Learning Outcome

- Define and use various types of data structure to solve problem:
  - Dictionary
  - Tuple
  - Set

What is data structure

- A data structure is a method of organizing data in a computer / program. We have already met one:

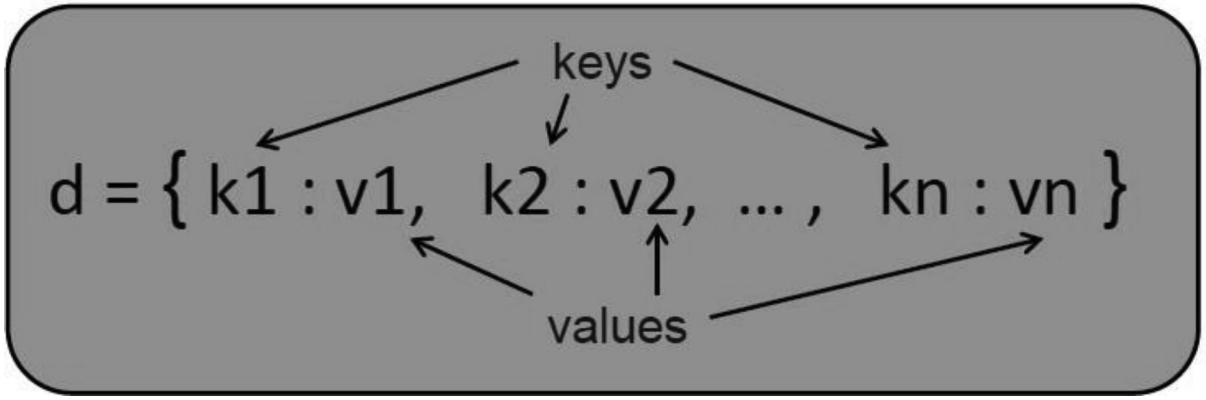
list: ordered sequence of values where we can add / remove values and edit any element. Some other languages call this an array. In Python the elements can be anything. Some languages may insist they are the same type (C++).

- More data structures : dict (dictionary),tuple and set

Dictionary

- Sequences (e.g. strings, lists) index their data by positive integers (counting from 0).
- **Dictionaries** are a data structure which indexes by other data.
  - We call the data we index **by keys** and the data that is indexed values.
  - formed by key-value pairs.

Dictionary (Continue)



- curly brackets
- keys & values connected by colons
- key-value pairs separated by commas

Dictionary – key note(Continue)

- order of entry irrelevant;
- index notation like lists but index by key instead of order;
- you can add more entries after creation; Use delete to remove element in dictionary
- The in operator can be used to ask about keys; hence can run a for-loop over dictionary.

```
>>> stock = {'oranges':55,
'apples':43, 'bananas':31}
```

```
>>> stock
{'apples': 43, 'bananas': 31,
'oranges': 55}
```

```
>>> stock['apples']
43
```

```
>>>stock['watermelon']=10
>>>del stock['apples']
```

```
>>> "apples" in stock
True
>>> "grapes" in stock
False
```

Your turn

`d={'a':100,'b':[1,2],'100':50}` Note: (Assume each instruction is execute sequentially)

`print (d['a'])`	`100`
`print (d['100'])`	`50`
`d['100'] = 40` `print (d)`	`{'a': 100, 'b': [1, 2], '100': 40}`
`print (d['b'][1])`	`2`
`d['b'][1] = 10` `print (d)`	`{'a': 100, 'b': [1, 10], '100': 40}`

Tuple

- On the surface tuples look just like lists: sequences of values separated by commas. The only difference being round brackets instead of square.

List: L = [1,2,3,4] **Tuple :** T = (1,2,3,4)

- The real difference is that we can edit the individual elements of lists but not those of tuples.
  - immutable: cannot be modified after created. E.g. strings, tuples, sets
  - mutable: can be modified after created. E.g. lists, dictionaries.

You had met tuple before

- When we pass more than one argument to a function we separate them with commas. This defines a single tuple, which is then passed to the function.
- Any function that returns more than one value does so as a tuple.

```
:
:
return (a,b,c)
```

Tuple Syntax

- Technically you can define tuples without any brackets. They are the default.
- But to avoid confusion try to use round brackets if you mean tuple.
- Note that for a tuple with one element you nee an extra comma.
- Tuples are very simple. They have hardly any methods : count and index

```
>>> t = 1,2,3
>>> type(t)
<class 'tuple'>
```

```
>>> t = (1)
>>> type(t)
<class 'int'>
>>> t = (1, )
>>> type(t)
<class 'tuple'>
```

```
>>> t=(1,2,4,3,4,3)
>>> t.count(4)
2
```

```
>>> t=(1,2,4,3,4,3)
>>> t.index(2)
1
```

Why not using list ?

- Ensuring that data cannot be changed
  - can be a good tool for error prevention;
- Tuples can actually be more efficient than lists:
  - Very memory efficient since the amount of memory needed is always known in advance
- Lists cannot be used as dictionary keys

Dictionary values can be anything (including lists & tuples). But **dictionary keys must be immutable**.
If the keys were to change Python could not use them to find the values!

Tuples or Lists?

Generally speaking: use tuples to store data that will not change and lists otherwise.

BUT:

- Tuples cannot grow; so if you do not know how much data there will be, use a list.
- Lists cannot be used as dictionary keys; so if the data is to be part of a dictionary, use a tuple.

Set

- Unordered collection of unique values
  - Order of the elements is not relevant
  - No duplication elements

```
>>> s={10,5,3,6}
>>> s
{10, 3, 5, 6}
```

```
>>> s={1,3,4,7,4,5,6,5,6}
>>> s
{1, 3, 4, 5, 6, 7}
```

- Might be assigned with duplication and with an order, but will not store in that way

Set - Methods

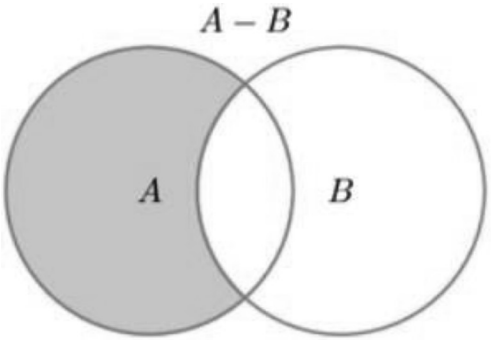
- add(x) – add a new element, x, in the set
- update([1,2,3]) – add multiple elements
- remove(x) – remove element x from the set

```
>>> A={1,2,4,5}
>>> A.add(3)
>>> A
{1, 2, 3, 4, 5}
```

```
>>> A.update([6,7,8])
>>> A
{1, 2, 3, 4, 5, 6, 7, 8}
```

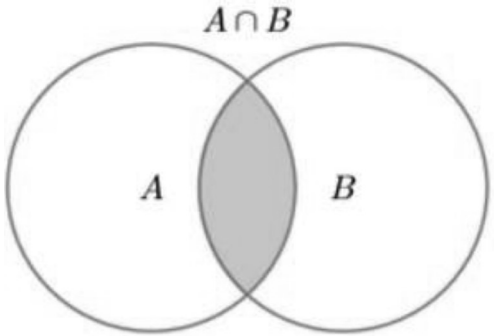
```
>>> A.remove(5)
>>> A
{1, 2, 3, 4, 6, 7, 8}
```

Set methods



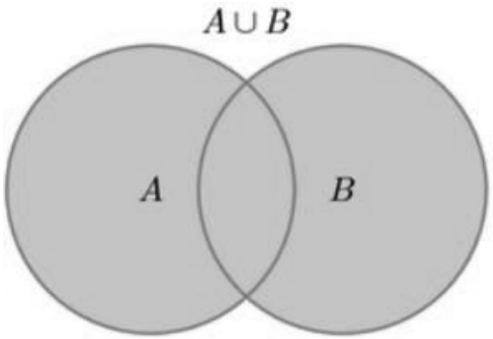
difference

```
>>> A={1,2,3,4,5}
>>> B={3,4,6,7}
>>> A.difference(B)
{1, 2, 5}
```



intersection

```
>>> A={1,2,3,4,5}
>>> B={3,4,6,7}
>>> A.intersection(B)
{3, 4}
```



union

```
>>> A={1,2,3,4,5}
>>> B={3,4,6,7}
>>> A.union(B)
{1, 2, 3, 4, 5, 6, 7}
```

Choosing the right data structure for a program can make life a lot easier!