Section A2 List&Dictionary

List

pprint produces clear hierarchical views on collection of data such as *lists*, it is also used for debugging purposes (for situations where we use print statements).

```
from pprint import pprint
lst1 = ['John', 91, [90, [False, 'Smith'], 19, True, 'David'], 'Denise', 91, 'David', 'David', 91]
pprint(lst1, indent=1)
```

```
['John',
 91,
 [90, [False, 'Smith'], 19, True, 'David'],
 'Denise',
 91,
 'David',
 'David',
 91]
```

Lists Basic Operators

- **+** is used to concatenate lists
- ***** is used to repeat a list
- **in** is used to determine if a specific element is in the list.
- **not in** is to check if a specific element **is not** in the list.

Add List Elements

- To add an element at the end of the list, you need to use **append(element)** function
- To append elements from one list to another, you can use the **extend()** function. The difference between the + function (that we learnt earlier) and extend() is that the former creates a new list and the latter modifies an existing list.

Remove List Elements

- To search and remove the first matching element in a list, you need to use **remove(element)** function. If the element is not in the list, the method returns an error.
- If you use **pop()** function, it will remove the last element in the list.
- What if you want to delete the entire list? You can use **del** keyword.

```
colors = ['black', 'white ', 'yellow', 'red']
del colors
print(colors)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
ipykernel.py in <module>
      1 colors = ['black', 'white ', 'yellow', 'red']
      2 del colors
----> 3 print(colors)

NameError: name 'colors' is not defined
```

- Instead of deleting the list completely, you can also empty the list by using **clear()** function.

```
colors = ['black', 'white ', 'yellow', 'red']
colors.clear()
print(colors)
```

```
[]
```

Copy Lists
When you assign a list (lst1) to another list (lst2), e.g. lst2 = lst1, both lst2 and lst1 refer to the **same** list. If you modify either of the lists, the other list is modified as well.

You can do this using **list()** or by using the **copy()** method. With this, you assign the lists to different memory spaces.
lst2 = list(lst1) or **lst2 = lst1.copy()**

```
lst1 = ['black', 'white ', 'yellow', 'red']
lst2 = list(lst1)
lst1.append('purple')
print(lst1)
print(lst2)
```

```
['black', 'white ', 'yellow', 'red', 'purple']
['black', 'white ', 'yellow', 'red']
```

Sort Lists
Python has **sort()** function that will sort the elements in a list in ascending order, by default.
If you want to sort the list in descending order, you will have to use **reverse = True**, as shown in the examples below.

```
lst1 = ['black', 'white ', 'yellow', 'red', 'purple']
#Sort the list descending
lst1.sort(reverse = True)
print(lst1)
```

```
['yellow', 'white ', 'red', 'purple', 'black']
```

The sort() function is case-sensitive; thus, if you have a list with both capital and lower-case letters, it will sort all the capital letter elements first, before the lower-case letters.

```
lst1 = ['black', 'White ', 'Yellow', 'red', 'purple']
lst1.sort()
print(lst1)
```

```
['White ', 'Yellow', 'black', 'purple', 'red']
```

In Python, we are allowed to customise your sort by using key. So, if you want to make your sort() function case-insensitive, you can do it as follows.

```
lst1 = ['black', 'White ', 'Yellow', 'red', 'purple']
lst1.sort(key = str.lower)
print(lst1)
```

```
['black', 'purple', 'red', 'White ', 'Yellow']
```

也可以用序列类型函数 sorted(list) 进行排序
sorted(list) 返回一个对象，可以用作表达式。原来的 list 不变，生成一个新的排好序的 list 对象。
list.sort() 不会返回对象，改变原有的 list。

Count List Elements
You can use the **count(value)** function to return the number of elements with the specified value in a given list. Note that the value that we search for could be any Python data type (e.g., *string*, *number* or even *list*).

Reverse List Elements
You can use the reverse function to reverse the order of the elements in a given list

```
lst1 = ['black', 'White ', 'Yellow', 'red', 'purple']
lst1.reverse()
print(lst1)
```

```
['purple', 'red', 'Yellow', 'White ', 'black']
```

For Loops

```
for var in iterable:
    statement(s)
```

```
#This code prints all the characters in a string
str1 = 'Hello!'
for s in str1:
    print(s)
```

Creating a sequence using the range( ) function

The built-in function **range()** takes 3 values: the start of the range, the end of the range, and the increment (or step) between values. All of these are optional other than the endpoint. If not specified, the start is 0 and the step is 1. For example, range(10) generates the list [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Dictionary

Creating a Dictionary

```
age1 = {"John":52, "Siobhan":21, "Ye":18}
print(type(age1))

age2 = dict(John=52, Siobhan=21, Ye=18)
print(type(age2))
print(len(age2))
```

You aren't limited to strings. Your **key** can be any type. Well, almost. The type has to be *immutable*, which means it can't be modifiable. int, float, bool, string are all immutable, so your key can be any of these.
The type(s) of the **values** have no restrictions.
You can also mix types for both the key and the values
If duplicate keys were encountered, the values will be overwritten using existing values.
from Python version 3.7 onwards, dictionaries are *ordered* (as lists), meaning the elements in a dictionary have a defined order, and that order won't change

Add Dictionary Elements & Access Dictionary Elements

To add an element to a dictionary, you can use a new index key and assign a value to it

```
details = {"fname":"John", "height":"190cm", "weight":"52kg", "age":30}
#add new dictionary elements
details["lname"] = "Smith"
details["interests"] = ["programming", "reading", "cooking", "gardening"]
```

To access elements of a dictionary, you can use the specific key name inside of square brackets

Update Dictionary Elements

You can change the value of a specific element by using its key name. In addition, you can use the **update()** function with key:value pairs

```
details = {'fname': 'John', 'height': '190cm', 'weight': '52kg', 'age': 30, 'lname': 'Smith', 'interests': ['programming', 'readin
g', 'cooking', 'gardening']}
#update dictionary elements
details["lname"] = "Simon"
print(details)

details.update({"lname": "Denise"})
print(details)
```

Remove Dictionary Elements

To remove a dictionary element, use **pop()** function with the specified key name. Same as lists you can use **clear()** to empty the dictionary and **del** to delete the dictionary.

```
#remove dictionary elements
details.pop("interests")
print(details)
```

Dictionary Operations

```
# repr( ) returns the string representation of the dictionary
print("age dictionary: " + repr(age))

# len( ) gives the length of the dictionary
print("There are "+ str(len(age)) + " key:value pairs in the dictionary")

# copy( ) makes a copy of the dictionary
ageCopy = age.copy()
print("age dictionary copy: " + repr(ageCopy)) # return the string representation of the dictionary

# difference between == and is
print("copies are equivalent? ", age == ageCopy) # == and != check if lists have same key:value pairs
print("are age and ageCopy the same dictionary? ", age is ageCopy) # note the difference between == and is!

# removing elements from the dictionary
# pop( ) removes and returns the value
ageJohn = age.pop("John")
print(ageJohn)

# del ( ) removes but doesn't return the return value
del age["Siobhan"]

# John and Siobhan have both been removed so list should now be empty
print("age dictionary: " + repr(age))

# but the copy of the dictionary hasn't been changed
print("age dictionary copy: " + repr(ageCopy))
print("copies are equivalent? ", age == ageCopy)

# clear( ) removes everything in a dictionary
ageCopy.clear()
print("age dictionary copy: " + repr(ageCopy)) # return the string representation of the dictionary
```

Checking if a key is in your Dictionary

To avoid an error trying to remove a key that doesn't exist in the dictionary, you should check first if the key exists.

```
age = { }
if "John" in age:
    print("John is in age dictionary")
    del age["John"]
```

Manipulating Dictionary Data

- **items()** get all the key:value pairs
- **keys()** get all the keys
- **values()** get all the values

```
age = {"John":52, "Siobhan":21, "Ye":18}

for k,v in age.items():
    print(k, ":", v)
print()

for k in age.keys():
    print(k)
print()
```