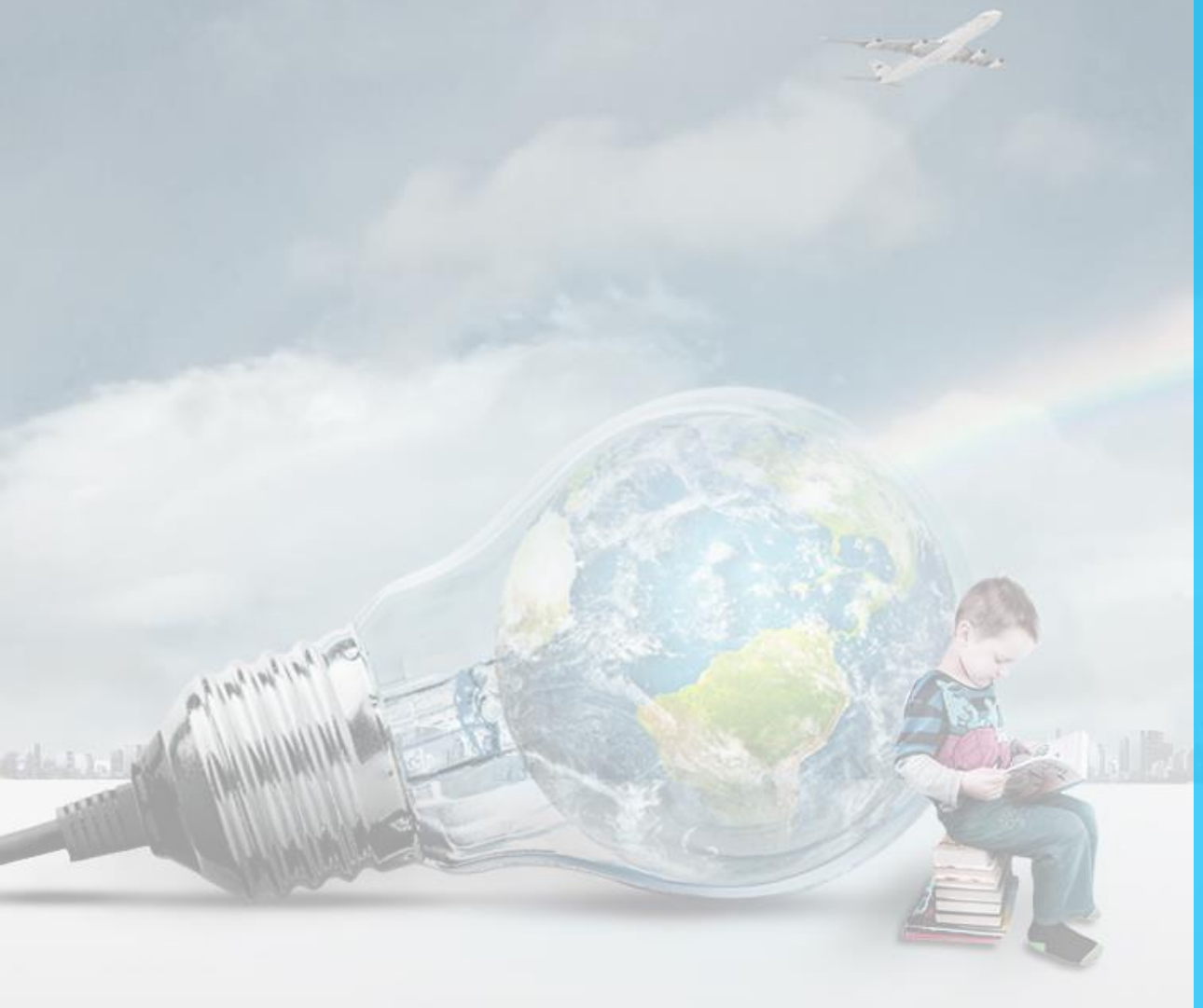
COMP07027 Introduction to Programming

School of Computing, Engineering and Physical Sciences Scotland Academy Wuxi

Lecture 3: List, Tuple

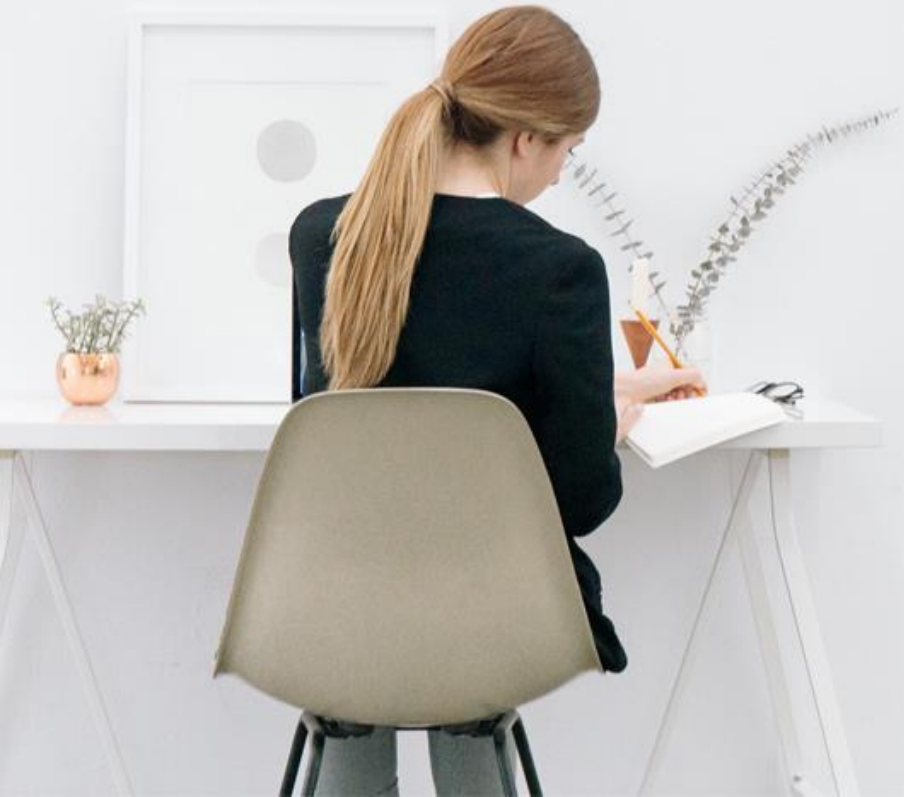
Dr Muhammad Aslam
Lecturer UWS Wuxi
Muhammad.Aslam@uws.ac.uk

Agenda of the lecture 3



01 **Python Lists**

02 **Python Tuples**



1-Python Lists

Python Lists

- With the help of Lists multiple items can be stored in a single variable.
- Lists are one of four built-in Python data structures for storing collections of data; the other three are Tuple, Set, and Dictionary, all of which have different properties and applications.
- Square brackets are used to make lists:

```
mylist = [“Football", “Hockey", “Tennis"]
print(mylist)
```

- **List items are ordered:** Entries of lists are arranged in a specific order that will not change. When you add new items to a list, they are added to the end of the list.
- **Changeable:** The list is changebale, which means that after it has been generated, we can update, add, and remove entries from it.
- **Allow Duplicate values:**Lists with the same value can exist since they are indexed:

```
mylist = ["apple", "banana", "cherry", "apple", "cherry"]
print(mylist)
```

Python Lists length and datatypes

- Use the len() method to find out how many items are in a list:

```
mylist = [“Football", “Hockey", “Tennis"]
print(len(mylist))
```

- Any data type can be used as a list item:

```
l1 = [“Football", “Hockey", “Tennis"]
l2 = [6, 4, 76, 6, 3]
l3 = [9.5, 5.3, 4.5]
```

- Different data types can be found in a single list:

```
list1 = [“Jin", 32, False, 22, “Hi”, 22]
```

- Lists are defined as objects of the data type 'list' from the Python perspective:

```
mylist = [“Football", “Hockey", “Tennis"]
print(type(mylist))
```

- When building a new list, you can also use the list() constructor.

```
mylist = list((“Football", “Hockey", “Tennis“))
print(mylist)
```

Python - Access List Items

- The items in the list are indexed, and you can find them by looking up the index number:

```
mylist = [“Football", “Hockey", “Tennis“]
print(mylist[0])
```

- Starting at the end is referred to as negative indexing.

```
mylist = [“Football", “Hockey", “Tennis“]
print(mylist[-1])
```

- Slicing of Lists

```
mylist = [“Football", “Hockey", “Tennis“]
print(mylist[0:1])
print(mylist[0:])
print(mylist[:2])
print(mylist[-1:-2])
```

- Use the in keyword to see if a specific item is contained in a list:
-

```
mylist = [“Football", “Hockey", “Tennis“]
if “tennis" in mylist:
  print("Yes, ‘tennis' is in the sports list")
```

Python - Change List Items

- Refer to the index number to update the value of a certain item:

```
mylist = [“Football", “Hockey", “Tennis“]
mylist[0] = [“table tennis”]
        print(mylist[0:1])
```

- Change a Range of Item Valuesthislist =

```
["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

- If you replace more itmes than you insert, the new ones will be placed where you stated.

```
thislist = ["apple", "banana", "cherry"]
thislist[1:2] = ["blackcurrant", "watermelon"]
print(thislist)
```

- Inserts an item at the supplied index with the insert() method:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(2, "watermelon")
print(thislist)
```

Python - Change List Items

- Refer to the index number to update the value of a certain item:

```
mylist = [“Football", “Hockey", “Tennis“]
mylist[0] = [“table tennis”]
        print(mylist[0:1])
```

- Change a Range of Item Valuesthislist =

```
["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

- If you replace more itmes than you insert, the new ones will be placed where you stated.

```
thislist = ["apple", "banana", "cherry"]
thislist[1:2] = ["blackcurrant", "watermelon"]
print(thislist)
```

- Inserts an item at the supplied index with the insert() method:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(2, "watermelon")
print(thislist)
```

Python - Add List Items

- Add lists: Use the append() method to add an item to the end of the list:

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

- Insert lists: Use the insert() method to insert a list item at a specific index.

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```

- Extend Lists: Use the extend() method to append elements from another list to the current one.

```
thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
thislist.extend(tropical)
print(thislist)
```

Python - Remove List Items

- Remove Specified Item: Removes the provided object with the remove() function.:

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

- Remove Specified Index

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

- By default pop() will remove last index value

```
thislist = ["apple", "banana", "cherry"]
thislist.pop()
print(thislist)
```

- Delete the ite with del

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

Python - Remove List Items

- Remove Specified Item: Removes the provided object with the remove() method:

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

- Remove Specified Index

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

- By default pop() will remove last index value

```
thislist = ["apple", "banana", "cherry"]
thislist.pop()
print(thislist)
```

- Delete the item with del

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

Python - Loop Lists

- Loop Through a List

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
      print(x)
```

- List Comprehension offers the shortest syntax for looping through lists:

```
thislist = ["apple", "banana", "cherry"]
[print(x) for x in thislist]
```

Python - Sort Lists

- By default, the sort() function on list objects sorts the list alphanumerically, ascending:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort()
print(thislist)
```

- Sort the list numerically: by default it will be ascending order

```
thislist = [100, 50, 65, 82, 23]
thislist.sort()
print(thislist)
```

- Sort Descending:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort(reverse = True)
print(thislist)
```

- Reverse Order:

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.reverse()
print(thislist)
```

Python - Copy Lists

- You can't just do list2 = list1 to clone a list since list2 is only a reference to list1, and any changes made to list1 will be reflected in list2.

- The copy() method can be used to duplicate a list. :

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist))
```

- Another way to make a copy is to use the built-in method list(). :

```
thislist = ["apple", "banana", "cherry"]
mylist = list(thislist)
print(mylist)
```

Python - Join Lists

- The easiest ways are by using the + operator..

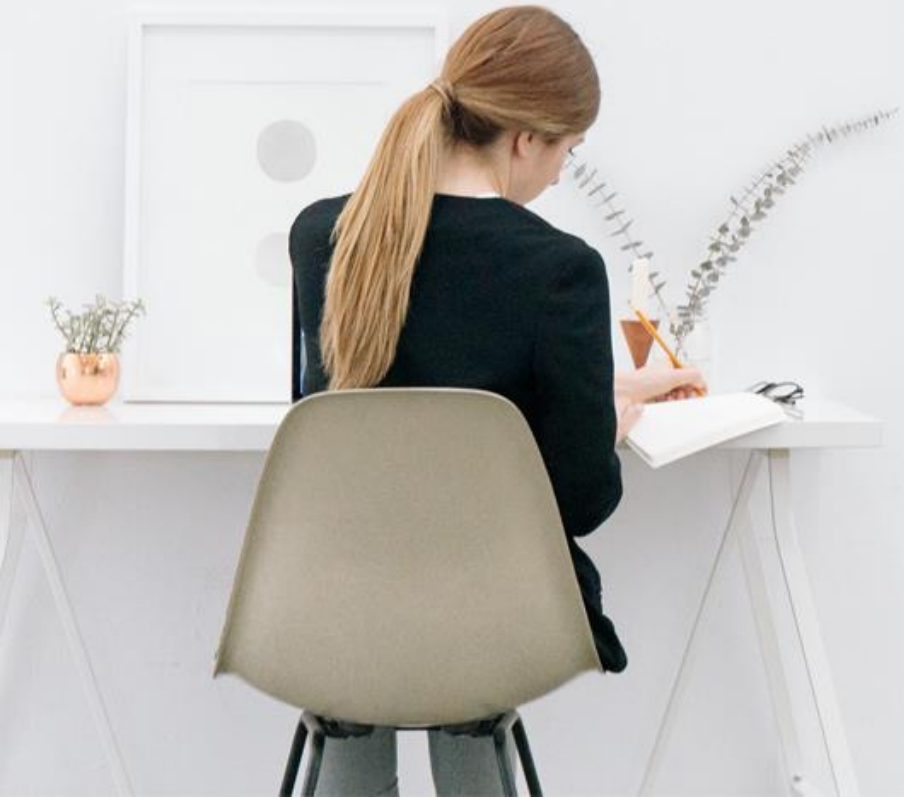
```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]

list3 = list1 + list2
print(list3)
```

- Append list2 into list1

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

for x in list2:
  list1.append(x)

print(list1)
```

- Or you can use the extend() method

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

list1.extend(list2)
print(list1)
```



1-Python Tuples

Python Tuples

- With the help of tuple multiple items can be stored in a single variable.
- Round brackets are used to make lists:

```
mytuple = (“Football", “Hockey", “Tennis“)
print(mytuple)
```

- **Tuples items are ordered:** Entries of lists are arranged in a specific order that will not change.
- **Unchangeable:** The Tuples are unchangeable, which means that after it has been generated, we can not update, add, and remove entries from it.
- **Allow Duplicate values:**

```
mytuple = ("apple", "banana", "cherry", "apple", "cherry“)
print(mytuple)
```

Python Tuples length and datatypes

- Use the len() method to find out how many items are in a tuple:

```
mytuple = (“Football", “Hockey", “Tennis“)
print(len(mytuple))
```

- Any data type can be used as a list item:

```
t1 = (“Football", “Hockey", “Tennis“)
t2 = (6, 4, 76, 6, 3)
t3 = (9.5, 5.3, 4.5)
```

- Different data types can be found in a single tuple:

```
tuple1 = (“Jin", 32, False, 22, “Hi”, 22)
```

- tuple are defined as objects of the data type ‘tuple' from the Python perspective:

```
mytuple = (“Football", “Hockey", “Tennis“)
print(type(mytuple))
```

- When building a new tuple, you can also use the tuple() constructor.

```
mytuple = tuple((“Football", “Hockey", “Tennis“))
print(mytuple)
```

Python - Access tuple Items

- The items in the list are indexed, and you can find them by looking up the index number:

```
mytuple = (“Football", “Hockey", “Tennis“)
print(mytuple[0])
```

- Starting at the end is referred to as negative indexing.

```
 mytuple = (“Football", “Hockey", “Tennis“)
print(mytuple[-1])
```

- Slicing of tuples:

```
 mytuple = (“Football", “Hockey", “Tennis“)
print(mytuple[0:1])
print(mytuple[0:])
print(mytuple[:2])
print(mytuple[-1:-2])
```

- Use the in keyword to see if a specific item is contained in a tuple:
-

```
mytuple = (“Football", “Hockey", “Tennis“)
if “tennis" in mytuple:
  print("Yes, ‘tennis' is in the sports tuple")
```

Python - Update Tuples

- Tuples are unchangeable, meaning that you cannot change, add, or remove items once the tuple is created.
- Change Tuple Values: Convert the tuple into a list to be able to change it:
- ```
  x = (“Football", “Hockey", “Tennis“)
  y = list(x)
  y[1] = “cricket"
  x = tuple(y)
  print(x)
  ```
- Add Value: Convert the tuple into a list, add "orange", and convert it back into a tuple:
- ```
  mytuple = (“Football", “Hockey", “Tennis“)
  y = list(mytuple)
  y.append("orange")
  mytuple = tuple(y)
  ```
- **Add tuple to a tuple**:

```
thistuple = ("apple", "banana", "cherry")
y = ("orange",)
thistuple += y
print(thistuple)
```

Python - Unpack Tuples

- The items in the list are indexed, and you can find them by looking up the index number:
- Packing a tuple:

```
mytuple = (“Football", “Hockey", “Tennis“)
print(mytuple)
```

- Unpacking a tuple:

```
fruits = ("apple", "banana", "cherry")
(x, y, z) = fruits
print(x)
print(y)
print(z)
```

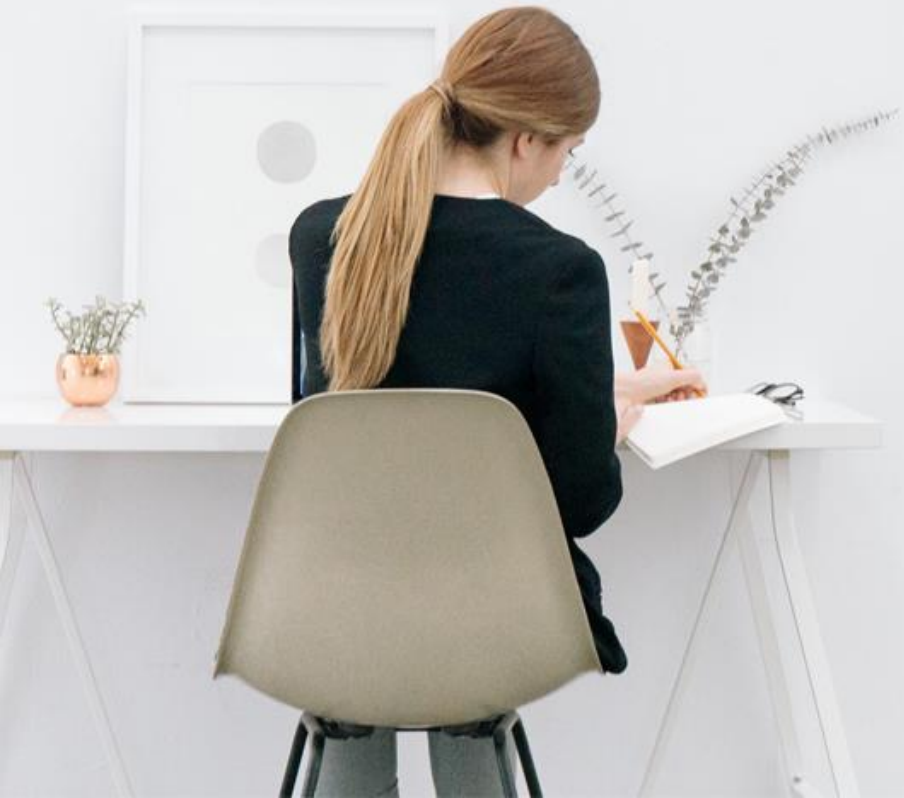
Python - Loop Tuples

- Iterate through the items and print the values:

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```

- Print all items by referring to their index number:

```
thistuple = ("apple", "banana", "cherry")
for i in range(len(thistuple)):
  print(thistuple[i])
```



1-Python Sets

Python Sets

- Sets are used to store multiple items in a single variable.
- A set is a collection which is both *unordered* and *unindexed*.
- Round brackets are used to make lists:

```
myset = {“Football", “Hockey", “Tennis”}
print(myset)
```

- **Tuples items are unordered and unindexed:** Set items are unordered, unchangeable, and do not allow duplicate values.

```
myset = {“Football", “Hockey", “Tennis”}
print(myset)
```

- Duplicate values will be ignored:

```
myset = {"apple", "banana", "cherry", "apple"}
print(myset)
```

Python Sets length and datatypes

- Use the len() method to find out how many items are in a se:

```
myset = {“Football", “Hockey", “Tennis“}
print(len(myset))
```

- Any data type can be used as a list item:

```
s1 = {“Football", “Hockey", “Tennis“}
s2 = {6, 4, 76, 6, 3}
s3 = {9.5, 5.3, 4.5}
```

- Different data types can be found in a single set:

```
myset = {“Jin", 32, False, 22, “Hi”, 22}
```

- tuple are defined as objects of the data type ‘set' from the Python perspective:

```
myset = {“Football", “Hockey", “Tennis“}
print(type(myset))
```

- In building a new set, we can also use the set() constructor.

```
myset = set((“Football", “Hockey", “Tennis“))
print(myset)
```

Python - Access set Items

- The items in the set are unindexed, that's why we can not access the item od set with indexing.
- Loop through the set, and print the values:
- ```
  mytset = {“Football", “Hockey", “Tennis”}
  for x in myset:
    print(x)
  ```
- Check if "banana" is present in the set:
- ```
  mytset = {“Football", “Hockey", “Tennis”}
  print(“Hockey" in myset)
  ```

- Once a set is created, you cannot change its items, but you can add new items.

Python - Add sets

- To add one item to a set use the add() method

```
myset = {“Football", “Hockey", “Tennis”}
myset.add(“cricket")
print(myset)
```

- Add elements from tropical into myset:

```
thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}
thisset.update(tropical)
print(thisset)
```

- Add Any Iterable:

```
thisset = {"apple", "banana", "cherry"}
mylist = ["kiwi", "orange"]
thisset.update(mylist)
print(thisset)
```

Python - Remove Set Items

We can specify the data type, although we do not need particularly.

```
myset = {“Football", “Hockey", “Tennis”}
myset.remove(“Hockey")
print(myset)
```

- Remove the last item by using the pop() method:

```
myset = {“Football", “Hockey", “Tennis”}
myset.pop()
print(myset)
```

- The clear() method empties the set:

```
myset = {“Football", “Hockey", “Tennis”}
myset.clear()
print(myset)
```

Python - Loop Sets

- Loop through the set, and print the values:

```
myset = {“Football", “Hockey", “Tennis”}
for x in myset:
  print(x)
```

Join Two Sets

•You can use the union() function to create a new set that has all of the items from both sets:

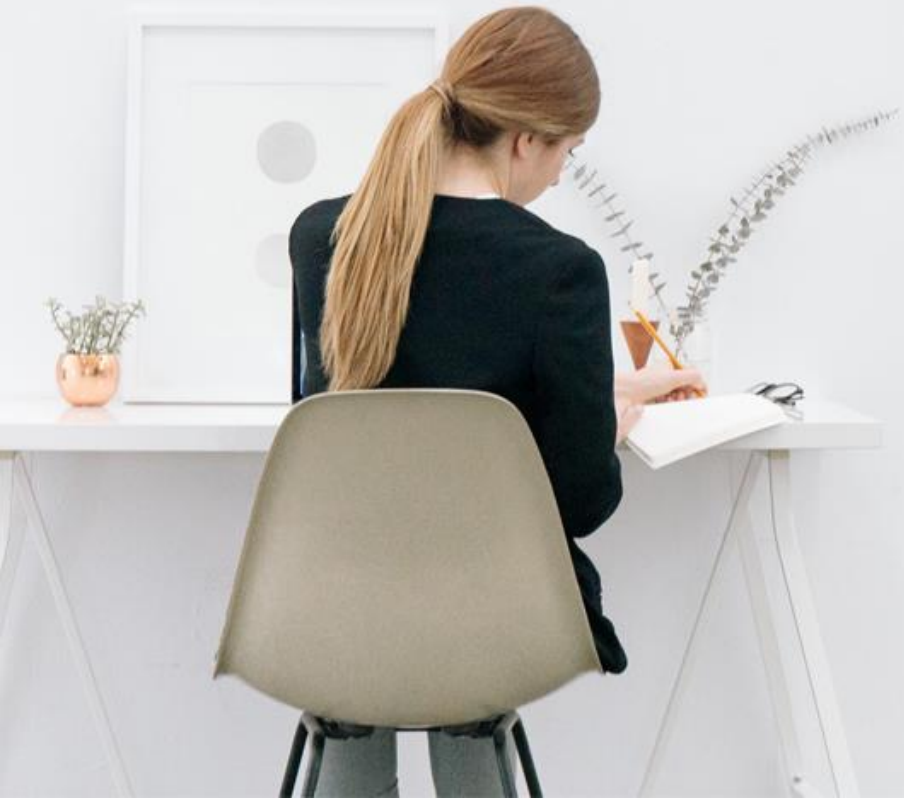
```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
set3 = set1.union(set2)
print(set3)
```

- you can use the update() method to insert all of the things from one set into another:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
set1.update(set2)
print(set1)
```

- The intersection_update() method will keep only the items that are present in both sets.

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
x.intersection_update(y)
print(x)
```



1-Python Dictionaries

Python Dictionaries

- Data values are stored in key:value pairs using dictionaries.
- Curly brackets are used to write dictionaries, which have keys and values:.

```
mydict = {
  “class": “CS",
  “suject": “Programming",
  "year": 2021
}
print(mydict)
```

- Dictionary items are ordered, changeable, and does not allow duplicates.
- The key name can be used to refer to dictionary elements, which are given in key:value pairs.

```
mydict = {
  “class": “CS",
  “suject": “Programming",
  "year": 2021
}
print(mydict[“class”])
```

Python Dictionaries

- Dictionaries cannot have two items with the same key:

```
mydict = {
  “class": “CS",
  “suject": “Programming",
  "year": 2021,
  "year": 2020,
}
print(mydict)
```

- Print the number of items in the dictionary:

```
mydict = {
  “class": “CS",
  “suject": “Programming",
  "year": 2021

}
print(len(mydict))
```

Dictionary Items - Data Types

- String, int, boolean, and list data types:

```
mydict = {
  “class": “CS",
  “suject": “Programming",
  "year": 2021

}
print(mydict)
```

- Print the data type of a dictionary:

```
mydict = {
  “class": “CS",
  “suject": “Programming",
  "year": 2021

}
print(type(mydict))
```

There is also a method called get() that will give you the same result.

Python - Access Dictionary Items

- The items in a dictionary can be accessed by referring to the key name, which is enclosed in square brackets:

```
mydict = {
  “class": “CS",
  “suject": “Programming",
  "year": 2021
}
print(mydict[“class”])
```

- Get the different part of doctionary:

```
x = mydict.get("model")
print(x)
y = mydict.keys()
print(y)
z = mydict.values()
print(z)
u = mydict.items()
print(u)
```

Python - Change Dictionary Items

- You can change the value of a specific item by referring to its key name:
-
```
mydict = {
  “class": “CS",
  “suject": “Programming",
  "year": 2021
}
mydict[“class”] = 2022
```

- Update the "year" of the car by using the update() method:

```
mydict = {
  “class": “CS",
  “suject": “Programming",
  "year": 2021
}
mydict.update({"year": 2022})
```

Python - Add Dictionary Items

- Adding an item to the dictionary is done by using a new index key and assigning a value to it:

```
mydict = {
  “class": “CS",
  “suject": “Programming",
  "year": 2021
}
mydict[“room"] = 230
print(mydict)
```

- Add a month item to the dictionary by using the update() method:

```
mydict = {
  “class": “CS",
  “suject": “Programming",
  "year": 2021
}
mydict.update({“month”: “Sep”})
print(mydict)
```

Python - Remove Dictionary Items

- The pop() method removes the item with the specified key name:

```
mydict = {
  “class": “CS",
  “suject": “Programming",
  "year": 2021
}
mydict.pop(“year”)
print(mydict)
```

- The clear() method empties the dictionary::

```
mydict = {
  “class": “CS",
  “suject": “Programming",
  "year": 2021
}
mydict.clear()
print(mydict)
```

Loop Through a Dictionary

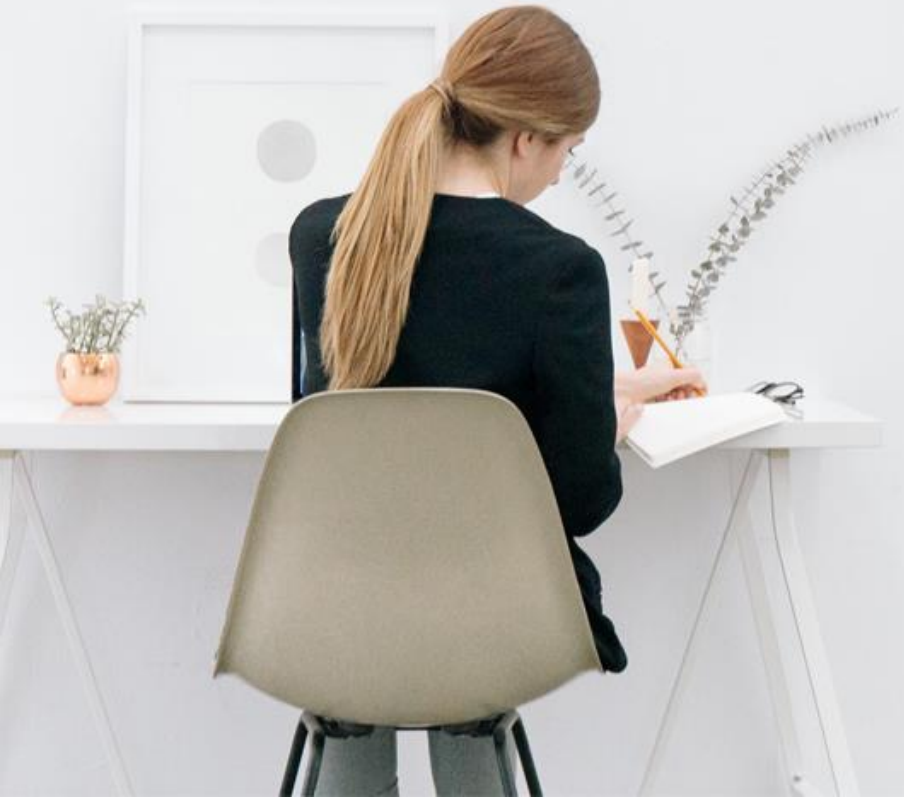
- Print all key names in the dictionary, one by one:

```
mydict = {
  “class": “CS",
  “suject": “Programming",
  "year": 2021
}
for x in mydict:
  print(x)
for x in mydict.values():
  print(x)
for x in mydict.keys():
  print(x)
for x, y in thisdict.items():
  print(x, y)
```

Python - Copy Dictionaries

- Make a copy of a dictionary with the copy() method:

```
mydict = {
  “class": “CS",
  “suject": “Programming",
  "year": 2021
}
newdict = mydict.copy()
print(mydict)
print(newdict)
```



1-Python if…else

Python If … Else

- Python supports the following standard mathematical logical conditions:
    1. Equals: `a == b`
    2. Not Equals: `a != b`
    3. Less than: `a < b`
    4. Less than or equal to: `a <= b`
    5. Greater than: `a > b`
    6. Greater than or equal to: `a >= b`
- These conditions can be employed in a variety of situations, the most popular of which being "if statements" and loops.
- The if keyword is used to create a "if statement.“

```
x = 88
y = 76
if x > y:
  print(“x is greater than y")
```

- Indentation: We have to follow basic python syntax to follow the line rules
-

```
x = 88
y = 76
if x > y:
print(“x is greater than y") # this will show us an error of indentation
```

Python Elif

- The elif keyword in Python means "attempt this condition if the previous conditions were not true:

```
x = 88
y = 76
if x > y:
  print(“x is greater than y")
elif x < y:
    print(“y is greater than x")
```

- Indentation: We have to follow basic python syntax to follow the line rules

```
x = 88
y = 76
if x > y:
print(“x is greater than y") # this will show us an error of
indentation
```

Python else

- Anything that isn't covered by the preceding conditions is caught by the else keyword.

```
x = 88
y = 76
if x > y:
  print(“x is greater than y")
elif x == y:
    print(“Both values are equal")
Else:
    print(“y is greater than x ")
```

- You can also have an else without the elif:

```
x = 88
y = 76
if x > y:
  print(“x is greater than y")
Else:
    print(“y is greater than x ")
```

Python else

- Anything that isn't covered by the preceding conditions is caught by the else keyword.

```
x = 88
y = 76
if x > y:
  print(“x is greater than y")
elif x == y:
    print(“Both values are equal")
Else:
    print(“y is greater than x ")
```

- You can also have an else without the elif:

```
x = 88
y = 76
if x > y:
  print(“x is greater than y")
Else:
    print(“y is greater than x ")
```

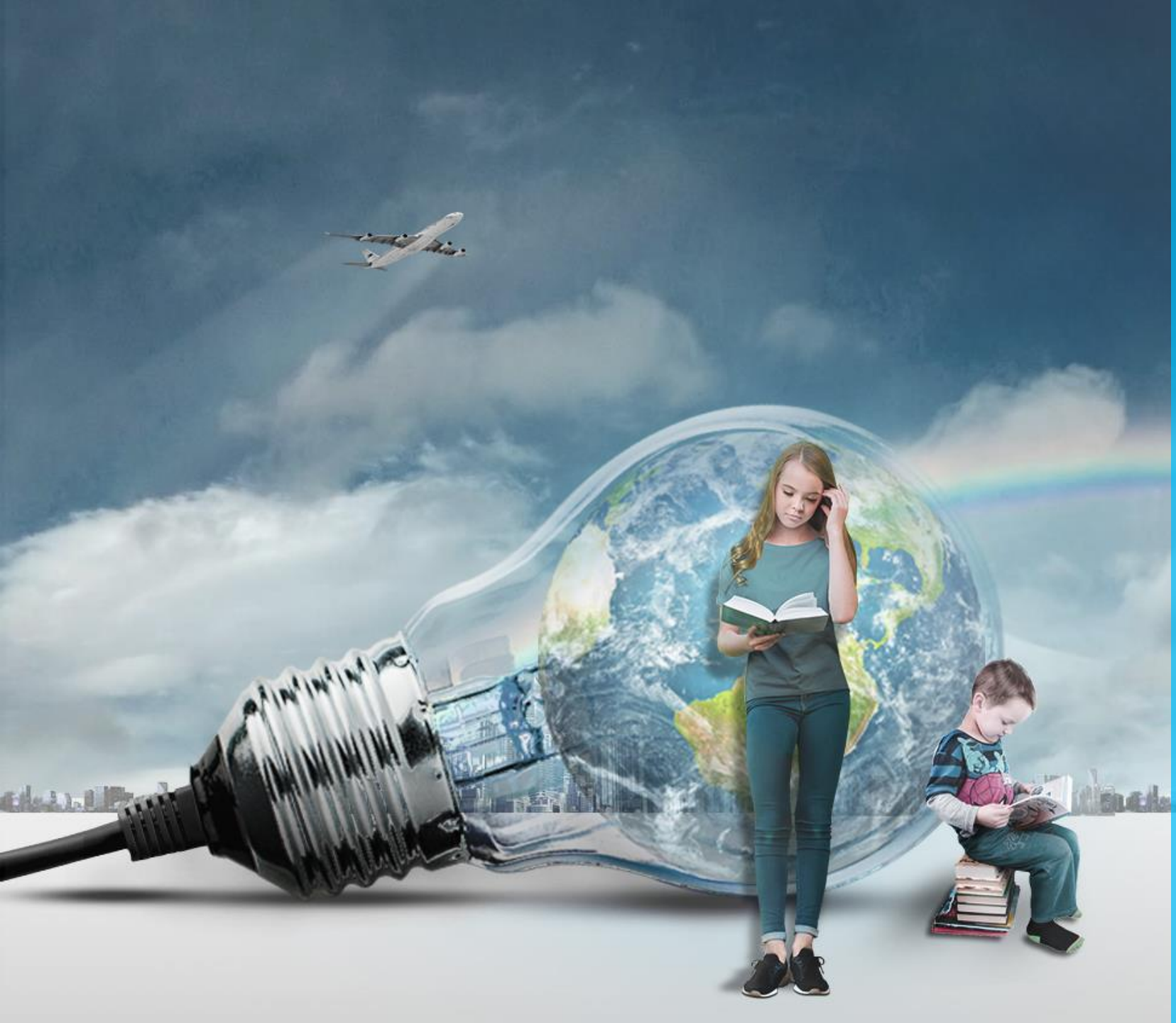
Python with And condition

- The and keyword is a logical operator that joins conditional expressions together:

```
x = 88
y = 76
Z = 30
if x > y and y > z:
  print("x is greater than y and z")
elif x < y and y > z :
    print("y is greatest value")
elif x < y and y < z :
    print("z is greatest value")
Else:
    print(" Other possible situation can be true")
```

- The or keyword is a logical operator:

```
x = 88
y = 76
Z = 30
if x > y or z < y:
  print("Both statements are true")
```



Thank You