Dictionary, Tuple, Set

Agenda

- Tuple
- Set
- Dictionary
- Zip
- List comprehension

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	×
Floating point	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	tuple	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')
```

- Tuple elements can be of any data type.

```
tuple2 = ('some values',[1,2,3,4], 50.2)
tuple2
```

```
('some values', [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')
```

```
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 tuple.
- You cannot remove an element in tuple.
- You cannot replace an element in 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 a fixed-sized block and a variable-sized block of memory .

Tuple – packing and unpacking

- Packing: Assign multiple values into a tuple.

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

- Unpacking: Assign a tuple into multiple variables.

```
name, email, age, year = student_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.

- Mircrosoft
- 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}
set1
```

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

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

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

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

- The elements of the set are immutable, but the set itself is mutable, that is, it can be changed.
- https://www.w3schools.com/python/python_sets_methods.asp

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']
```

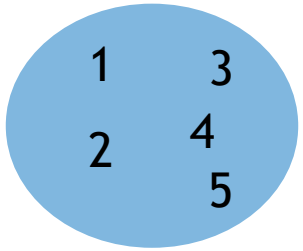
```
myset = set(mylist)
myset
```

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

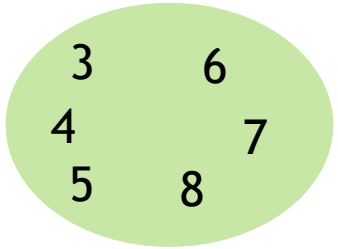
Set - methods

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

SetA



SetB

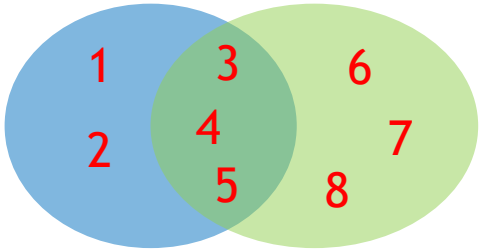



```
setA = {1,2,3,4,5}
setB = {3,4,5,6,7,8}
```

Union

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

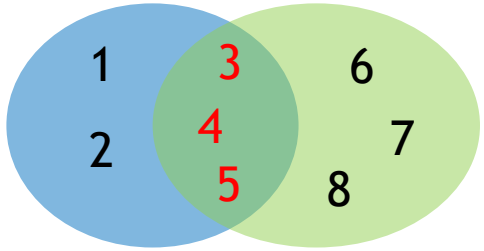
```
{1, 2, 3, 4, 5, 6, 7, 8}
```



Intersection

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

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

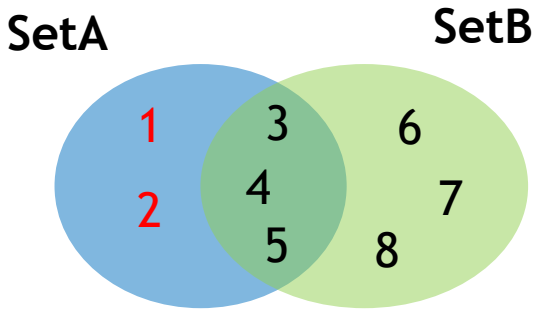


Set - methods

- The elements in SetA that are not in SetB

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

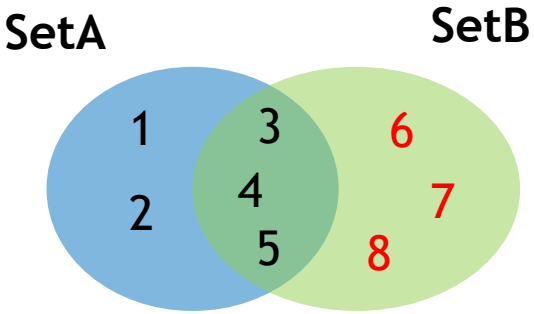
```
{1, 2}
```



- The elements in SetB that are not in setA

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

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



Exercise

(B.1) Student A and student B choose the following courses for the next semester. Print out all the courses they will take together

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

(B.2) Get all the unique letters from the following list and store them in a new variable `myset`.

```
mylist = ['a','c','e','b','c','b','a','e','d','e','d','a','e','d','b','c','d']
```

(B.3) Use `mylist` and `myset` in (B.2). Calculate the frequency of each letter.

Expected result:

b:3

d:4

e:4

...

Dictionary

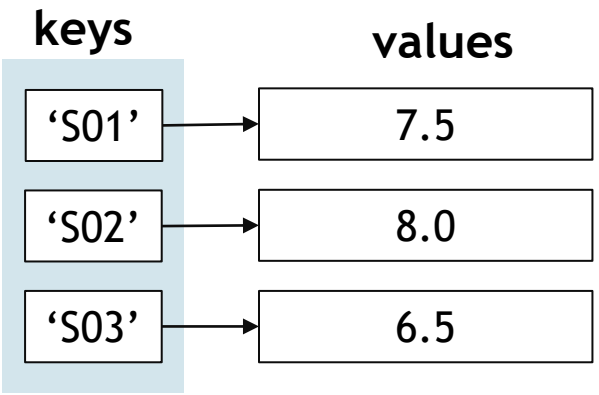
- A dictionary is an unordered map-like structure. Unlike lists that store elements in an ordered sequence, dictionaries use key-value pairs instead.

- 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}
dict1
```

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



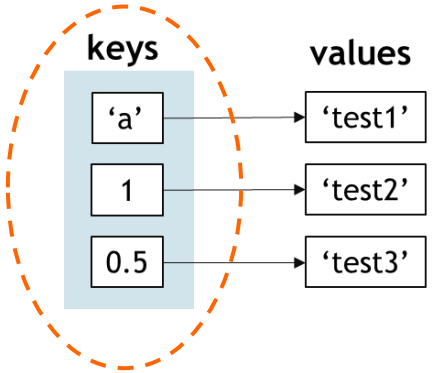
Use student ID as key and score as value.

Dictionary

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

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

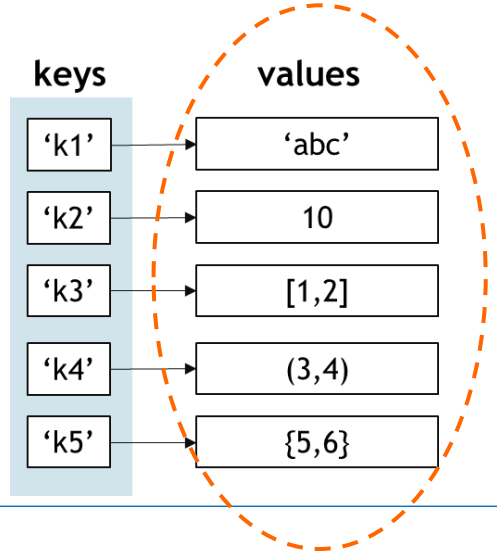
```
{'a': 'test1', 1: 'test2', 0.5: 'test3'}
```



- A dictionary values can be of any data type.

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

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



Dictionary - access value by key

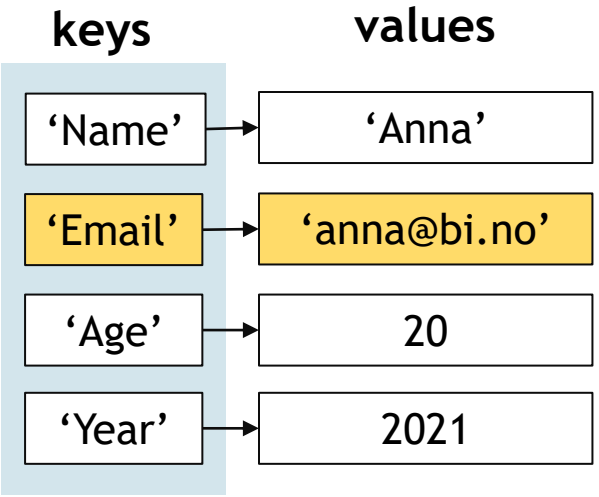
- Keys are unique within a dictionary and can not be duplicated inside a dictionary.
- A dictionary allows user to get a value by specifying a key without knowing an index location.

```
S01_info = {'Name':'Anna', 'Email':'anna@bi.no', 'Age':20, 'Year':2021}
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}
dict1
```

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

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

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

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

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

```
# Modify the value
dict1['S02'] = 9.0
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`.

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

```
Name
Email
Age
Year
```

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

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

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

```
The key is 'Name' and the value is 'Anna.'
The key is 'Email' and the value is 'anna@bi.no.'
The key is 'Age' and the value is '20.'
The key is 'Year' and the value is '2021.'
```

Dictionary – methods

- Create a dictionary by a list of keys and initializing all with the same value.

```
id_list = ["S01", "S02", "S03", "S04", "S05"]
```

```
score_dict =  dict.fromkeys(id_list,0)
score_dict
```

Initial value

A list of keys

```
{'S01': 0, 'S02': 0, 'S03': 0, 'S04': 0, 'S05': 0}
```

- The None keyword is used to define a null variable or an object. In Python, **None** keyword is an object, and it is a data type of the class NoneType.

```
info_list = ['Name', 'Email', 'Age', 'Year']
```

```
info_dict = dict.fromkeys(info_list)
info_dict
```

```
{'Name': None, 'Email': None, 'Age': None, 'Year': None}
```

https://www.pythontutorial.net/advanced-python/python-none/

Exercise

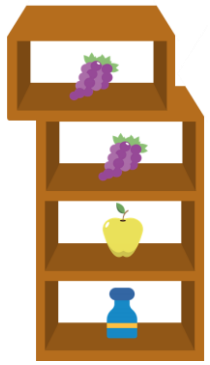
(C.1) Create a dictionary with `studentID` as the key and `score list` as the value. Print out the dictionary.

```
#studentID: S01, S02, S03
S01_score = [7.0, 8.5, 7.5]
S02_score = [6.5, 7.5, 8.0]
S03_score = [8.0, 7.0, 7.0]
```

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

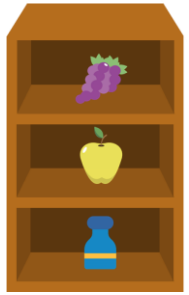
Different types of containers

List



- Ordered
- Mutable

Tuple

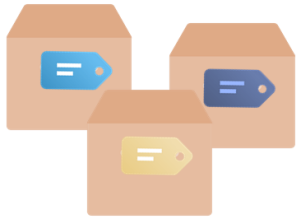


- Ordered
- Immutable

Set

- Unordered
- Mutable
- Unique elements

Dictionary



- Unordered
- Mutable
- Key-value pairs

Zip – use zip function to create a set of tuples

- Use function `zip()` to create a **zip object** containing a set of tuples.

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

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

The zip() function returns a zip object.

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

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

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

A zip object is an iterator.

- The nature of iterator is that once it's done iterating the data - it points to an empty collection

- Lists, tuples, dictionaries, and sets are all iterable objects. All these objects have a method to get an iterator.
- https://docs.python.org/3.8/library/functions.html#zip

Zip – create a list from the zip object

- We want to 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]
```

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

```
# step2: Convert zip object to list
mylist = list(myzip)
mylist
```

```
[('S01', 5), ('S02', 5), ('S03', 6), ('S04', 8), ('S05', 4)]
```

Zip – create a dictionary from the zip object

- We want to use elements in **`ID_list`** as keys and elements in **`score_list`** as value when creating a dictionary.

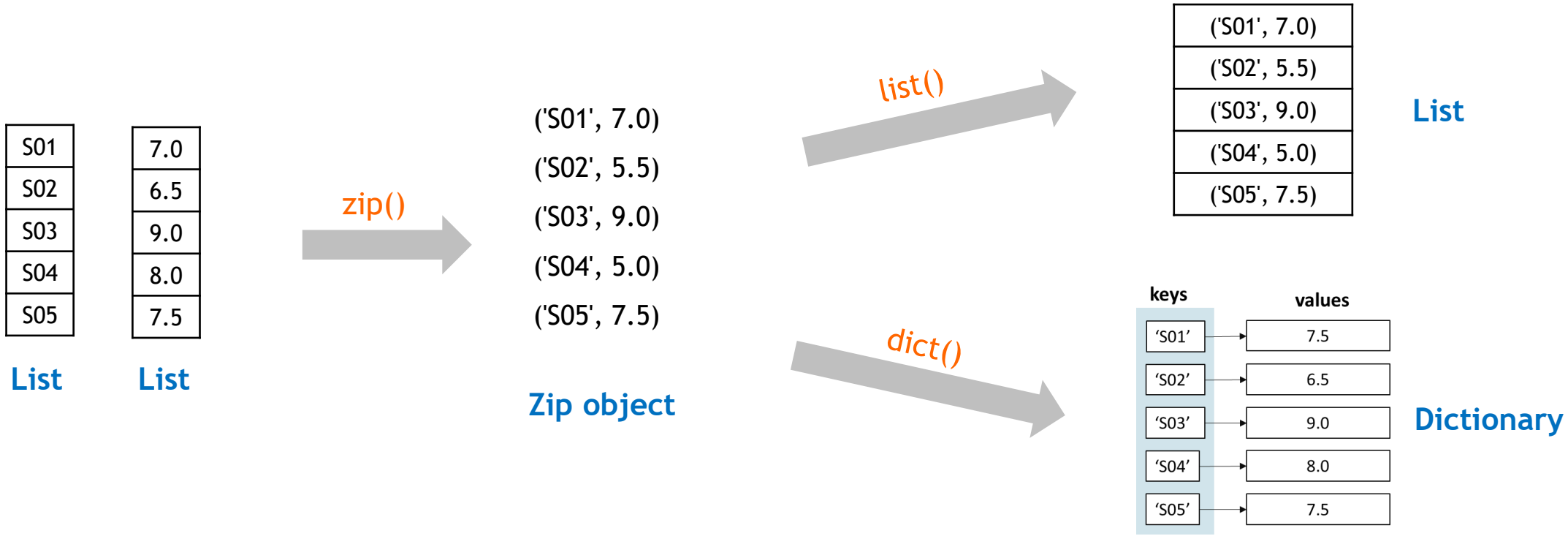
```
ID_list = ['S01', 'S02', 'S03', 'S04', 'S05']
score_list = [7.0, 6.5, 9.0, 8.0, 7.5]
```

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

```
# step2: Convert zip object to dictionary
mydict = dict(myzip)
mydict
```

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

Zip - create a list/dictionary from the zip object



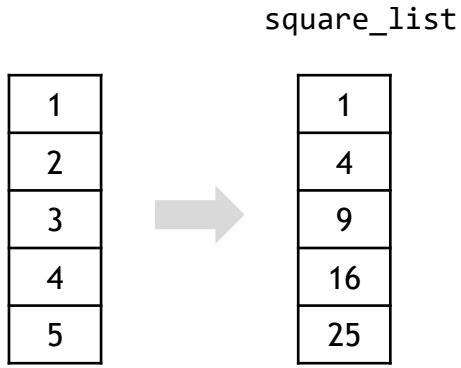
List comprehension

- List comprehension uses a shorter syntax to create a new list.

```
square_list = []
for i in range(1,6):
    square_list.append(i**2)

print(square_list)
```

```
[1, 4, 9, 16, 25]
```




```
square_list = [i**2 for i in range(1,6)]
print(square_list)
```

```
[1, 4, 9, 16, 25]
```

List comprehension

- Select some elements in the list based on conditions.

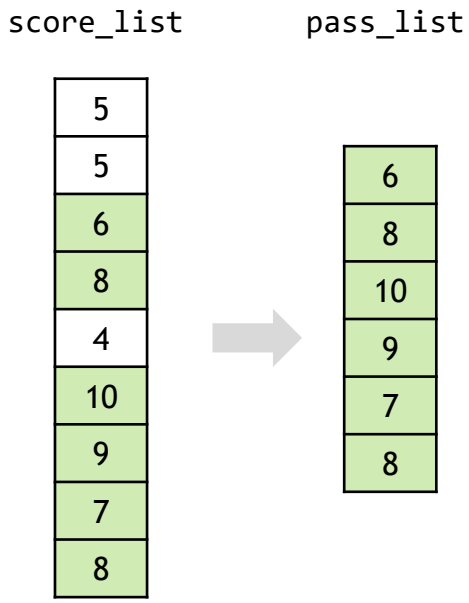
```
score_list = [5,5,6,8,4,10,9,7,8]
pass_list = []
for s in score_list:
    if s >= 6:
        pass_list.append(s)

print(pass_list)
```

```
[6, 8, 10, 9, 7, 8]
```

```
pass_list = [s for s in score_list if s >=6]
pass_list
```

```
[6, 8, 10, 9, 7, 8]
```



https://docs.python.org/3.8/tutorial/datastructures.html#list-comprehensions

List comprehension

- Select some tuples in the list based on conditions.

```
# create a list of tuples
ID_list = ['S01', 'S02', 'S03', 'S04', 'S05']
score_list = [7.0, 5.5, 9.0, 5.0, 7.5]
mylist = list(zip(ID_list, score_list))
mylist
```

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

```
# use list comprehension to create a new list
pass_list = [student for student in mylist if student[1] >=6]
pass_list
```

```
[('S01', 7.0), ('S03', 9.0), ('S05', 7.5)]
```

Exercise

(D.1) Use the following lists to create a zip object.

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

(D.2) Use the zip object obtained in (D.1) to create a list of tuples. Print out the list.

(D.3) Select products with a price of less than 60 and store them in a new list.