * Movies that were directed by a specific director
- * **Be creative and think of other ways of filtering the data**

Tips

- * Everything you need for this exercise is in the 'lesson 5' chapters of the **examples-3** Jupyter Notebook.
- * The **import json** statement should be the first thing in your file.
- * When using f-strings, you can get tripped up when putting quotes in the curly braces, because they are already declaring the strings.
- * You don't need to convert the values from the JSON file to int, a value without quotes in JSON automatically becomes a number!
- * Don't call your file **json.py**, but call it something else (e.g. **moviedb.py**)
- * When you get encoding errors on Windows, try using Spyder or the Anaconda Prompt
- * Remember the type that **input()** returns and the type of the **year** value in the dictionary