



Dictionary, Tuple, Set



Outline

- Tuple
- Set
- Dictionary
- Zip function

Data types

Name	Type	Description	Example	mutable
String	str	A sequence of characters	"hello", "course", "covid-19", "2"	×
Integer	Int	Whole numbers	2, 4, 100, 4000	×
Float	float	Numbers containing one or more decimals	3.8, 50.9, 100.0	×
Booleans	bool	Logical value indicating TRUE or FALSE	True, False	×
List	list	Ordered sequence of objects	["hello", "world", "2021"] ["hello", 5, 100.0]	✓
Dictionary	dict	Key: value pairs	{"key1": name1, "key2": name2}	✓
Tuple	tup	Ordered immutable sequence of objects	(10,20) ("hello", "world")	×
Set	set	Unordered collection of unique objects	{2,4,6,8} {3,"hello", 50.9}	✓

Tuple

- Use **parenthesis** to create a tuple and separate the elements by comma.
- Tuple elements are **ordered**, **immutable**, and allow duplicate elements.

```
tuple1 = ('A','B','C')  
print(tuple1)  
  
('A', 'B', 'C')
```

- Tuple elements can be of any data type.

```
tuple2 = ('Python', [1,2,3,4], 50.2)  
print(tuple2)  
  
('Python', [1, 2, 3, 4], 50.2)
```

Tuple - accessing elements

- Indexing and slicing work the same way for tuples as they do for lists.

```
tuple1 = ('A', 'B', 'C')
```

'A'	'B'	'C'	→ element
0	1	2	→ Index

```
tuple1[0]
```

Get the first element

```
'A'
```

```
tuple1[0:2]
```

Get the first two elements

```
('A', 'B')
```

```
tuple1[-1]
```

Get the last element

```
'C'
```

Tuple - compare with list

- Tuples are also iterable objects.

```
for i in tuple1:  
    print(i)
```

A
B
C

- The main difference between tuples and lists is that **tuples are immutable**, and **lists are mutable**.

```
tuple2[0] = "new values"
```


```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-70-1a0bfbbb6298> in <module>  
----> 1 tuple2[0] = "new values"  
  
TypeError: 'tuple' object does not support item assignment
```

- You cannot add an element to a tuple.
- You cannot remove an element from a tuple.
- You cannot replace an element in a tuple.
- You cannot sort a tuple.

- A tuple is allocated in a fixed-sized block of memory because it doesn't require extra space to store new data.
- A list is allocated in a variable-sized block of memory.


Tuple - packing and unpacking

- Packing: Assign multiple values into a tuple.



```
tuple_info = ('Anna', 'anna@bi.no', 20, 2021)
```

- Unpacking: Assign a tuple into multiple variables.



```
name, email, age, year = tuple_info
```

```
print(name, email, age, year, sep = "\n" )
```

```
Anna
anna@bi.no
20
2021
```

Exercise

(A.1) Create a tuple named `company_info` that contains the following elements. Print out the tuple.

- Microsoft
- Software
- Bill Gates
- 182268

(A.2) Print out the last two elements in `company_info`.

(A.3) Assign the values in `company_info` into four variables: `company_name`, `industry`, `founder`, `employees`. Print out these variables.

Set

- Use **curly brackets** to create a set and separate the elements by comma.
- Set elements are **unordered**, **mutable**, and only **unique** elements are allowed.

```
set1 = {1,2,3}  
print(set1)
```

```
{1, 2, 3}
```

- Set elements can be of any **immutable** data type.

```
set2 = {'abc', (1,2,3), 50.2}  
print(set2)
```

```
{50.2, (1, 2, 3), 'abc'}
```

Set - accessing elements

- You can't use the index to access elements of the set, but you can use a **for loop** to iterate through all the elements of the set.

```
set1[0]
```

```
-----  
TypeError                                 Traceback (most recent call last)  
<ipython-input-63-c38563f1af7a> in <module>  
----> 1 set1[0]  
  
TypeError: 'set' object is not subscriptable
```

```
# use a for loop  
for i in set1:  
    print(i)
```

A set is an iterable object.

```
1  
2  
3
```

Set - compare with list

- The main difference between sets and lists is that a list can contain **duplicate** elements, while a set only contains **unique** elements.
- Convert a list into a set.

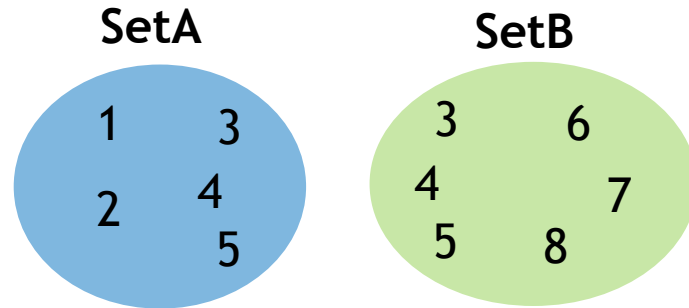
```
mylist = ['A','B','B','A','C','C','A','B','C','B','A','C','A','B']
```

```
myset = set(mylist)  
print(myset)
```

```
{'B', 'A', 'C'}
```

Set - methods

- Sets support mathematical set operations like **union**, **intersection**, **difference**.

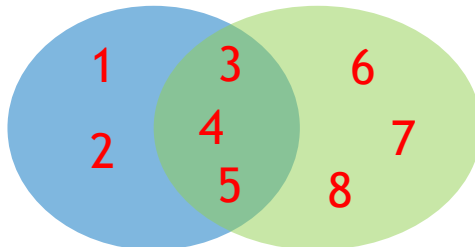


```
setA = {1, 2, 3, 4, 5}
setB = {3, 4, 5, 6, 7, 8}
```

Union

```
setA.union(setB)
```

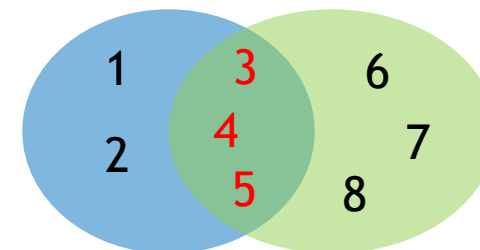
{1, 2, 3, 4, 5, 6, 7, 8}



Intersection

```
setA.intersection(setB)
```

{3, 4, 5}

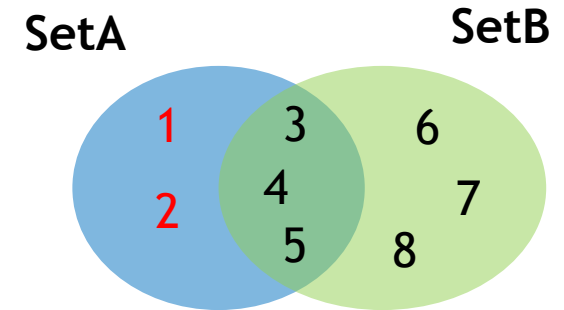


Set - methods

- The elements in SetA that are not in SetB

```
setA.difference(setB)
```

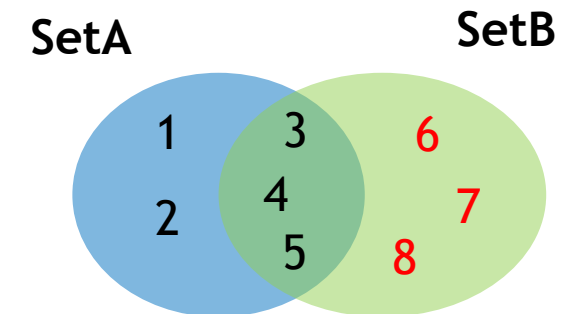
```
{1, 2}
```



- The elements in SetB that are not in setA

```
setB.difference(setA)
```

```
{6, 7, 8}
```



Exercise

Exercise.B

(B.1) Student A and student B have chosen the following courses for the next semester. Store their chosen courses in two sets named `set_a` and `set_b` . Print out these two sets.

- student A: ELE0505, ELE3400, ELE1295, ELE7163, ELE9145
- student B: ELE0099, ELE7163, ELE0705, ELE3400, ELE6027

(B.2) Which courses will they take together?

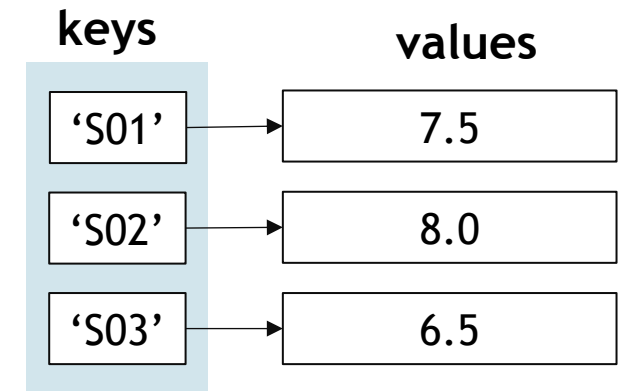
Dictionary

- A dictionary is an **unordered** map-like structure. Unlike lists that store elements in an ordered sequence, dictionaries use **key-value pairs** instead.
- To create a dictionary,
 - Use **curly brackets** `{}` to enclose the key-value pair
 - Use **colons** to separate keys and values
 - Use **comma** to separate key-value pairs

`{'key1': value1, 'key2':value2, 'key3':value3}`

```
dict1 = {'S01':7.5, 'S02':8.0, 'S03':6.5}  
print(dict1)
```

```
{'S01': 7.5, 'S02': 8.0, 'S03': 6.5}
```

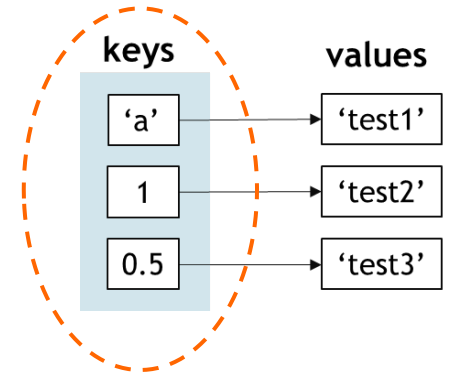


Use student ID as key and score as value.

Dictionary

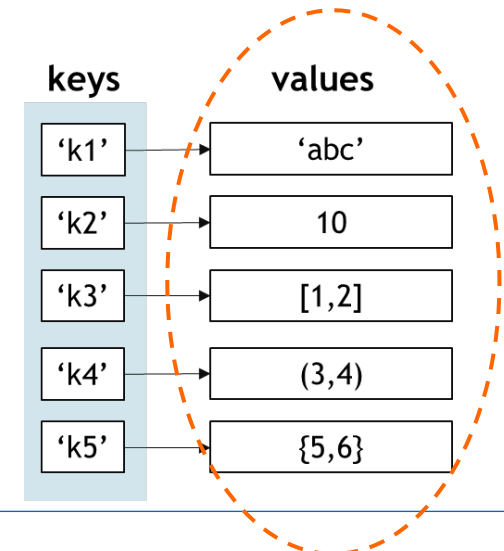
- A dictionary key can be any immutable data type, such as string, integer, or float.

```
dict2 = {'a':'test1', 1:'test2', 0.5: 'test3'}  
print(dict2)  
  
{'a': 'test1', 1: 'test2', 0.5: 'test3'}
```



- A dictionary values can be of any data type.

```
dict3 = {'k1':'abc', 'k2':10, 'k3':[1,2], 'k4':(3,4), 'k5':{5,6}}  
print(dict3)  
  
{'k1': 'abc', 'k2': 10, 'k3': [1, 2], 'k4': (3, 4), 'k5': {5, 6}}
```



Dictionary - access value by key

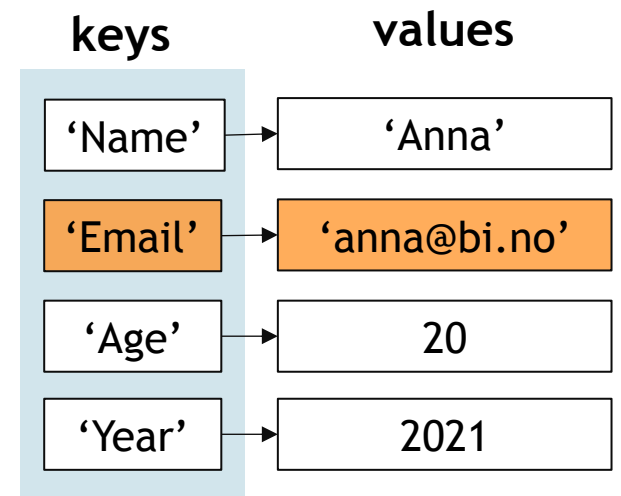
- Keys are **unique** within a dictionary and can not be duplicated inside a dictionary.
- A dictionary allows user to get a value by specifying a key without knowing an index location.

```
S01_info = {'Name':'Anna', 'Email':'anna@bi.no', 'Age':20, 'Year':2021}  
print(S01_info)
```

```
{'Name': 'Anna', 'Email': 'anna@bi.no', 'Age': 20, 'Year': 2021}
```

```
S01_info['Email']
```

```
'anna@bi.no'
```



Dictionary - operations

- A dictionary is a mutable object.

```
dict1 = {'S01':7.5, 'S02':8.0, 'S03':6.5}  
print(dict1)
```

```
{'S01': 7.5, 'S02': 8.0, 'S03': 6.5}
```

```
# Add a key-value pair  
dict1['S04'] = 7.0  
print(dict1)
```

```
{'S01': 7.5, 'S02': 8.0, 'S03': 6.5, 'S04': 7.0}
```

```
# Delete a key-value pair  
del dict1['S01']  
print(dict1)
```

```
{'S02': 8.0, 'S03': 6.5, 'S04': 7.0}
```

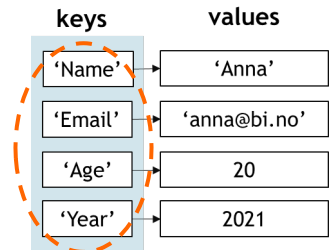
```
# Modify the value  
dict1['S02'] = 9.0  
print(dict1)
```

```
{'S02': 9.0, 'S03': 6.5, 'S04': 7.0}
```

Dictionary - methods

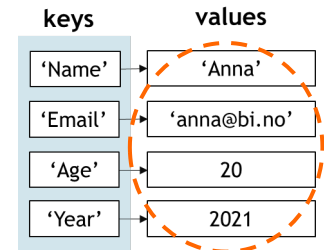
- Get keys

```
S01_info.keys()  
dict_keys(['Name', 'Email', 'Age', 'Year'])
```



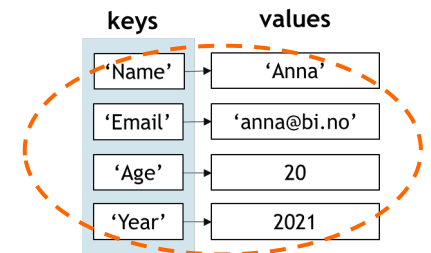
- Get values

```
S01_info.values()  
dict_values(['Anna', 'anna@bi.no', 20, 2021])
```



- Get key-value pairs

```
S01_info.items()  
dict_items([('Name', 'Anna'), ('Email', 'anna@bi.no'), ('Age', 20), ('Year', 2021)])
```



Dictionary - methods

- Iterate over the keys/values/items in a dictionary using a `for` loop.

```
# Print all keys in the dictionary
for key in S01_info.keys():
    print(key)
```

```
Name
Email
Age
Year
```

```
# Print all values in the dictionary
for value in S01_info.values():
    print(value)
```

```
Anna
anna@bi.no
20
2021
```

```
# Print all items in the dictionary
for items in S01_info.items():
    print(items)
```

```
('Name', 'Anna')
('Email', 'anna@bi.no')
('Age', 20)
('Year', 2021)
```

```
# Print all keys and values in the dictionary
for key, value in S01_info.items():
    print(f"key: {key}; value: {value}")
```

```
key: Name; value: Anna
key: Email; value: anna@bi.no
key: Age; value: 20
key: Year; value: 2021
```

Exercise

(C.1) Define a dictionary based on the following table. Print out the dictionary.

- **Key:** Student_ID.
- **Value:** A list of three scores.

Student_ID	Score for Test-1	Score for Test-2	Score for Test-3
S01	70	85	75
S02	65	75	80
S03	80	70	60

(C.2) Print out the score list of student S02 .

(C.3) Print second test scores for all students.

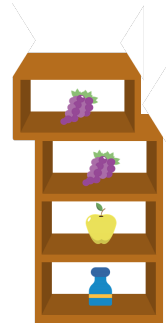
Expected result:

S01: 85

S02: 75

S03: 70

Different types of containers



List

- Ordered
- Mutable



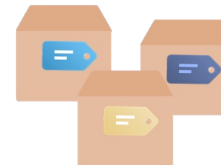
Tuple

- Ordered
- Immutable



Set

- Unordered
- Mutable
- Unique elements



Dictionary

- Unordered
- Mutable
- Key-value pairs

Different types of containers

Data type	Ordered	Mutable	Elements	Example
List	✓	✓	<ul style="list-style-type: none">Any data type	[1, 2, 3]
Tuple	✓	✗	<ul style="list-style-type: none">Any data type	(1, 2, 3)
Set	✗	✓	<ul style="list-style-type: none">Any immutable data typeUnique elements	{1, 2, 3}
Dictionary	✗	✓	<ul style="list-style-type: none">Key: Any immutable data typeValue: Any data type	{"a":1, "b":2, "c":3}

Zip - use zip function to create a set of tuples

- Use function `zip()` to create a **zip object** containing a set of tuples.
- Example: Pair student IDs with their scores.

```
ID_list = ['S01', 'S02', 'S03', 'S04', 'S05']  
score_list = [7.0, 6.5, 9.0, 8.0, 7.5]
```

```
zipped = zip(ID_list, score_list)  
print(type(zipped))
```

```
<class 'zip'>
```

The `zip()` function returns a zip object.

ID_list	score_list
S01	7.0
S02	6.5
S03	9.0
S04	8.0
S05	7.5



('S01', 7.0)
('S02', 6.5)
('S03', 9.0)
('S04', 8.0)
('S05', 7.5)

Zip object

Zip - iterator



- A zip object is an iterator.
 - Iterators can only be iterated over **once**. After iterating over all items, it is just an empty collection.

```
for item in zipped:  
    print(item)
```

```
('S01', 7.0)  
( 'S02', 6.5)  
( 'S03', 9.0)  
( 'S04', 8.0)  
( 'S05', 7.5)
```

```
for item in zipped:  
    print(item)
```

If you try to iterate over all items again, you won't see anything.

Zip - create a list/dictionary from the zip object

S01	7.0
S02	6.5
S03	9.0
S04	8.0
S05	7.5

List

7.0
6.5
9.0
8.0
7.5

List

zip()

('S01', 7.0)
('S02', 5.5)
('S03', 9.0)
('S04', 5.0)
('S05', 7.5)

Zip object

list()

('S01', 7.0)
('S02', 5.5)
('S03', 9.0)
('S04', 5.0)
('S05', 7.5)

List

dict()

keys	values
'S01'	7.5
'S02'	6.5
'S03'	9.0
'S04'	8.0
'S05'	7.5

Dictionary

Zip - create a list

- Convert the zip object to a list using the `list()` function.

```
ID_list = ['S01', 'S02', 'S03', 'S04', 'S05']  
score_list = [7.0, 6.5, 9.0, 8.0, 7.5]
```

```
# step1: Create a zip object  
zipped = zip(ID_list, score_list)
```

```
# step2: Convert zip object to list  
list1 = list(zipped)  
print(list1)
```

```
[('S01', 7.0), ('S02', 6.5), ('S03', 9.0), ('S04', 8.0), ('S05', 7.5)]
```

Zip - create a dictionary

- Convert the zip object to a dictionary using the `dict()` function.
 - Use the elements in `ID_list` as **keys** and elements in `score_list` as **values**.

```
ID_list = ['S01', 'S02', 'S03', 'S04', 'S05']  
score_list = [7.0, 6.5, 9.0, 8.0, 7.5]
```

```
# step1: Create a zip object  
zipped = zip(ID_list, score_list)
```

```
# step2: Convert zip object to dictionary  
dict1 = dict(zipped)  
print(dict1)
```

```
{'S01': 7.0, 'S02': 6.5, 'S03': 9.0, 'S04': 8.0, 'S05': 7.5}
```

Exercise

Exercise.D

(D.1) Product names and their prices are stored in the following lists. Create a zip object using these lists.

```
product_list = ["A", "B", "C", "D", "E"]  
price_list = [100, 50, 20, 80, 90]
```

(D.2) Convert the zip object obtained in (D.1) to a list. Print out the list.

(D.3) Print out all products priced below 60.

Expected result:

('B', 50)

('C', 20)