Lab 1: Python Basic (Part B)

Section 3. More on Python data structures: Tuples, Lists, Dictionaries

More about Tuples

Review: A tuple is an **immutable** sequence type that contains an ordered collection of items. After you create a tuple, you usually cannot change it anymore. Previously, we learned that a tuple can be created by including the sequence of data items inside ( ). In fact, a tuple can be created even without the ( ).

Example 3.1: Try the following statements to create a few tuples and check them:

```
tuple1 = 1, 3, 5, 7
print(tuple1)
tuple2 = (1, 3, 5, 7)
print(tuple2)

#Are tuple1 and tuple2 the same? "==" is a comparison operator
print(tuple1 == tuple2)

tuple3 = ("Jack", "John", "Alice")
print(tuple3)
tuple4 = (1, "Jack", 2, "John")
print(tuple4)
```

Negative indexing: Recall that tuple[0] refers to the first item, tuple[8] refers to the 9-th item, etc. More interestingly, Python also supports **negative indexing**: e.g., tuple[-1] refers to the last item, tuple[-2] refers to the 2nd last item, etc. This is convenient if you want to access the data in the reverse order.

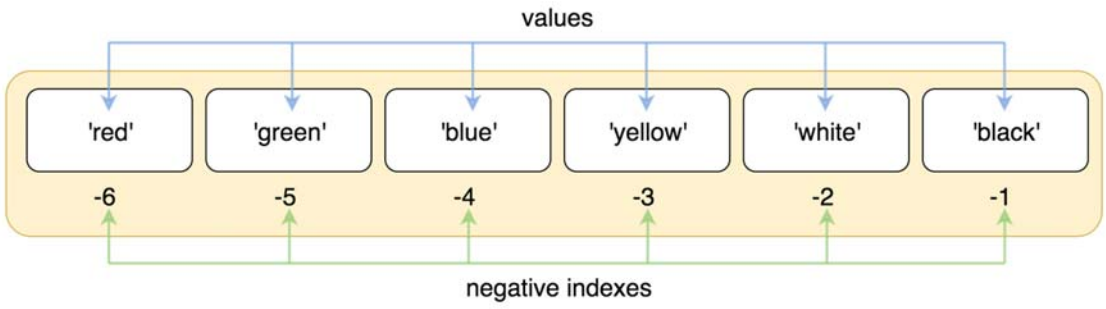

Fig. 1 An illustration of negative indexing

Example 3.2: Try the following statements to access the items in a tuple through indexing

```
print(tuple4[0])
print(tuple4[1])
print(tuple4[2])
print(tuple4[3])
print(tuple4[-1])
```

```
print(tuple4[-2])
print(tuple4[-3])
print(tuple4[-4])
```

More about Lists

Review: A list is a **mutable** sequence type that can hold any number of ordered data items with any data type. Due to this flexibility, it is one of the most commonly used data structures in Python.

A list can be generated by providing a list of data items enclosed by **square brackets []**, separated by commas. A list can also be generated from an existing non-list sequence (such as a range or a tuple) by built-in function **list()**.

After you create a list, you can further revise it by adding new items, removing existing items, or changing the order of items.

Example 3.3: Try the following statements to create a few lists in different ways:

```
list1 = [1, 3, 5, 7]
print(list1)
list2 = ["Jack", "John", "Alice"]
print(list2)
list3 = [1, "Jack", 2, "John"]
print(list3)

#Generate a list from a range
my_range = range(10, 20, 2)
print(my_range)
list4 = list(my_range)
print(list4)

#Generate a list from a tuple
tuple1 = ("John", "Alice", "Jack")
list5 = list(tuple1)
print(list5)

#Is list5 equal to tuple1?
print(list5 == tuple1)
```

Remark: In Example 3.3, although **list5** has the same data items as **tuple1**, they are different because they have different types (list vs. tuple) and they support different operations.

List Operations: List structure supports many useful operations (i.e., data processing!). Below is a list of common list operations:

Operation	Explanation
Indexing through []	To access an item in the list
len(mylist)	Return the number of items in list ***mylist***
*mylist.**append**(object)*	Add an ***object*** at the end of the list ***mylist***
*mylist.**insert**(index, object)*	Add an ***object*** at the ***index*** location of the list ***mylist***. The index begins with 0.
*mylist.**remove**(object)*	Remove the ***object*** from the list ***mylist***. If multiple items in

	the list have the same value, the first one will be removed.
*mylist.**sort()***	Sort the ***items*** in the list ***mylist***.
*mylist.**reverse()***	Rearrange the items of list ***mylist*** in reverse order.

Example 3.4: Try the following statements to get familiar with list operations:

```
list6 = ["Jack", "John", "Alice"]
print(list6)
print(list6[0])
print(list6[-1])
print(len(list6))
list6.append("Jack")
print(list6)
list6.insert(2, "Jack")
print(list6)
list6.remove("Jack")
print(list6)
list6.sort()
print(list6)
list6.reverse()
print(list6)
```

There is another important operation in Python: **slicing**, that deserves a separate treatment.

Slicing means the access of a subset of items in a sequence of data items, which is very important in data analysis.

- You can access a range of items in a list **mylist** through mylist[start:stop], where **start** Is the index of the first item in your slice, and **stop−1** is the index of the last item in your slice.
- If your slice begins with the first element in the list, you can set **start** empty (i.e., mylist[:stop]).
- If your slice includes the items up to the end of the list, you can set **stop** empty (i.e., mylist[start:]).
- Slicing with increments through mylist[start:stop:step], where **step** specifies the increment between the indices.

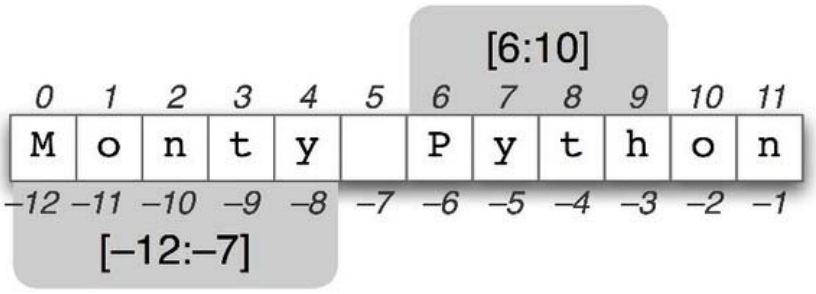

Fig. 2 An illustration of indexing and slicing

Let's learn from the following example:

Example 3.5: Try the following statements to get familiar with **list slicing**:

```
a = list(range(10))
print(a)
b = a[2:6]
print(b)
c = a[:5]
print(c)
d = a[5:]
print(d)
e = a[3:10:2]
print(e)
```

Dictionaries

Key-Value Pairs: Information is very often more than individual numbers or strings: they have relationships. For example, a person named "Jack" has an age of 35; his wife "Linda" has an age of 32; his daughter "Alice" has an age of 6; his father "David" has an age of 68. In Python, we can create a pair of data items that are related, e.g., "Jack":35, "Linda":32, "Alice":6, and "David":68. They are called **key-value pairs**: the key "Jack" has a value 35; the key "Linda" has a value 32; the key "Alice" has a value 6, and the key "David" has a value 68. Notice that the key and its value are separated by a colon.

Property of key-value pairs: In general, a key is a fixed and representative property of an object, and hence it should be immutable. Furthermore, the ***key*** should be unique in a collection of key-value pairs. E.g., the student ID is unique in the set HKBU students, so student ID can serve as the key for the set of student records. On the other hand, the value associated with a key is mutable and can be anything. The ***value*** can even be a list that includes a lot of data items, such as names, courses, marks, etc. E.g., a key value pair can be "17212346" : ["CHAN DAI MAN", "2000-01-01", "Male"].

Python Dictionary: If we have thousands to millions of key-value pairs like {"Jack":35}, could we quickly find out the age of a person (i.e., the value) if you know the name (i.e., the key)? The question is similar to looking up a dictionary to find out the explanation of an English word: the word is the key, and the explanation is the value. In Python, a data structured called ***dictionary*** is provided to organize and process key-value pairs.

A dictionary can be generated by providing a sequence of key-value pairs, enclosed by curly braces **{ }** and separated by comma.

Position indexing vs Key indexing: Different from tuples or lists, the items in a dictionary cannot be accessed through position indexing because the items are organized according to the keys, rather than the order provided by the users. To access the value of a key, the user can use the key as the index: **my_dict[key]**.

To add a new key-value pair into a dictionary, just use: **my_dict[new_key] = new_value**. If the **new_key** already exists in the dictionary, the old value will be replaced by **new_value** because a dictionary doesn't allow two key-value pairs with the same key.

To remove a key-value pair from a dictionary, use method **my_dict.pop(key)**.

Example 3.6: Try the following statements to get familiar with **dictionary**:

```
family = {"Jack":35, "Linda":32, "Alice":6}
print(family)
print(len(family))
family["David"] = 68
print(family)
print(len(family))
family["Alice"] = 7
print(family)
family.pop("David")
print(family)

stu = {"17212345":["CHAN DAI MAN", "2000-01-01", "Male"],
       "17216789":["WONG Maggie", "2000-06-21", "Female"]}

print(stu["17212345"])
#stu["17212345"] is a list. So what is stu["17212345"][0]?
print(stu["17212345"][0])

print(stu["17216789"])
#stu["17216789"] is a list. So what is stu["17216789"][2]?
print(stu["17216789"][2])

#add the Address information into the dictionary
stu["17212345"].append("KOWLOON TONG")
print(stu["17212345"])
stu["17216789"].append("SAI KONG")
print(stu["17216789"])
```

Summary:

In this Lab1B, we have reviewed and learnt the following data structures that can store a sequence of data items:

- tuple, list, dictionary

Tuples are immutable, while lists and dictionaries are mutable.

We also learned how to access, add, and delete data items of lists and dictionaries.

What to submit:

Please save all your source codes as a single Python file and submit through BU Moodle.