ABDELGHAFOR'S VIRTUAL INTERNSHIP

PYTHON PROGRAM

SESSION (1)

PREPARED BY : MARK KOSTANTINE

HELLO!

Warm greetings to all present. As we gather here today, I am excited to be with you in **Abdelghafor's Virtual Internships - Python Program ,** Congratulations to you all for being selected

AGENDA OVERVIEW

04

LISTS

13

TUPLES

22

DICTIONARIES

35

TASKS

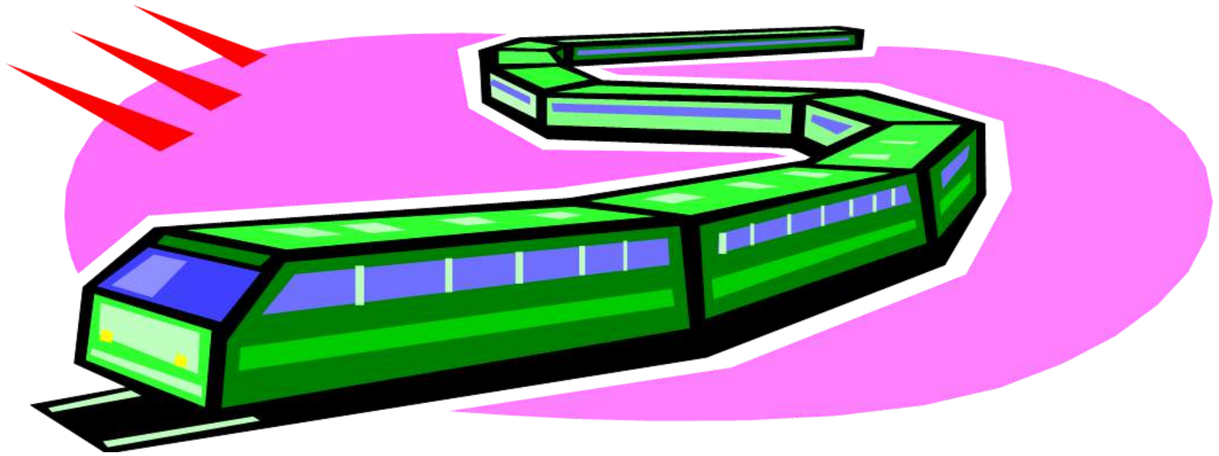
39

QUESTIONS

LISTS

Lists are used to store multiple items in a single variable

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



LISTS

List items are ordered, changeable, and allow duplicate values.
List items are indexed, the first item has index [0], the second item has index [1] etc.

01

ORDRED

- When we say that lists are ordered, it means that the items have a defined order, and that order will not change.
- If you add new items to a list, the new items will be placed at the end of the list.

02

CHANGABLE

The list is changeable, meaning that we can change, add, and remove items in a list after it has been created

03

ALLOW DUPLICATES

Since lists are indexed, lists can have items with the same value

LIST LENGTH

To determine how many items a list has, use the **len()** function

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

LIST ITEMS - DATA TYPES

List items can be of any data type

```
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]
```

ACCESS LIST ITEMS

List items are indexed and you can access them by referring to the index number

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

CHANGE LIST ITEMS

To change the value of a specific item, refer to the index number

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

ADD LIST ITEMS

Append Items : To add an item to the end of the list, use the **append()** method

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

Insert Items : To insert a list item at a specified index, use the **insert()** method

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

ADD LIST ITEMS

Extend List : To append elements from another list to the current list, use the **extend()** method

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

Note : List before extend function is the first list in the sequence

REMOVE LIST ITEMS

Remove Specified Item : The **remove()** method removes the specified item

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

Remove Specified Index : The **pop()** method removes the specified index

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

REMOVE LIST ITEMS

If you do not specify the index, the **pop()** method removes **the last item**

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

Clear the List : The **clear()** method empties the list , The list still remains, but it has no content

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

LISTS EXERCISE

Print the second item in the fruits list

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

TUPELS

- Tuples are used to store multiple items in a single variable.
- A tuple is a collection which is **ordered** and **unchangeable.**
- Tuples are written with **round brackets.**

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

- Tuple items are ordered, unchangeable, and allow duplicate values.
- Tuple items are indexed, the first item has index [0], the second item has index [1] etc.

ORDERED

When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.

UNCHANGEABLE

Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

ALLOW DUPLICATES

Since tuples are indexed, they can have items with the same value

TUPLE LENGTH

- To determine how many items a tuple has, use the **len()** function

```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

CREATE TUPLE WITH ONE ITEM

- To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

```
thistuple = ("apple",)
print(type(thistuple))
```

```
#NOT a tuple
thistuple = ("apple")
print(type(thistuple))
```

TUPLE ITEMS - DATA TYPES

- Tuple items can be of any data type

```
tuple1 = ("apple", "banana", "cherry")
tuple2 = (1, 5, 7, 9, 3)
tuple3 = (True, False, False)
```

- A tuple can contain different data types

```
tuple1 = ("abc", 34, True, 40, "male")
```

ACCESS TUPLE ITEMS

- You can access tuple items by referring to the index number, inside **square brackets**

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

NEGATIVE INDEXING

- Negative indexing means start from the end.
- -1 refers to the last item, -2 refers to the second last item etc.

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[-1])
```

RANGE OF INDEXES

- You can specify a range of indexes by specifying where to start and where to end the range.
- When specifying a range, the return value will be a new tuple with the specified items.

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:5])
```

CHECK IF ITEM EXISTS

- Check if "apple" is present in the tuple

```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")
```

- Tuples are unchangeable, meaning that you cannot change, add, or remove items once the tuple is created. But there are some workarounds.
- Once a tuple is created, you cannot change its values. Tuples are unchangeable, or immutable as it also is called. But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

DELETE TUPLE COMPLETELY

The **del** keyword can delete the tuple completely

```
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) #this will raise an error because the tuple no longer exists
```

UNPACKING A TUPLE

When we create a tuple, we normally assign values to it. This is called **"packing"** a tuple

```
fruits = ("apple", "banana", "cherry")
```

But, in Python, we are also allowed to extract the values back into variables. This is called **"unpacking"**

```
fruits = ("apple", "banana", "cherry")

(green, yellow, red) = fruits

print(green)
print(yellow)
print(red)
```

Note : The number of variables must match the number of values in the tuple, if not, you must use an asterisk to collect the remaining values as a list.

UNPACKING A TUPLE

If the number of variables is less than the number of values, you can add an * to the variable name and the values will be assigned to the variable as a list

```
fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")

(green, yellow, *red) = fruits

print(green)
print(yellow)
print(red)
```

If the asterisk is added to another variable name than the last, Python will assign values to the variable until the number of values left matches the number of variables left.

TUPLES EXERCISE

Use negative indexing to print the last item in the tuple

```
fruits = ("apple", "banana", "cherry")
print(          )
```

DICTIONARIES

- Dictionaries are used to store data values in key:value pairs.
- A dictionary is a collection which is **ordered**, **changeable** and **do not allow duplicates**.
- Dictionaries are written with **curly brackets**, and have **keys and values**
- Dictionary items are presented in key:value pairs, and **can be referred to by using the key name**

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

01

ORDERED OR UNORDERED

When we say that dictionaries are ordered, it means that the items have a defined order, and that order will not change.

02

CHANGEABLE

Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.

03

DUPLICATES NOT ALLOWED

Dictionaries cannot have two items with the same key

DICTIONARY LENGTH

To determine how many items a dictionary has, use the **len()** function:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964,
  "year": 2020
}
print(len(thisdict))
```

ITEMS - DATA TYPES

The values in dictionary items can be of any data type

```
thisdict = {
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["red", "white", "blue"]
}
```

ACCESS DICTIONARY ITEMS

- You can access the items of a dictionary by referring to its key name, inside **square brackets**

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict["model"]
```

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

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict.get("model")
print(x)
```

GET KEYS

- The **keys()** method will return a list of all the keys in the dictionary
- The list of the keys is a view of the dictionary, meaning that any changes done to the dictionary will be reflected in the keys list.

```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.keys()

print(x) #before the change

car["color"] = "white"

print(x) #after the change
```

GET VALUES

- The **values()** method will return a list of all the values in the dictionary
- The list of the values is a view of the dictionary, meaning that any changes done to the dictionary will be reflected in the values list.

```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.values()

print(x) #before the change

car["year"] = 2020

print(x) #after the change
```

```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.values()

print(x) #before the change

car["color"] = "red"

print(x) #after the change
```

GET ITEMS

- The **items()** method will return each item in a dictionary, as tuples in a list
- The returned list is a view of the items of the dictionary, meaning that any changes done to the dictionary will be reflected in the items list

```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.items()

print(x) #before the change

car["year"] = 2020

print(x) #after the change
```

```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.items()

print(x) #before the change

car["color"] = "red"

print(x) #after the change
```

CHECK IF KEY EXISTS

To determine if a specified key is present in a dictionary use the **in** keyword

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
if "model" in thisdict:
  print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

CHANGE DICTIONARY ITEMS

You can change the value of a specific item by referring to its key name

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["year"] = 2018
```

UPDATE DICTIONARY

- The **update()** method will update the dictionary with the items from the given argument.
- The argument must be a dictionary, or an iterable object with key:value pairs.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.update({"year": 2020})
```

ADD DICTIONARY ITEMS

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

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

UPDATE DICTIONARY

- The **update()** method will update the dictionary with the items from a given argument. If the item does not exist, the item will be added.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.update({"color": "red"})
```

REMOVE DICTIONARY ITEMS

There are several methods to remove items from a dictionary

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

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

- The **popitem()** method removes the last inserted item

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.popitem()
print(thisdict)
```

REMOVE DICTIONARY ITEMS

- The **del** keyword removes the item with the specified key name

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict["model"]
print(thisdict)
```

- The **clear()** method empties the dictionary

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.clear()
print(thisdict)
```

NESTED DICTIONARIES

A dictionary can contain dictionaries, this is called nested dictionaries.

```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}
```

```
child1 = {
  "name" : "Emil",
  "year" : 2004
}
child2 = {
  "name" : "Tobias",
  "year" : 2007
}
child3 = {
  "name" : "Linus",
  "year" : 2011
}

myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}
```

ACCESS ITEMS IN NESTED DICT.

To access items from a nested dictionary, you use the name of the dictionaries, starting with the outer dictionary

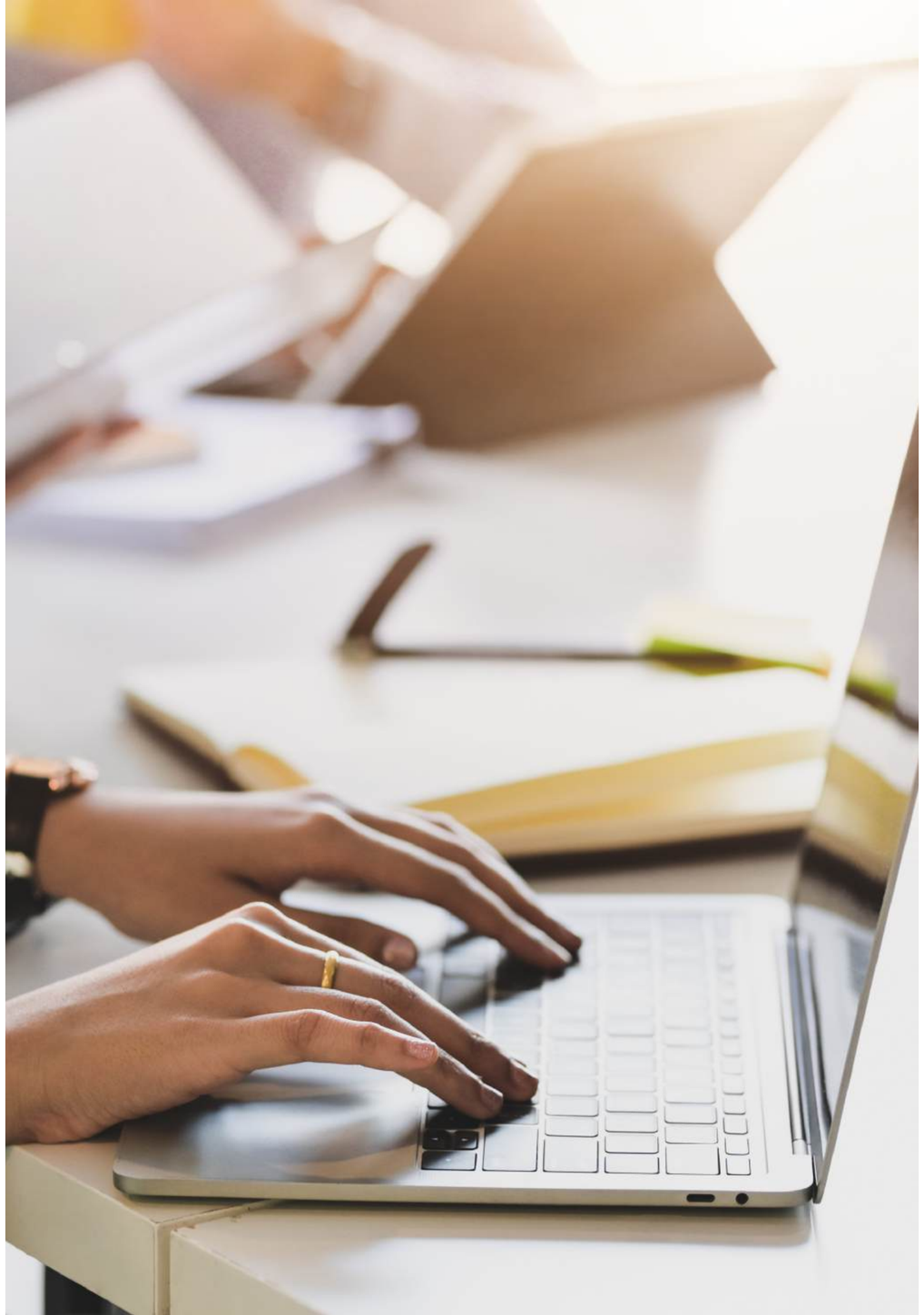
```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}

print(myfamily["child2"]["name"])
```

DICTIONARY EXERCISE

Use the get method to print the value of the "model" key of the car dictionary.

```
car =   {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(                    )
```



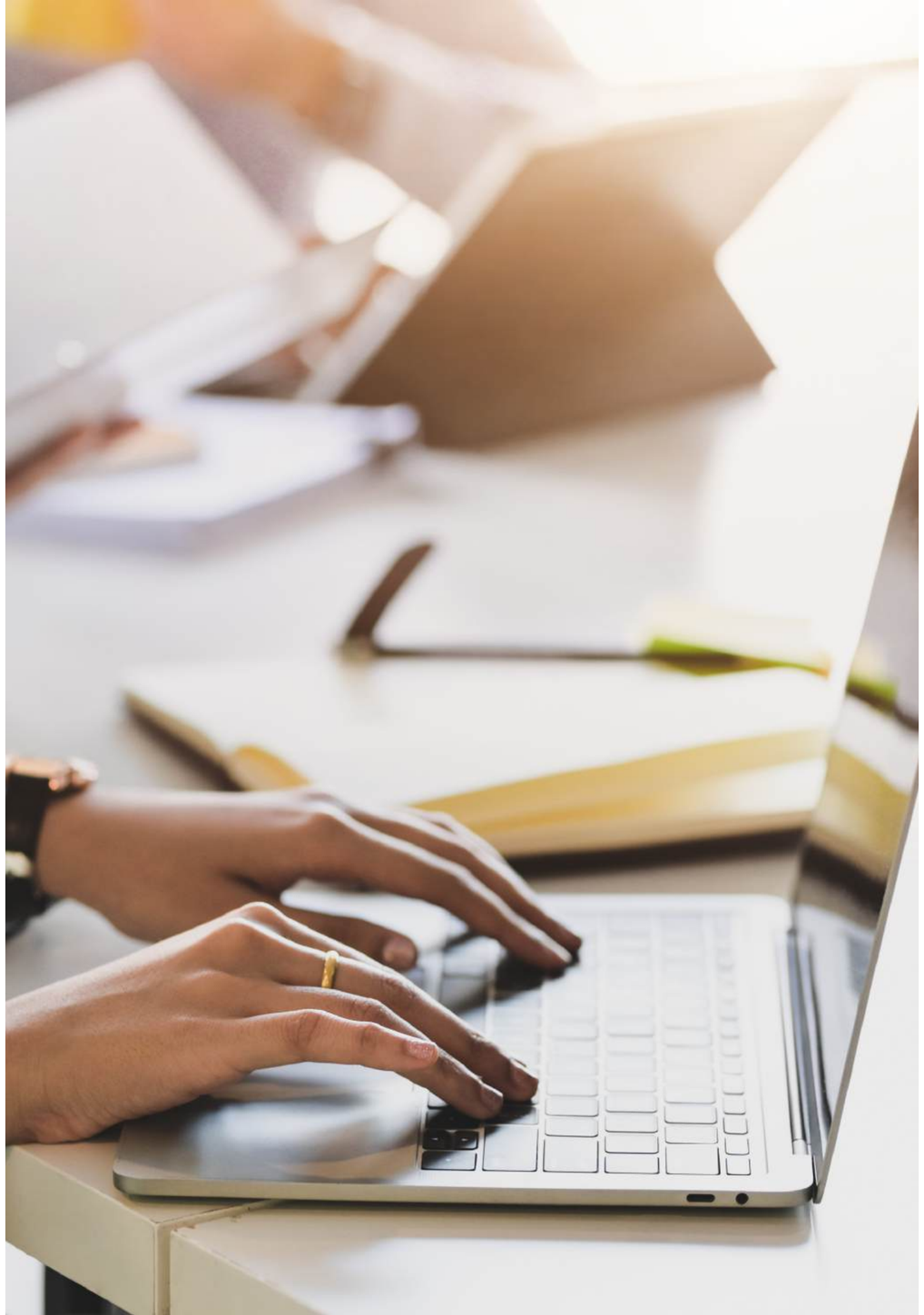
TASKS

Beginner Level :

- List Creation and Indexing: Create a list of 5 favorite fruits and print the first and last fruit using indexing.

- Tuple Usage: Define a tuple with 3 cities you want to visit. Attempt to change one city in the tuple to observe that tuples are immutable.

- Dictionary Basics: Create a dictionary to store 3 key-value pairs, where the keys are subjects (like Math, Science, English) and the values are your scores in each. Print the score of a specific subject.



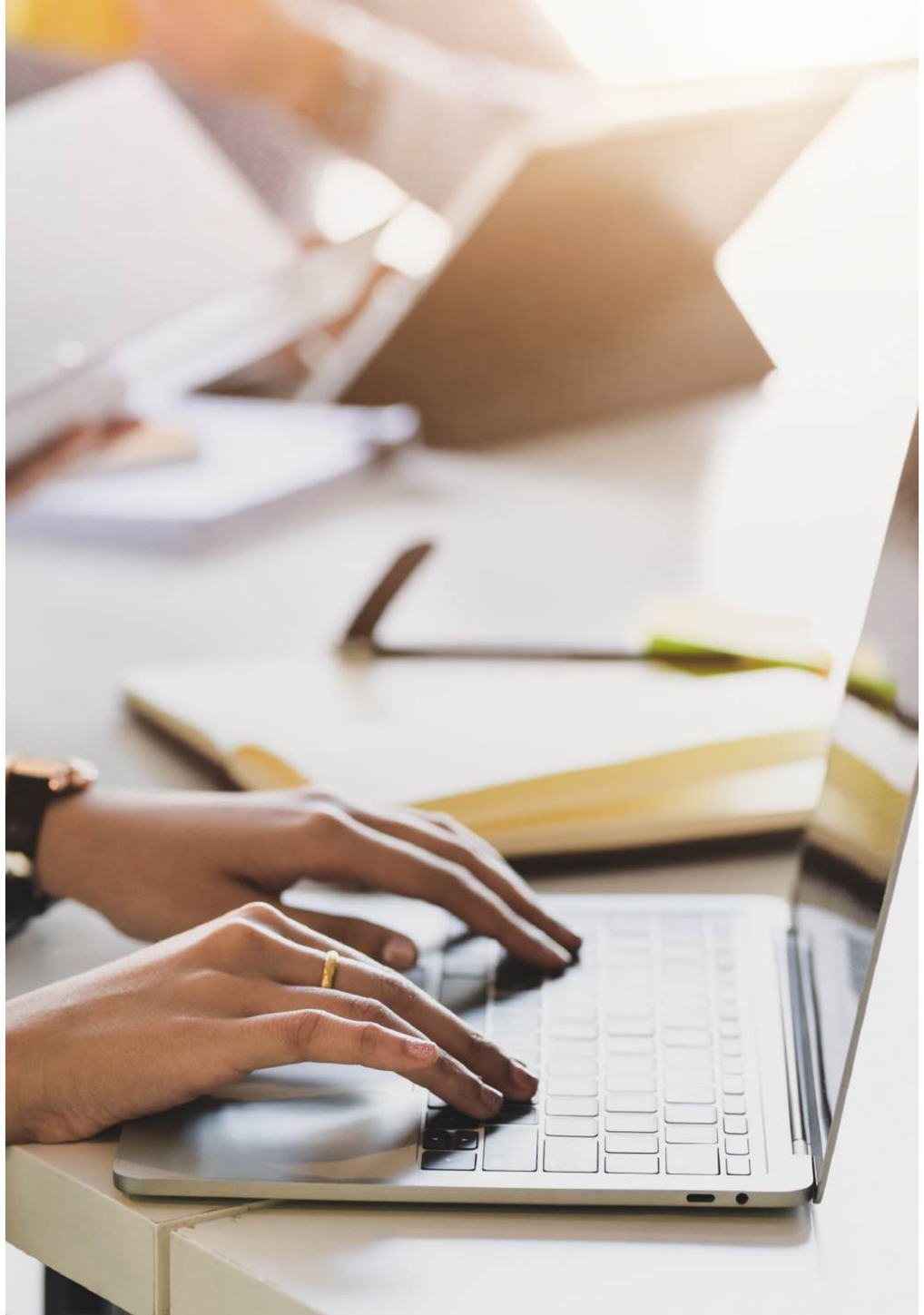
TASKS

Intermediate Level :

- List Operations: Create a list of 10 random numbers. Add 2 more numbers to the list, remove one number, and then sort the list in ascending order.

- Tuple Unpacking: Given a tuple (10, 20, 30), unpack its values into three variables a, b, and c, and print them.

- Dictionary Manipulation: Create a dictionary with 4 items where the keys are student names and the values are their ages. Update the age of one student and remove another student from the dictionary.



TASKS

Advanced Level :

- Nested Lists: Create a list that contains 3 lists, where each inner list contains 3 integers. Manually calculate and print the sum of each inner list.

- Tuples as Dictionary Keys: Create a dictionary where the keys are tuples representing coordinates (x, y) on a grid, and the values are the names of objects located at those coordinates. Access the object at a specific coordinate by using its tuple key.

- Complex Dictionary Operations: Create a dictionary that maps student names to a list of their scores in 3 subjects. Manually calculate and print the average score for each student.

ANY QUESTIONS ?

PYTHON PROGRAM

THANK YOU

UPCOMING NEXT WEEK : SESSION (2)