Python

Introduction to Programming

Comp07027

Lecture 10

Files

Files

While a program is running, its data is in memory.

When programs stop running, data in memory disappears.

If we want to store data permanently then we have to put it somewhere else, outside of the program.

Python can store data in external files or databases (files are simpler, but not as flexible as databases). We'll look at files.

Text Files

FilesDemo.py

```
1  string1 = ("John")
2  string2 = ("Paul")
3  string3 = ("George")
4  string4 = ("Ringo")
5
6  demo_file = open("demo.txt","w")
7
8  demo_file.write(string1)
9  demo_file.write(string2)
10 demo_file.write(string3)
11 demo_file.write(string4)
12 demo_file.write("Fab")
13 demo_file.write("4")
14
15 demo_file.close()
16
17 demo_file = open("demo.txt","r")
18 contents_of_file = demo_file.read()
19 print(contents_of_file)
```

Run: FilesDemo

```
JohnPaulGeorgeRingoFab4

Process finished with exit code 0
```

If we open a file (***"demo.txt"***) we create a file object and assign it to *demo_file*. **Notice the quotes ""**
The **open()** function has two parameters – the name (and maybe the path) of the file, and a "**w**" because we plan to write to the file.

Text Files

FilesDemo.py

```
1  string1 = ("John")
2  string2 = ("Paul")
3  string3 = ("George")
4  string4 = ("Ringo")
5
6  demo_file = open("demo.txt","w")
7
8  demo_file.write(string1)
9  demo_file.write(string2)
10 demo_file.write(string3)
11 demo_file.write(string4)
12 demo_file.write("Fab")
13 demo_file.write("4")
14
15 demo_file.close()
16
17 demo_file = open("demo.txt","r")
18 contents_of_file = demo_file.read()
19 print(contents_of_file)
```

Run: FilesDemo

```
JohnPaulGeorgeRingoFab4

Process finished with exit code 0
```

If ***demo.txt*** does not already exist it will be created.
If it does exist then it will be overwritten by the new version we create.

Text Files

FilesDemo.py

```
1  string1 = ("John")
2  string2 = ("Paul")
3  string3 = ("George")
4  string4 = ("Ringo")
5
6  demo_file = open("demo.txt","w")
7
8  demo_file.write(string1)
9  demo_file.write(string2)
10 demo_file.write(string3)
11 demo_file.write(string4)
12 demo_file.write("Fab")
13 demo_file.write("4")
14
15 demo_file.close()
16
17 demo_file = open("demo.txt","r")
18 contents_of_file = demo_file.read()
19 print(contents_of_file)
```

Run: FilesDemo

```
JohnPaulGeorgeRingoFab4

Process finished with exit code 0
```

To put data in the file we use the file object's **write** method. We can write variables (as long as they contain strings) or we can write values directly (again, only if they are strings)

Text Files

FilesDemo.py

```
1  string1 = ("John")
2  string2 = ("Paul")
3  string3 = ("George")
4  string4 = ("Ringo")
5
6  demo_file = open("demo.txt","w")
7
8  demo_file.write(string1)
9  demo_file.write(string2)
10 demo_file.write(string3)
11 demo_file.write(string4)
12 demo_file.write("Fab")
13 demo_file.write("4")
14
15 demo_file.close()
16
17 demo_file = open("demo.txt","r")
18 contents_of_file = demo_file.read()
19 print(contents_of_file)
```

Run: FilesDemo

```
JohnPaulGeorgeRingoFab4

Process finished with exit code 0
```

When we have finished writing to the file we call its **close()** method.
This closes and saves the text file and makes it available for reading later.

Text Files

FilesDemo.py

```
1  string1 = ("John")
2  string2 = ("Paul")
3  string3 = ("George")
4  string4 = ("Ringo")
5
6  demo_file = open("demo.txt","w")
7
8  demo_file.write(string1)
9  demo_file.write(string2)
10 demo_file.write(string3)
11 demo_file.write(string4)
12 demo_file.write("Fab")
13 demo_file.write("4")
14
15 demo_file.close()
16
17 demo_file = open("demo.txt","r")
18 contents_of_file = demo_file.read()
19 print(contents_of_file)
```

Run: FilesDemo

```
JohnPaulGeorgeRingoFab4

Process finished with exit code 0
```

To **read** the file we open and assign it exactly as before except that instead of "**w**" the second parameter is "**r**", indicating that we want to read it.

Text Files

FilesDemo.py

```
1  string1 = ("John")
2  string2 = ("Paul")
3  string3 = ("George")
4  string4 = ("Ringo")
5
6  demo_file = open("demo.txt","w")
7
8  demo_file.write(string1)
9  demo_file.write(string2)
10 demo_file.write(string3)
11 demo_file.write(string4)
12 demo_file.write("Fab")
13 demo_file.write("4")
14
15 demo_file.close()
16
17 demo_file = open("demo.txt","r")
18 contents_of_file = demo_file.read()
19 print(contents_of_file)
```

Run: FilesDemo

```
JohnPaulGeorgeRingoFab4

Process finished with exit code 0
```

The contents of *demo.txt* are assigned to *contents_of_file* and this string variable can be used e.g. printed

Text Files

FilesDemo.py

```
1  string1 = ("John")
2  string2 = ("Paul")
3  string3 = ("George")
4  string4 = ("Ringo")
5
6  demo_file = open("demo.txt","w")
7
8  demo_file.write(string1)
9  demo_file.write(string2)
10 demo_file.write(string3)
11 demo_file.write(string4)
12 demo_file.write("Fab")
13 demo_file.write("4")
14
15 demo_file.close()
16
17 demo_file = open("demo.txt","r")
18 contents_of_file = demo_file.read()
19 print(contents_of_file)
```

Run: FilesDemo

```
JohnPaulGeorgeRingoFab4

Process finished with exit code 0
```

The output from the print() function is **one** string of text corresponding to the **six** strings written to the file earlier. Notice that they are concatenated with no space between.

Text Files

FilesDemo.py

```
1  string1 = ("John")
2  string2 = ("Paul")
3  string3 = ("George")
4  string4 = ("Ringo")
5
6  demo_file = open("demo.txt","w")
7
8  demo_file.write(string1)
9  demo_file.write(string2)
10 demo_file.write(string3)
11 demo_file.write(string4)
12 demo_file.write("Fab")
13 demo_file.write("4")
14
15 demo_file.close()
16
17 demo_file = open("demo.txt","r")
18 contents_of_file = demo_file.read()
19 print(contents_of_file)
```

Run: FilesDemo

```
JohnPaulGeorgeRingoFab4

Process finished with exit code 0
```

demo.txt - Notepad

File Edit Format View Help

JohnPaulGeorgeRingoFab4

The text file can be viewed in notebook (for instance). Notice that the strings are concatenated here too.

Text Files

This way of saving data is effective (we can store & retrieve) but is decidedly limiting. We can only store strings!!

Therefore anything we want to store has to be converted to a string (in the last example we stored “4” not 4)

There is a more useful way – one which ***preserves*** data types

Pickling

PicklingDemo.py | FilesDemo.py

```
1  import pickle
2
3  string1 = "Creosote"
4  number1 = 37
5  list1 = [12, "Monkeys"]
6
7  demo_pck_file = open("pck_file.pck", "wb")
8
9  pickle.dump(str
10 pickle.dump(numb
11 pickle.dump(list
12 pickle.dump("Qu
13
14 demo_pck_file.c
15
16 demo_pck_file =
17
18 item1 = pickle.load(demo_pck_file)
19 item2 = pickle.load(demo_pck_file)
20 item3 = pickle.load(demo_pck_file)
21 item4 = pickle.load(demo_pck_file)
22
23 print(item1, "is a", type(item1))
24 print(item2, "is a", type(item2))
25 print(item3, "is a", type(item3))
26 print(item4, "is a", type(item4))
```

Run: PicklingDemo

```
Creosote is a <class 'str'>
37 is a <class 'int'>
[12, 'Monkeys'] is a <class 'list'>
Quite an assortment is a <class 'str'>

Process finished with exit code 0
```

The basic procedure for saving to a pickle file is the same (with some slightly different commands). However, the big difference is that the original data types are preserved.

Pickling

PicklingDemo.py | FilesDemo.py

```
1  import pickle
2
3  string1 = "Creosote"
4  number1 = 37
5  list1 = [12, "Monkeys"]
6
7  demo_pck_file = open("pck_file.pck","wb")
8
9  pickle.dump(string1, demo_pck_file)
10 pickle.dump(number1, demo_pck_file)
11 pickle.dump(list1, demo_pck_file)
12 pickle.dump("Quite an assortment", demo_pck_file)
13
14 demo_pck_file.close()
15
16 demo_pck_file = open("pck_file.pck", "rb")
17
18 item1 = pickle.load(demo_pck_file)
19 item2 = pickle.load(demo_pck_file)
20 item3 = pickle.load(demo_pck_file)
21 item4 = pickle.load(demo_pck_file)
22
23 print(item1, "is a", type(item1))
24 print(item2, "is a", type(item2))
25 print(item3, "is a", type(item3))
26 print(item4, "is a", type(item4))
```

Create three variables, a string, a number and a list

Run: PicklingDemo

```
'list'>
lass 'str'>

Process finished with exit code 0
```

Pickling

PicklingDemo.py | FilesDemo.py

```
1  import pickle
2
3  string1 = "Creosote"
4  number1 = 37
5  list1 = [12, "Monkeys"]
6
7  demo_pck_file = open("pck_file.pck","wb")
8
9  pickle.dump(string1, demo_pck_file)
10 pickle.dump(number1, demo_pck_file)
11 pickle.dump(list1, demo_pck_file)
12 pickle.dump("Quite an assortment", demo_pck_file)
13
14 demo_pck_file.close()
15
16 demo_pck_file = open("pck_file.pck", "rb")
17
18 item1 = pickle.load(demo_pck_file)
19 item2 = pickle.load(demo_pck_file)
20 item3 = pickle.load(demo_pck_file)
21 item4 = pickle.load(demo_pck_file)
22
23 print(item1, "is a", type(item1))
24 print(item2, "is a", type(item2))
25 print(item3, "is a", type(item3))
26 print(item4, "is a", type(item4))
```

Run: PicklingDemo

```
'list'>
lass 'str'>

code 0
```

Create three variables, a string, a number and a list

Open a pickle file **“pck_file.pck”** and use **“wb”** to write to the file

Pickling

PicklingDemo.py | FilesDemo.py

```
1  import pickle
2
3  string1 = "Creosote"
4  number1 = 37
5  list1 = [12, "Monkeys"]
6
7  demo_pck_file = open("pck_file.pck","wb")
8
9  pickle.dump(string1, demo_pck_file)
10 pickle.dump(number1, demo_pck_file)
11 pickle.dump(list1, demo_pck_file)
12 pickle.dump("Quite an assortment", demo_pck_file)
13
14 demo_pck_file.close()
15
16 demo_pck_file = open("pck_file.pck", "rb")
17
18 item1 = pickle.load(demo_pck_file)
19 item2 = pickle.load(demo_pck_file)
20 item3 = pickle.load(demo_pck_file)
21 item4 = pickle.load(demo_pck_file)
22
23 print(item1, "is a", type(item1))
24 print(item2, "is a", type(item2))
25 print(item3, "is a", type(item3))
26 print(item4, "is a", type(item4))
```

Run: PicklingDemo

'list'>
lass 'str'>
code 0

Create three variables, a string, a number and a list

Open a pickle file **“pck_file.pck”** and use **“wb”** to write to the file

dump each of the variables into the file

Pickling

PicklingDemo.py | FilesDemo.py

```
1  import pickle
2
3  string1 = "Creosote"
4  number1 = 37
5  list1 = [12, "Monkeys"]
6
7  demo_pck_file = open("pck_file.pck","wb")
8
9  pickle.dump(string1, demo_pck_file)
10 pickle.dump(number1, demo_pck_file)
11 pickle.dump(list1, demo_pck_file)
12 pickle.dump("Quite an assortment", demo_pck_file)
13
14 demo_pck_file.close()
15
16 demo_pck_file = open("pck_file.pck", "rb")
17
18 item1 = pickle.load(demo_pck_file)
19 item2 = pickle.load(demo_pck_file)
20 item3 = pickle.load(demo_pck_file)
21 item4 = pickle.load(demo_pck_file)
22
23 print(item1, "is a", type(item1))
24 print(item2, "is a", type(item2))
25 print(item3, "is a", type(item3))
26 print(item4, "is a", type(item4))
```

Run: PicklingDemo

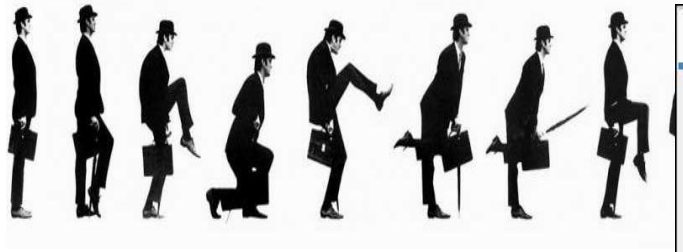
```
'list'>
lass 'str'>

code 0
```

Create three variables, a string, a number and a list

Open a pickle file **“pck_file.pck”** and use **“wb”** to write to the file

dump each of the variables into the file

close the file



Pickling

PicklingDemo.py | FilesDemo.py

```
1  import pickle
2
3  string1 = "Creosote"
4  number1 = 37
5  list1 = [12, "Monkeys"]
6
7  demo_pck_file = open("pck_file.pck","wb")
8
9  pickle.dump(string1, demo_pck_file)
10 pickle.dump(number1, demo_pck_file)
11 pickle.dump(list1, demo_pck_file)
12 pickle.dump("Quite an assortment", demo_pck_file)
13
14 demo_pck_file.close()
15
16 demo_pck_file = open("pck_file.pck", "rb")
17
18 item1 = pickle.load(demo_pck_file)
19 item2 = pickle.load(demo_pck_file)
20 item3 = pickle.load(demo_pck_file)
21 item4 = pickle.load(demo_pck_file)
22
23 print(item1, "is a", type(item1))
24 print(item2, "is a", type(item2))
25 print(item3, "is a", type(item3))
26 print(item4, "is a", type(item4))
```

Run: PicklingDemo

```
Creosote is a <class 'str'>
37 is a <class 'int'>
[12, 'Monkeys'] is a <class 'list'>
Quite an assortment is a <class 'str'>

Process finished with exit code 0
```

We can then **open** the pickled file **“pck_file.pck”** again and use **“rb”** to **load** from it.
We need a separate load for each element (in order). Now we can use them e.g. print them.

Files

- Text files
  - are human readable
  - only handle strings
  - easily parsed by many other programming languages
- Pickled files
  - are NOT human readable
  - preserve data structures and types (with some limitations)
  - Data can only be reconstructed by Python applications

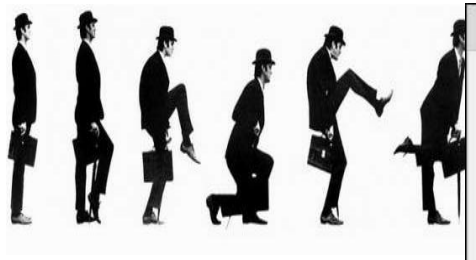
Pickled Files

Files are external to, and independent of, the program.

Therefore we can't manipulate them directly.

We need to bring the contents of the file into the program.

It makes sense to store data in a useful structure, like a **list**, because Python provides us with methods to manipulate a list when we bring it into the program.



Lists Revisted

Recall from the last lecture:

insert

append

delete

find

FilesDemo.py | PicklingDemo2.py | ListsRevisited.py

```
1  fruits = ["banana", "apple", "grape"]
2
3  def print_fruits():
4      print()
5      print("fruits = ",fruits)
6      print("The first element of fruits is", fruits[0])
7      print("The last element of fruits is",fruits[-1])
8      print()
9
10 print_fruits()
11
12 #******  INSERT  ******
13 fruits.insert(1,"orange")
14 print("After INSERTING")
15 print_fruits()
16
17 #******  APPEND  ******
18 fruits.append("raspberry")
19 print("After APPENDING")
20 print_fruits()
21
22 #******  DELETE  ******
23 del fruits[2]
24 print("After DELETING")
25 print_fruits()
26
27 #******  FINDING  ******
28 print("FINDING")
29 print("The index of grape is", fruits.index("grape"))
```

Run: ListsRevisited

```
"C:\Program Files\Python36\python.exe" "F:/Python Courses/FilesDemoFolde

fruits =  ['banana', 'apple', 'grape']
The first element of fruits is banana
The last element of fruits is grape

After INSERTING

fruits =  ['banana', 'orange', 'apple', 'grape']
The first element of fruits is banana
The last element of fruits is grape

After APPENDING

['banana', 'orange', 'apple', 'grape', 'raspberry']
element of fruits is banana
ement of fruits is raspberry

ING

['banana', 'orange', 'grape', 'raspberry']
element of fruits is banana
ement of fruits is raspberry

FINDING
The index of grape is 2

Process finished with exit code 0
```

We looked at this in a bit of detail in the previous lecture

Pickled Files and Lists

We have seen that lists are very handy for storing different types of data.

We can create a list, and pickle it.

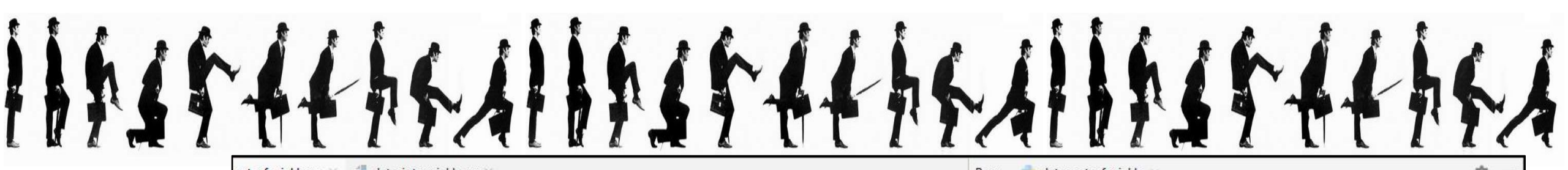
data_out_of_pickle.py | data_into_pickle.py

```
1  import pickle
2
3  the_list = [["a",1], ["b",2], ["c",3]]
4  print(the_list)
5  pickle_demo = open("pickle_demo.pck", "wb")
6
7  pickle.dump(the_list, pickle_demo)
8
9  pickle_demo.close()
10
```

Run: data_into_pickle

```
"C:\Program Files\Python36\python.exe" "F:/Python Co
[['a', 1], ['b', 2], ['c', 3]]

Process finished with exit code 0
```

This program creates our list in a python program and finishes by “dumping” the list into an external pickled file called, in this case, **pickle_demo**.
This only needs to be done once.



Pickles and Lists

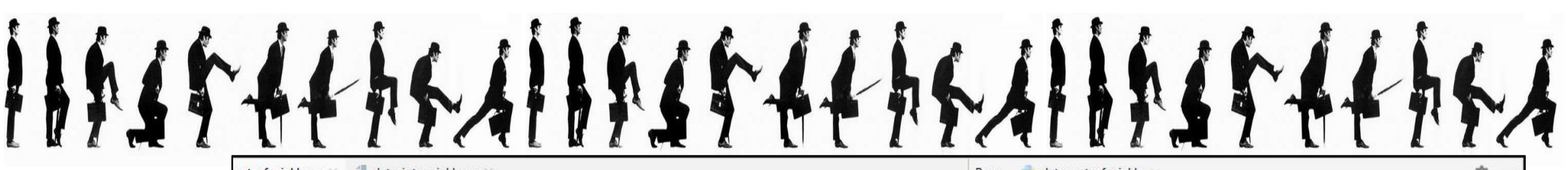
out_of_pickle.py | data_into_pickle.py

```
import pickle

demo_of_pickle = open("pickle_demo.pck", "rb")
my_list = pickle.load(demo_of_pickle)
demo_of_pickle.close()

print("Original version of the file")
print(my_list)

print("****** after APPEND ******")
my_list.append(["e",5])
print(my_list)

print("****** after INSERT ******")
my_list.insert(3,["d",4])
print(my_list)

#****** write new version of list to pickled fil
demo_of_pickle = open("pickle_demo.pck", "wb")
pickle.dump(my_list, demo_of_pickle)
demo_of_pickle.close()
```

Run: data_out_of_pickle

```
"C:\Program Files\Python36\python.exe" "F:/Python Courses/FilesDem
Original version of the file
[['a', 1], ['b', 2], ['c', 3]]
****** after APPEND ******
'b', 2], ['c', 3], ['e', 5]]
 INSERT ******
'b', 2], ['c', 3], ['d', 4], ['e', 5]]

d version of the file
'b', 2], ['c', 3], ['d', 4], ['e', 5]]

shed with exit code 0
```

This second program makes use of the pickled file **pickle_demo** that we created previously in another program.
This program begins by "loading" the list in the pickled file into the local list called **my_list**.
We can then treat **my_list** as we would any other list



Pickles and Lists

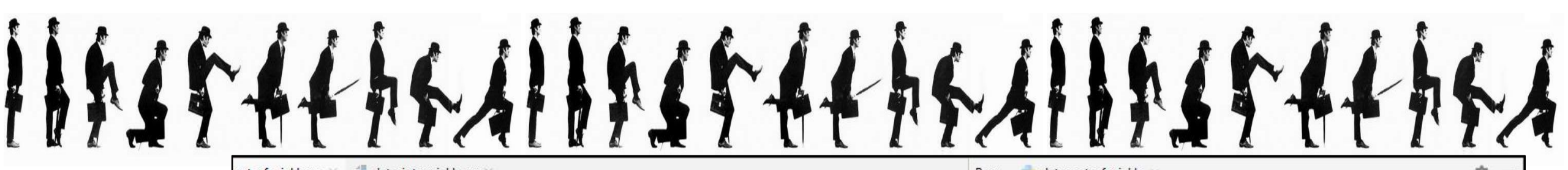
out_of_pickle.py | data_into_pickle.py

```
import pickle

demo_of_pickle = open("pickle_demo.pck", "rb")
my_list = pickle.load(demo_of_pickle)
demo_of_pickle.close()

print("Original version of the file")
print(my_list)

print("****** after APPEND ******")
my_list.append(["e",5])
print(my_list)

print("****** after INSERT ******")
my_list.insert(3,["d",4])
print(my_list)

#****** write new version of list to pickled file
demo_of_pickle = open("pickle_demo.pck", "wb")
pickle.dump(my_list, demo_of_pickle)
demo_of_pickle.close()
```

We can now load the list we had pickled.

Run: data_out_of_pickle

```
"C:\Program Files\Python36\python.exe" "F:/Python Courses/FilesDem
Original version of the file
[['a', 1], ['b', 2], ['c', 3]]
****** after APPEND ******
[['a', 1], ['b', 2], ['c', 3], ['e', 5]]
****** after INSERT ******
[['a', 1], ['b', 2], ['c', 3], ['d', 4], ['e', 5]]


Process finished with exit code 0
```



Pickles and Lists

out_of_pickle.py | data_into_pickle.py

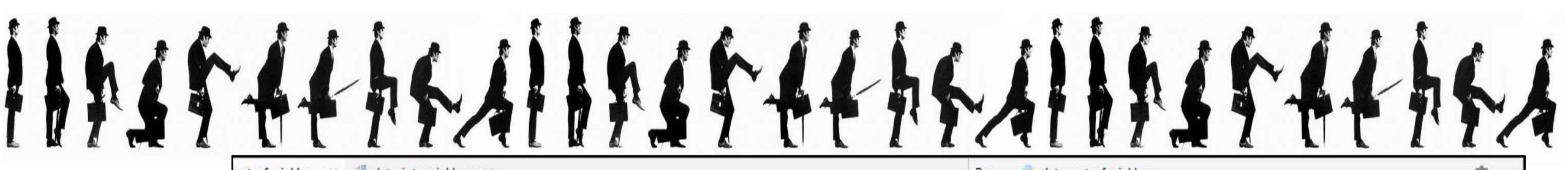
```
import pickle

demo_of_pickle = open("pickle_demo.pck", "rb")
my_list = pickle.load(demo_of_pickle)
demo_of_pickle.close()

print("Original version of the file")
print(my_list)

print("****** after APPEND ******")
my_list.append(["e",5])
print(my_list)

print("****** after INSERT ******")
my_list.insert(3,["d",4])
print(my_list)

#****** write new version of list to pickled file
demo_of_pickle = open("pickle_demo.pck", "wb")
pickle.dump(my_list, demo_of_pickle)
demo_of_pickle.close()
```

We can now load the list we had pickled.

When we have the list back in the program we can use all the functions and methods of lists.

Run: data_out_of_pickle

```
"C:\Program Files\Python36\python.exe" "F:/Python Courses/FilesDem
Original version of the file
[['a', 1], ['b', 2], ['c', 3]]
****** after APPEND ******
[['a', 1], ['b', 2], ['c', 3], ['e', 5]]
****** after INSERT ******
[['a', 1], ['b', 2], ['c', 3], ['d', 4], ['e', 5]]

Process finished with exit code 0
```



Pickles and Lists

out_of_pickle.py | data_into_pickle.py

```
import pickle

demo_of_pickle = open("pickle_demo.pck", "rb")
my_list = pickle.load(demo_of_pickle)
demo_of_pickle.close()

print("Original version of the file")
print(my_list)

print("****** after APPEND ******")
my_list.append(["e",5])
print(my_list)

print("****** after INSERT ******")
my_list.insert(3,["d",4])
print(my_list)

#****** write new version of list to pickled fi
demo_of_pickle = open("pickle_demo.pck", "wb")
pickle.dump(my_list, demo_of_pickle)
demo_of_pickle.close()
```

We can now load the list we had pickled.

When we have the list back in the program we can use all the functions and methods of lists.

When we have finished manipulating the list we can pickle it again

Run: data_out_of_pickle

```
"C:\Program Files\Python36\python.exe" "F:/Python Courses/FilesDem
Original version of the file
[['a', 1], ['b', 2], ['c', 3]]
****** after APPEND ******
[['a', 1], ['b', 2], ['c', 3], ['e', 5]]
****** after INSERT ******
[['a', 1], ['b', 2], ['c', 3], ['d', 4], ['e', 5]]

Process finished with exit code 0
```

Exercise 1

Create two programs

- The first should create an empty list (which will contain strings representing Scottish cities or towns) and save it to a pickled file.
- The second should
  - retrieve the list.
  - allow the user to change the data (add, delete, view, find)
  - save it.
- Run program 1 once.
- Run program 2 multiple times, editing the list each time.

Exercise 2

Edit Exercise 1 to allow the name of the town/city and its population to be stored/edited/retrieved.

Hint:

You could create a list of lists, each of the nested lists have two elements – name and population

Questions??