

LEARN CODING

ale66

LISTS AND THEIR PRACTICAL LIMITATIONS

SO FAR

While they admit arbitrary members, lists are best understood as iterables of homogeneous values



NESTED LISTS

Represent multi-dimensional data

The meaning of data and the objective of the code guide the organization in lists and sublists

Example: weekend temperatures in Naples (from Thur. to Sun.)

```
1 my2Dlist = [[18, 22, 20, 26], [26, 27, 28, 21], [18, 16, 21, 20]]
```

as with lists of strings, by simple nesting of indices we can access specific data points

Temperature on the first Thursday of October?

```
1 print(my2Dlist[0][0])
```

```
1 my2Dlist = [[18, 22, 20, 26], [26, 27, 28, 21], [18, 16, 21, 20]]
```

Average temperature on Thursday in October?

```
1 accumulator = 0
2 count = 0
3
4 for weekend in my2Dlist:
5     accumulator += weekend[0]
6     count += 1
7
8 if count > 0:
9     average_temp_thursday = accumulator / count
10 else:
11     # avoid 0 as it is a valid average temp.
12     average_temp_thursday = NaN
```

WORK WITH NESTED LISTS

Textual data sets may also be organized in 2-d lists

```
1 my_writers = [  
2     ['Dickens', 1812],  
3     ['King', 1947],  
4     ['J.K. Rowling', 1965],  
5     ['Christie', 1890]  
6 ]  
7  
8 for row in my_writers:  
9     print(row)
```

Here a clear semantic structure remains *latent* within the Python variables

Print authors only:

```
1 for row in my_writers:  
2     print(row[0])
```

Print authors and they their year of birth?

```
1 for row in my_writers:  
2     for element in row:  
3         print(element)
```


OBSERVATIONS

To extract the needed data we need to know something about their logical organization: the data structure

- each element is a weekend, starting Thursday

A fixed structure is required for instructions to work:

- each pair is author followed by year

A more flexible data structure is needed

DICTIONARIES

THE NON-LIST

An unsorted **bag** of values, each with its own label, called **key**.

```
1 my_data = {'author': 'Agatha Christie',  
2           'nickname': 'Mary Westmacott',  
3           'year_of_birth': 1890  
4           }
```



PROPERTIES

A dictionary is a collection of **key:value** pairs that is

- unordered
- changeable, and
- indexed

Dictionaries are Python most powerful data structure

EXTENSIBILITY

The more we have, the more we can put

```
1 my_writers = [  
2     {'author': 'Dickens', 'year_of_birth': 1812},  
3     {'author': 'King', 'year_of_birth': 1947},  
4     {'author': 'J.K. Rowling', 'year_of_birth': 1965},  
5     {'author': 'Agatha Christie', 'nickname': 'Mary Westmacott', 'year_of_birth': 1897},  
6 ]
```

ACCESSING ITEMS

```
1 my_dict = {'author': 'Agatha Christie',  
2           'nickname': 'Mary Westmacott',  
3           'year_of_birth': 1890  
4 }  
5  
6 print(my_dict['author'])
```

```
1 NOW = 2024  
2  
3 delta = NOW - my_dict['year_of_birth']  
4  
5 print(f{{my_dict['author']}} was born {delta} years ago.)
```

AMENDING VALUES

```
1 my_dict['year_of_birth'] = 2019
2
3 print(my_dict['year_of_birth'])
```

```
1 king = my_writers[1]
2
3 print(king['author'])
4
5 king['author'] = 'Stephen King'
```

We have more information but **my_writers** is unchanged

COMBINING INDICES

```
1 my_writers[1]['author'] = 'Stephen King'
2
3 print(my_writers)
4
5 [{ 'author': 'Dickens', 'year_of_birth': 1812 },
6  { 'author': 'Stephen King', 'year_of_birth': 1947 },
7  { 'author': 'J.K. Rowling', 'year_of_birth': 1965 },
8  { 'author': 'Agatha Christie', 'nickname': 'Mary Westmacott', 'year_of_birth': 1895 }]
```


ITERATION

As the data structure is completely arbitrary, simple indexing (**i=0,1, ...**) does not work anymore.

A lazy iterable, similar to **range**, will serve the **key:value** pairs on request

```
1  # always key first
2  for key, value in my_dict.items():
3      print(key, value)
```

```
1  author Agatha Christie
2  nickname Mary Westmacott
3  year_of_birth 1890
```

THE **key** AND **value** VARIABLE NAMES

```
1 for key in my_dict.items():  
2     print(key)
```

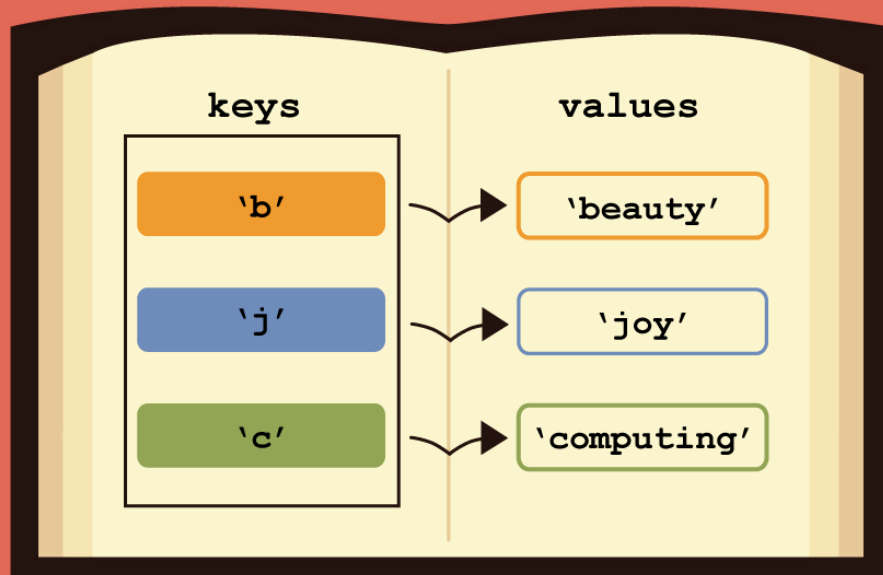
```
1 author  
2 nickname  
3 year_of_birth
```

```
1 for value in my_dict.items():  
2     print(key)
```

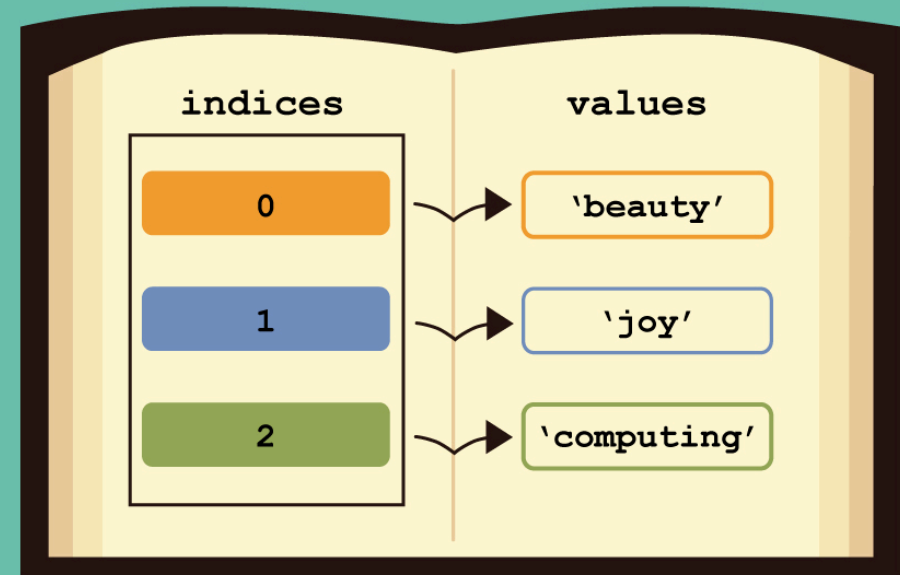
```
1 Agatha Christie  
2 Mary Westmacott  
3 1890
```

LISTS VS DICTIONARIES

 dictionaries



 lists



GROW A LIST

```
1 my_list = []  
2  
3 my_list.append(10)  
4  
5 my_list.append(20)  
6  
7 print(my_list)  
8  
9 [10, 20]
```

Direct access is always possible:

```
1 print(my_list[-1])  
2  
3 20
```

GROW A DICTIONARY

```
1 my_dict = {}  
2  
3 my_dict['name'] = 'Tom'  
4  
5 my_dict['age'] = 35  
6  
7 print(my_dict)
```

```
1 {'name': 'Tom', 'age': 35}
```

```
1 print(my_dict['name'])  
2  
3 Tom
```

Disadvantage: dictionaries are inherently unordered

ONE KEY, MANY VALUES

The screenshot shows the Microsoft Excel interface. The ribbon at the top includes 'AutoSave' (Off), 'File', 'Home', 'Insert', 'Draw', and 'Page Layout'. The active sheet is 'Book1'. The formula bar shows 'A5' and an empty formula. The table data is as follows:

	A	B
1	Student_Id	Student_Name
2	S1	John
3	S2	Mary
4		
5		

```
1 my_python = {'Student_ID': ['S1', 'S2'],  
2               'Student_Name': ['John', 'Mary']  
3 }
```

RECAP: COMMON OPERATIONS

```
1 # create an empty dictionary
2 my_dict = {}
3
4 # create a new dict with 3 items
5 my_dict = {'one':1, 'two':2, 'three':3}
6
7 # access an element
8 print(my_dict['two'])
```

Data updates:

```
1 # add an entry
2 my_dict['four'] = 4
3
4 # change an entry
5 my_dict['one'] = 'uno'
6
7 # count items
8 howmany = len(my_dict)
```

keys AND values

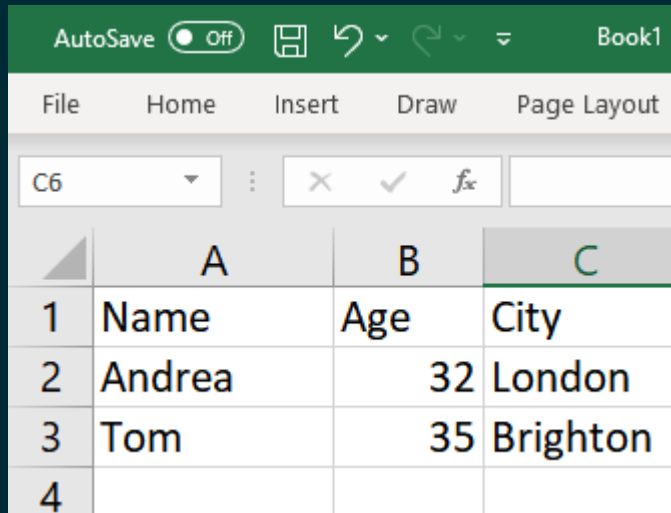
```
1  # iterate over keys
2  for item in my_dict.keys():
3      print (item)
4
5  # iterate over values
6  for item in my_dict.values():
7      print(item)
```

Inspect the data structure:

```
1  # list out the keys
2  my_dict.keys()
3
4  # list out the values
5  my_dict.values()
```


QUIZ!

Create a dictionary out of the following data:



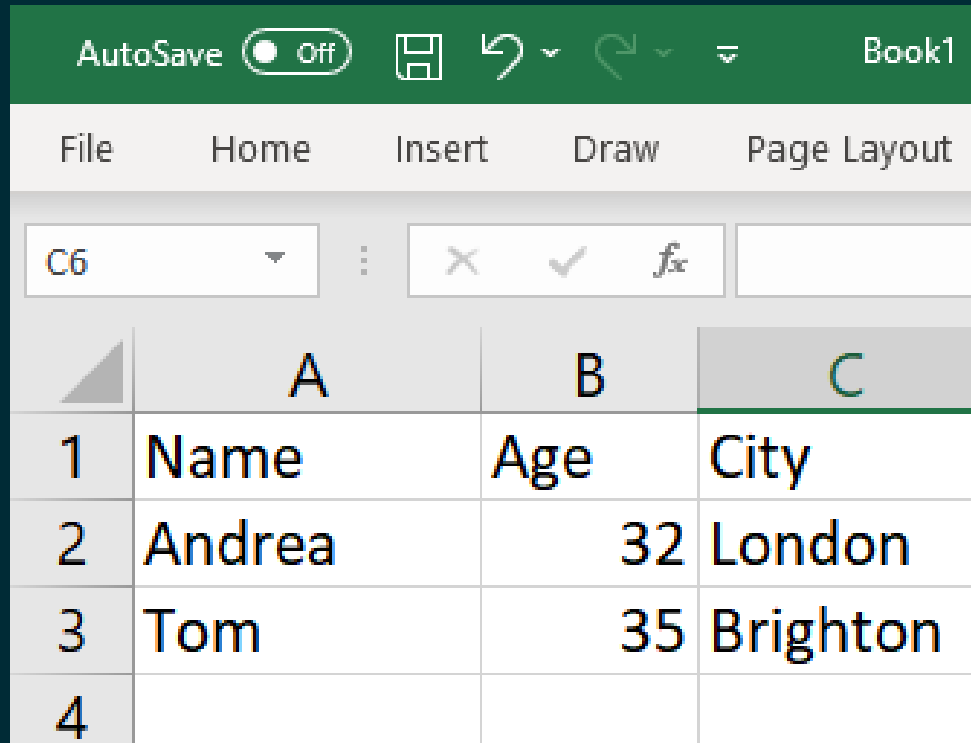
	A	B	C
1	Name	Age	City
2	Andrea	32	London
3	Tom	35	Brighton
4			



Pull up VS Code and test a solution

SOLUTION

Create a dictionary out of the following data:



	A	B	C
1	Name	Age	City
2	Andrea	32	London
3	Tom	35	Brighton
4			

```
1 my_data = {'Name': ['Andrea', 'Tom'],  
2           'Age': [32, 35],  
3           'City': ['London', 'Brighton']  
4 }
```