

COMP1021
Introduction to Computer Science

Dictionaries

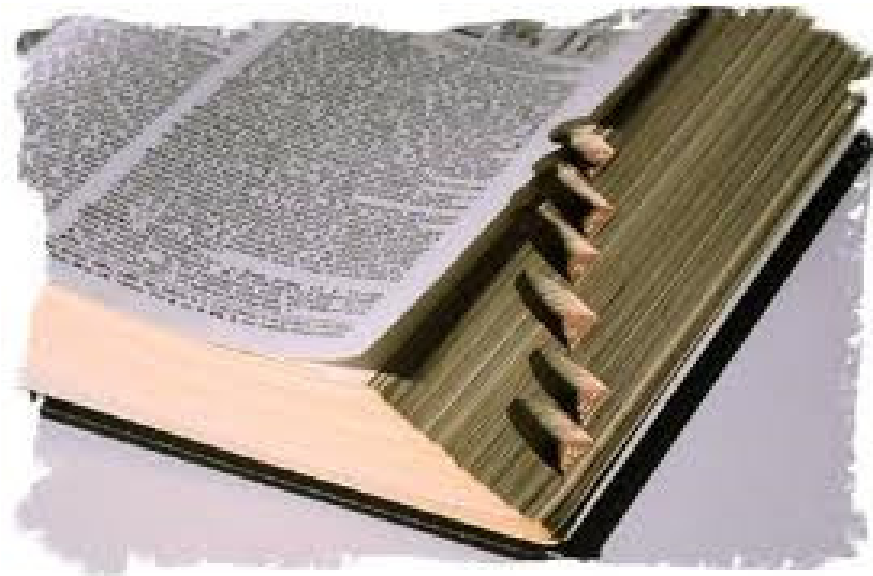
David Rossiter and Gibson Lam

Outcomes

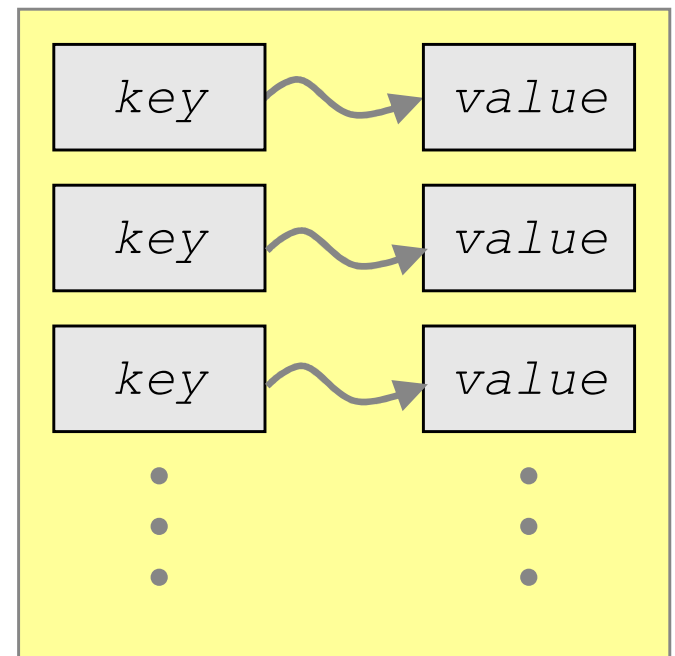
- After completing this presentation, you are expected to be able to:
 1. Explain the difference between a dictionary and a list (or a tuple)
 2. Create and use a dictionary in Python

A Dictionary

- A *dictionary* contains pairs of *key* and *value*
- One key-value pair forms one entry in the dictionary
- Almost anything can be used as a key such as a number, a string or a tuple
- Almost anything can be used as a value such as a number, a string or a list



Mapping



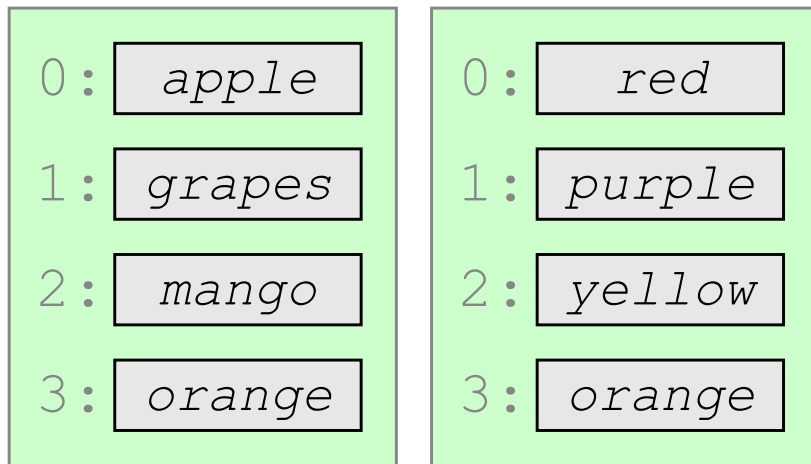
Accessing the Content

- A mapping structure is similar to a list or a tuple but the mapping structure gives a more flexible indexing of data
- For a list `[]` or a tuple `()` or a string (which is a kind of a list for text only):
 - You access the data using integer indices only i.e. `x[0]`, `x[45]`, and so on
- For a mapping:
 - You access the data by using keys, (which can be *almost* any kind of data type)

A Dictionary May be Better Than a List

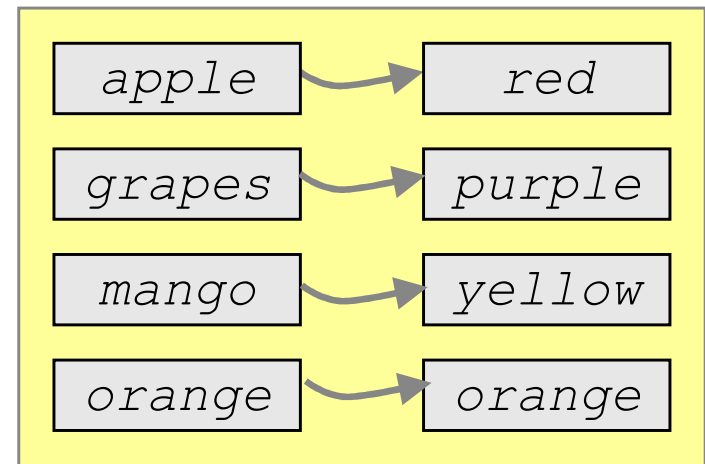
- For example, to create a mapping of four fruits and their colours, you could do this:

Using Two Lists



or

Using a Dictionary



- Possibly, two lists might be required (left)
- However, only one dictionary would be required (right), which is more suitable for this particular purpose

An Example Dictionary

- Let us use a dictionary to store the position and size of some heads in this picture:



Creating a Dictionary

- You can create a dictionary using a list of key and value pairs, like this:

The name of the dictionary



```
heads = {"David": (589, 106, 48, 63),  
         "Gibson": (474, 102, 44, 58),  
         "Jogesh": (438, 146, 45, 60),  
         "Paul": (522, 162, 55, 68) }
```

A dictionary uses braces { }



x position y position width height



- In this example the key is a string, and the value is a tuple

Retrieving Items

- You can use the key as an index to retrieve an item from a dictionary:

```
where_is_paul = heads["Paul"]
```

- After running the above code, `where_is_paul` contains a tuple of 4 numbers, i.e.

```
(522, 162, 55, 68)
```



Changing Items

- Items in a dictionary can be modified whenever you want, for example:

```
heads["David"] = (588, 104, 48, 57)
```

- After running the above line of code, the value for David will be changed from:

```
(589, 106, 48, 63)
```

to:

```
(588, 104, 48, 57)
```

Inserting and Deleting Items

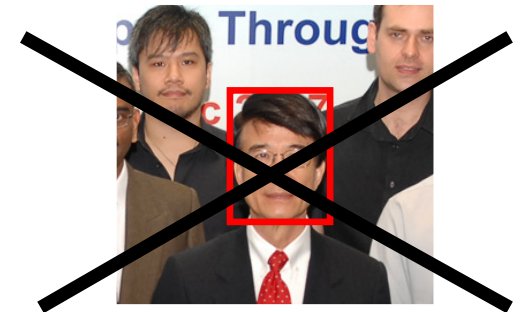
- You can insert or delete an item in a dictionary any time you want to
 - To insert a new item in the dictionary, simply assign a value to a new key:

```
heads["Sean"] = (628, 146, 46, 58)
```

- To delete an item, use the `del` keyword to delete a particular key and its value:


```
del heads["Paul"]
```

Paul Chu used to run HKUST but he left, so let's dump him from our database



Dictionary Traversal

- ‘Traversal’ means ‘going through each item’
- You can use `.items()` to access all dictionary content
- For example, to print the location of all heads you can use this code:



```
for key, value in heads.items() :  
    print(key, value)
```

Example Result of Dictionary Traversal

- The output of the code in the previous slide is:

```
Paul (522, 162, 55, 68)  
Gibson (474, 102, 44, 58)  
Jogesh (438, 146, 45, 60)  
David (588, 104, 48, 57)
```

- Note that when you use this kind of loop to look at the dictionary content you will not be able to delete a dictionary pair in the middle of the loop

Before Python Version 3.6

- It's not possible to know what the order of items, i.e. the keys, is in a dictionary
- For example, when we made the dictionary, the input order of the keys was:
“David”, “Gibson”, “Jogesh”, and “Paul”
- Surprisingly, the output order is different:
“Paul”, “Gibson”, “Jogesh” and “David”

```
Paul (522, 16  
Gibson (474,  
Jogesh (438,  
David (588, 1
```

Python Version 3.6 and Higher

- If you run the code a couple of slides ago in Python 3.6 or higher, you will see this result:

```
David (588, 104, 48, 57)
Gibson (474, 102, 44, 58)
Jogesh (438, 146, 45, 60)
Paul (522, 162, 55, 68)
```
- As you can see, the input order of the dictionary content has been preserved
- This happens from Python version 3.6 (released a few years ago) onwards

Getting the Keys Only

Paul
Gibson
Jogesh
David

- Possibly, you only want to consider the keys
- To do that you can use `.keys()` like this:

```
for key in heads.keys():  
    print(key)
```

- Alternatively, you can use the dictionary in the loop directly to get the keys, like this:

```
for key in heads:  
    print(key)
```

Getting the Values and Others

- To only consider the values, you can use `.values()` like this:

```
(522, 162, 55, 68)
(474, 102, 44, 58)
(438, 146, 45, 60)
(589, 106, 48, 63)
```

```
for value in heads.values():
    print(value)
```

- To check if a key is in the dictionary you can use the `in` operator:

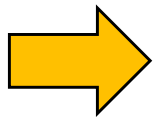
```
if "David" in heads:
    print("he is there!")
```

```
he is there!
```


You Cannot Use a List as a Key

- Although almost anything can be used as a key in a dictionary, you cannot use a list as a key, e.g.:

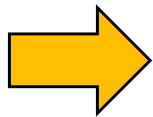
```
heads = { [474, 102, 44, 58]: "Gibson" }
```



```
Traceback (most recent call last):  
  File "<pyshell#3>", line 1, in <module>  
    heads = { [474, 102, 44, 58]: "Gibson" }  
TypeError: unhashable type: 'list'
```

- If you want to put a collection as a key, you will need to use a tuple

```
heads = { (474, 102, 44, 55): "Gibson" }  
print(heads)
```



```
{ (474, 102, 44, 55): 'Gibson' }
```