# 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”]<br>[“hello, 5, 100.0] | ✓ |
| Dictionary | dict | Key: value pairs | {“key1”: name1, “key2”:name2} | ✓ |
| Tuple | tup | Ordered immutable sequence of objects | (10,20)<br>(“hello”, “world”) | × |
| Set | set | Unordered collection of unique objects | {2,4,6,8}<br>{3,”hello”, 50.9} | ✓ |

# Different types of containers

List, tuple, set, and dictionary are fundamental data structures in Python used to store and manage collections of data.

| Data type | Ordered | Mutable | Elements | Example |
|---|---|---|---|---|
| List | ✓ | ✓ | • Any data type | [1, 2, 3] |
| Tuple | ✓ | ✗ | • Any data type | (1, 2, 3) |
| Set | ✗ | ✓ | • Any immutable data type<br>• Unique elements | {1, 2, 3} |
| Dictionary | ✗ (≤ Python 3.6)<br>✓ (≥ Python 3.7) | ✓ | • Key: Any immutable data type<br>• Value: Any data type | {“a”:1, “b”:2, “c”:3} |

Mutable means that it can be changed after it is created, such as adding elements or removing elements.

# Tuple - creation

- Use parenthesis to create a tuple and separate the elements by comma.

```
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

- Tuple elements are ordered, so 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

- The main difference between tuples and lists is that tuples are immutable, and lists are mutable.
  - Tuples are typically used to store collections of elements that should not be changed once created, such as coordinates, dates, or records.

```
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.

- Tuples are also iterable objects.

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

```
A
B
C
```

Tuples are more memory efficient than lists.
- 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: Extract values from a tuple and assign them to multiple variables.

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

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

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

You can use packing and unpacking with both tuple and list data types.

# Exercise

## Exercise.A

(A.1) Create a tuple named `timestamp` that contains the following three elements: 5, "Oct", 2023. Print out the tuple.

(A.2) Print out the last element in `timestamp`.

(A.3) Assign the values from `timestamp` to three variables named: day, month, year. Print out these variables.

# Set - creation

- Use curly brackets to create a set and separate the elements by comma.

```
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'}
```

- Sets are mutable, and you can add or remove elements after they are created.



- https://www.w3schools.com/python/python_sets_methods.asp

# Set – accessing elements

- The elements in the set are unordered, so you cannot 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.

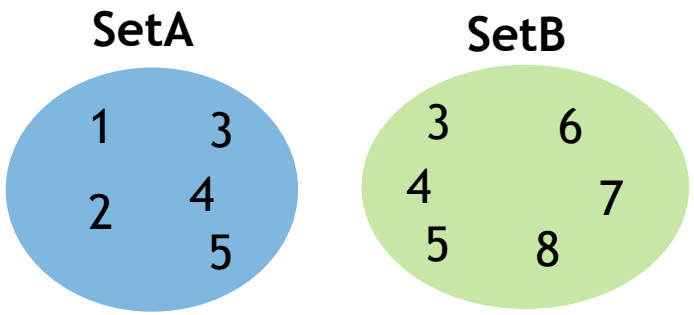
```
mylist = ['A','B','B','A','C','C','A','B','C','B','A','C','A','B']
```

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

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

# Set - methods

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

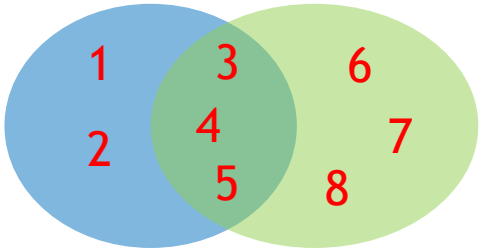



```
setA = {1,2,3,4,5}
setB = {3,4,5,6,7,8}
```

**Union**

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

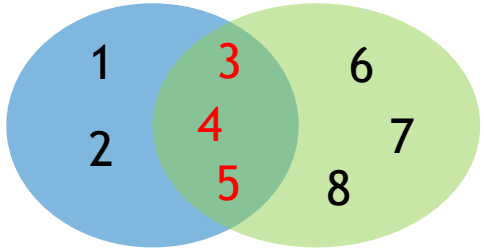
```
{1, 2, 3, 4, 5, 6, 7, 8}
```




**Intersection**

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

```
{3, 4, 5}
```

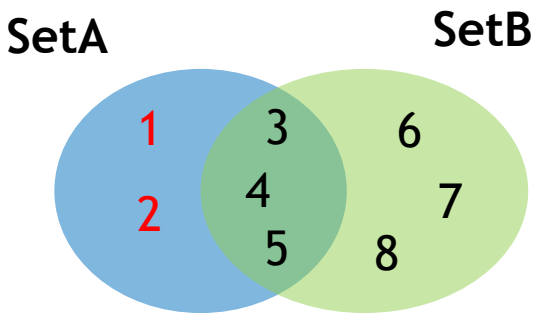


# Set - methods

- The elements in SetA that are not in SetB

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

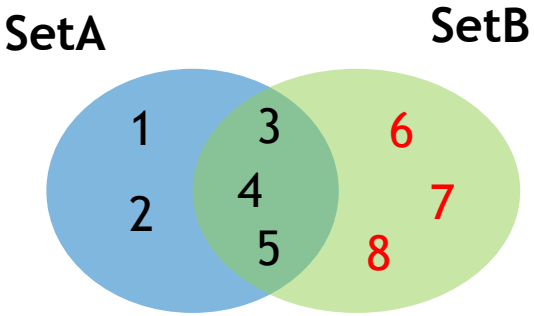
```
{1, 2}
```




- 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

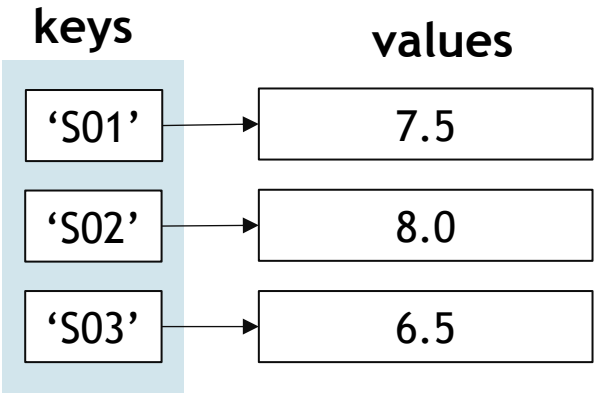
- A dictionary is a map-like structure that allows you to establish associations between keys and their corresponding values.

- To create a dictionary,
  - Use curly brackets {} to enclose the key-value pair
  - Use colons to separate key and value
  - 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.

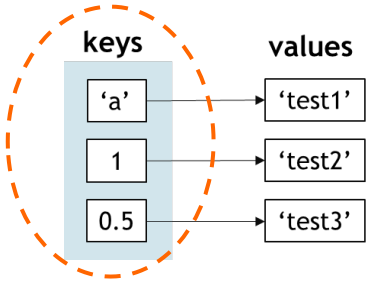
In Python 3.6 and earlier, dictionaries are unordered. Starting with Python 3.7, dictionaries are ordered, which means the order in which items are added to the dictionary is preserved.

# Dictionary

- A dictionary key can be <u>any immutable data type</u>, such as string, integer, or float. Keys are unique within a dictionary and can not be duplicated inside a dictionary.

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

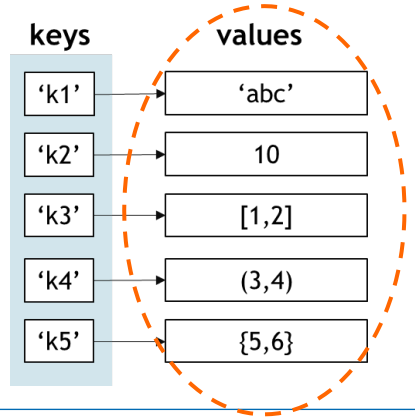
```
{'a': 'test1', 1: 'test2', 0.5: 'test3'}
```




- A dictionary values can be of <u>any data type</u>.

```
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

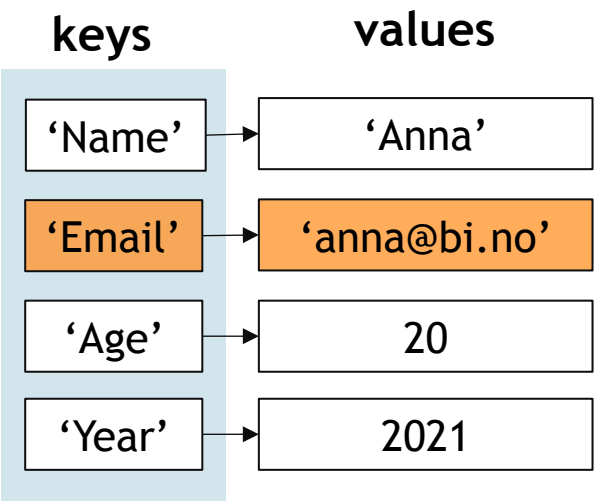
- A dictionary allows user to get a value by specifying a key without knowing an index location.

```
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}
```

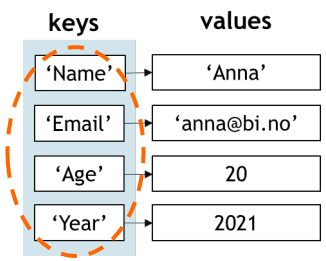
https://www.w3schools.com/python/python_ref_dictionary.asp

# Dictionary - methods

- Get keys

```
S01_info.keys()
```

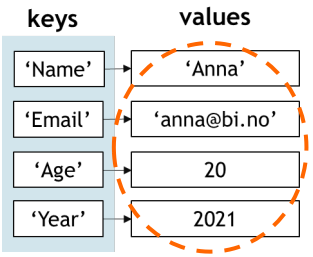
```
dict_keys(['Name', 'Email', 'Age', 'Year'])
```




- Get values

```
S01_info.values()
```

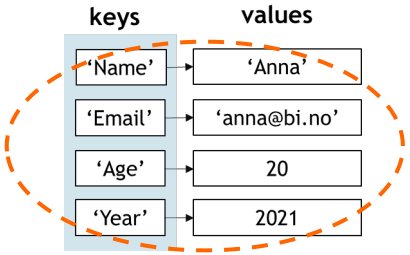
```
dict_values(['Anna', 'anna@bi.no', 20, 2021])
```




- 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`.

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

```
Name
Email
Age
Year
```

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

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

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

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

```python
# 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

# 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))
```

The zip() function returns a zip object.

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

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

zip()

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

**Zip object**

https://www.geeksforgeeks.org/zip-in-python/

# 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.

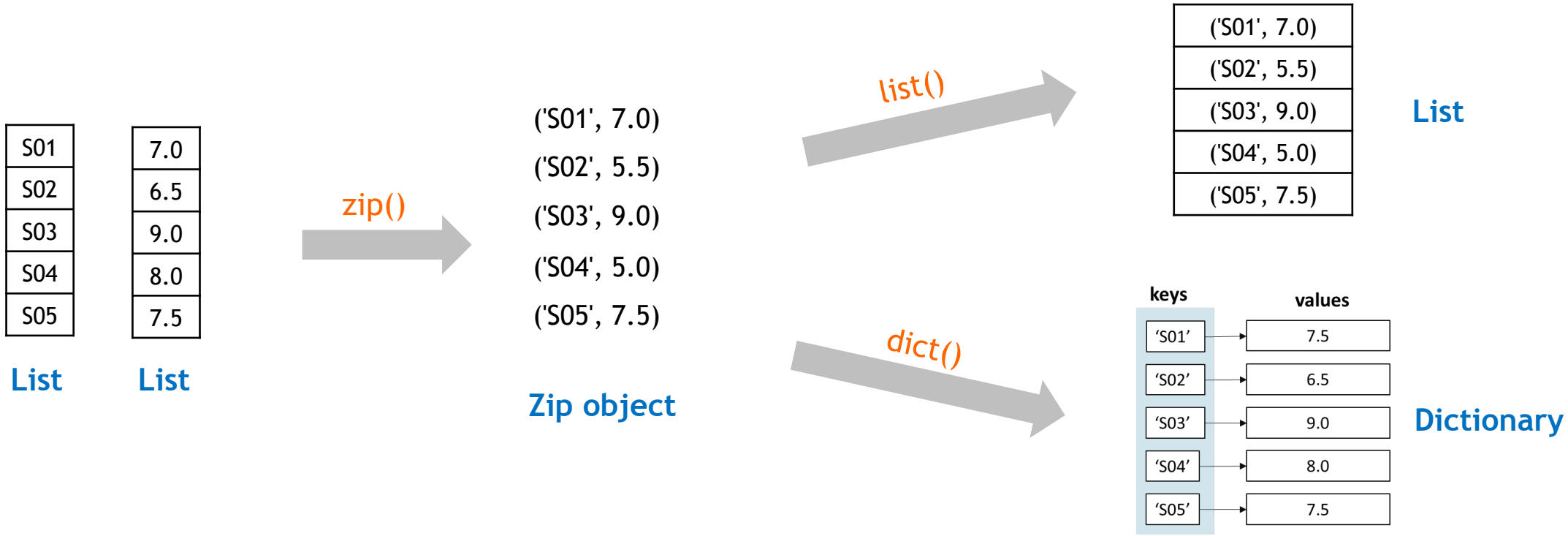
```
for item in zipped:
    print(item)
```

```
('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 a zip object



# Zip – create a list

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

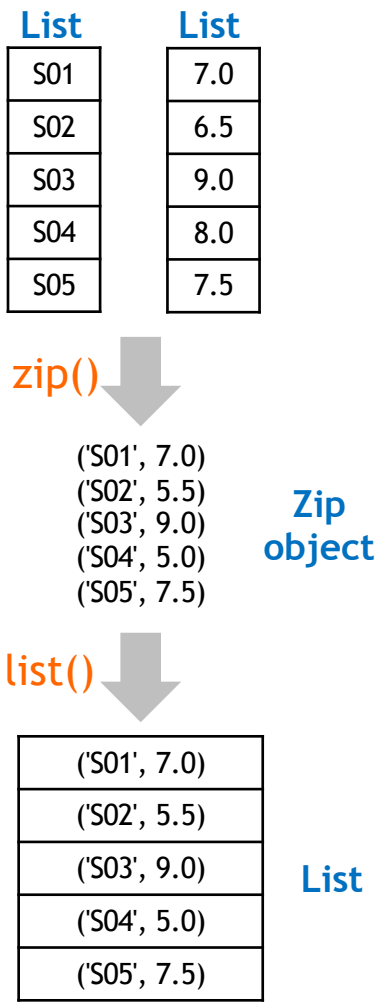
```
ID_list = ['S01', 'S02', 'S03', 'S04', 'S05']
score_list = [7.0, 6.5, 9.0, 8.0, 7.5]
```

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

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

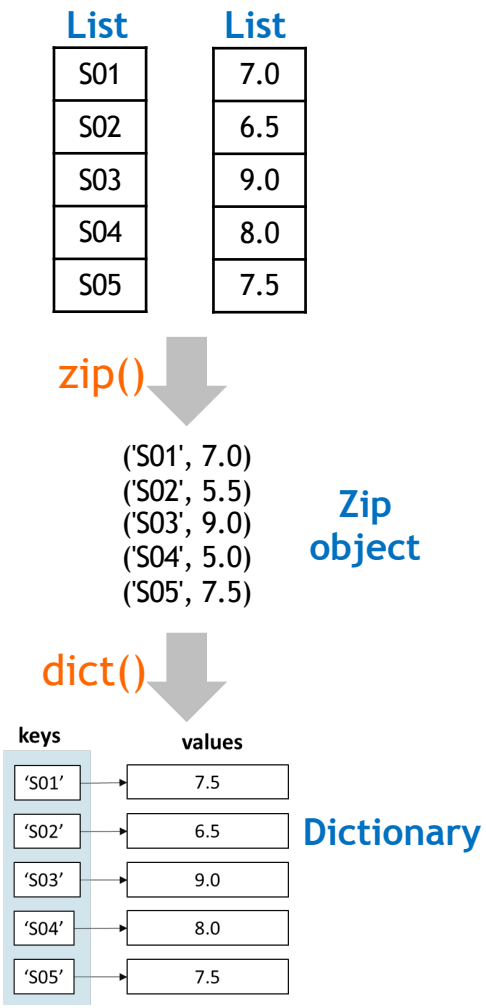
```
print(list1)
```

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



# Zip – create a dictionary

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




```
# 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}
```

# Different ways to create objects

More common when you need to convert one data type to another.

| Data type | Define directly | Use build-in functions |
|---|---|---|
| String | `s1 = “hello”` | `s1 = str(“hello”); s2 = str(100)` |
| Integer | `x1 = 2` | `x1 = int(2);  x2 = int(2.0)` |
| Float | `y1 = 3.8` | `y1 = float(3.8); y2 = float(3)` |
| Booleans | `z1 = True` | `z1 = bool(“True”) ; z2 = bool(1)` |
| List | `l1 = []; l2 = [1, 2, 3]` | `l1= list() ; l2 = list((1,2,3))` |
| Dictionary | `d1 = {}; d2 = {“k1”: 10, “k2”:20}` | `d1 = dict(); d2 = dict(zip(['k1', 'k2'], [10, 20]))` |
| Tuple | `t1 = (1, 2, 3)` | `t1 = tuple((1,2,3)); t2 = tuple([1,2,3])` |
| Set | `s1 = {1, 2, 3}` | `s1 = set([1,2,3])` |

# 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)