

Learning Objectives

1. Tuples in Python.
2. Sets in Python.
3. Dictionaries in Python.
4. Lab Tasks.

Python Tuples

- Tuples are used to store multiple items in a single variable.
- 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.

Python Tuples

{x}

```
✓ 1s  color = ("Black", "White", "Yellow")
    print(color)

    ('Black', 'White', 'Yellow')

✓ 0s  [2] print(type(color))

    <class 'tuple'>
```

Python - Access Tuple Items

- You can access tuple items by referring to the index number, inside square brackets.
- You can specify a range of indexes by specifying where to start and where to end the range.

Python - Access Tuple Items

```
✓ 0s  [5] print(color[2])

    Yellow

✓ 0s  [6] if "Yelloww" in color:
            print("Yellow color is in tuple")
        else:
            print("Yellow is not in tuple")

    Yellow is not in tuple
```

Python - Update Tuples

- Tuples are unchangeable, meaning that you cannot change, add, or remove items once the tuple is created.
- 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.

- Python - Update Tuples

```
[7]: color_list = list(color)
      color_list[0] = "Blue"
      color = tuple(color_list)
      print(color)
```

('Blue', 'White', 'Yellow')

```
[8]: color_list = list(color)
      color_list.append("Black")
      color = tuple(color_list)
      print(color)
```

↳ ('Blue', 'White', 'Yellow', 'Black')

Python - Unpack Tuples

- When we create a tuple, we normally assign values to it. This is called "packing" a tuple.
- But, in Python, we are also allowed to extract the values back into variables. This is called "unpacking".
- 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.
- 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.

- Python - Unpack Tuples

```
[9]: (color1, color2, color3, color4) = color
      print(color4)
```

Black

```
[10]: (color1, *colors) = color
      print(color1)
      print(colors)
```

Blue
['White', 'Yellow', 'Black']

```
[11]: (color1,*colors, color4) = color
      print(colors)
```

↳ ['White', 'Yellow']

Python - Loop Tuples

- You can loop through the tuple items by using a `for` loop.
- You can also loop through the tuple items by referring to their index number.
- Use the `range()` and `len()` functions to create a suitable iterable.

‐ Python - Loop Tuples

```
✓ [12] for value in color:  
    print(value)
```

```
Blue  
White  
Yellow  
Black
```

```
✓ [13] for i in range(len(color)):  
    print(color[i])
```

```
Blue  
White  
Yellow  
Black
```

```
✓ [14] i = 0  
while i in range(len(color)):  
    print(color[i])  
    i = i + 1
```

```
Blue  
White  
Yellow  
Black
```

Python - Join Tuples

- To join two or more tuples you can use the `+` operator.
- If you want to multiply the content of a tuple a given number of times, you can use the `*` operator.

‐ Python - Join Tuples

```
✓ [15] color_new = ("Purple", "Orange")  
color_join = color + color_new  
print(color_join)
```



```
('Blue', 'White', 'Yellow', 'Black', 'Purple', 'Orange')
```

```
✓ [16] color_multiply = color * 2  
print(color_multiply)
```



```
('Blue', 'White', 'Yellow', 'Black', 'Blue', 'White', 'Yellow', 'Black')
```

Python Sets

- Sets are used to store multiple items in a single variable.
- Sets are written with **curly brackets**.
- *A set is a collection which is unordered, unchangeable, unindexed and do not allow duplicate values.*

Python - Access Set Items

Access Items

- You cannot access items in a set by referring to an index or a key.
- But you can loop through the set items using a **for** loop, or ask if a specified value is present in a set, by using the **in** keyword.

The screenshot shows a Jupyter Notebook interface with three code cells:

- Cell 1:** Prints a set of three fruits.

```
[1]: thisset = {"apple", "banana", "cherry"}  
print(thisset)  
  
{'apple', 'cherry', 'banana'}
```
- Cell 2:** Prints each fruit in the set using a for loop.

```
[2]: thisset = {"apple", "banana", "cherry"}  
for x in thisset:  
    print(x)  
  
apple  
cherry  
banana
```
- Cell 3:** Checks if "banana" is in the set.

```
[3]: thisset = {"apple", "banana", "cherry"}  
print("banana" in thisset) #Check if "banana" is present in the set  
  
True
```

Python - Add Set Items

Add Items

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

Add Sets

- To add items from another set into the current set, use the `update()` method.

Add Any Iterable

- The object in the `update()` method does not have to be a set, it can be any iterable object (tuples, lists, dictionaries etc.)

Python - Add Set Items

Add Items

```
Os [4] thisset = {"apple", "banana", "cherry"}  
      thisset.add("orange")  
      print(thisset)  
  
{'apple', 'cherry', 'orange', 'banana'}
```

Add Sets

```
Os [5] thisset = {"apple", "banana", "cherry"}  
      tropical = {"pineapple", "mango", "papaya"}  
      thisset.update(tropical)  
      print(thisset)  
  
{'apple', 'cherry', 'papaya', 'mango', 'pineapple', 'banana'}
```

Add Any Iterable

```
Os [6] thisset = {"apple", "banana", "cherry"}  
      mylist = ["kiwi", "orange"]  
      thisset.update(mylist)  
      print(thisset)  
  
{'cherry', 'apple', 'kiwi', 'orange', 'banana'}
```

Python - Remove Set Items

Remove Item

- To remove an item in a set, use the `remove()`, or the `discard()` method.
- **Note:** If the item to remove does not exist, `remove()` will raise an error.
- **Note:** If the item to remove does not exist, `discard()` will NOT raise an error.

Python - Loop Sets

Loop Items

- You can loop through the set items by using a `for` loop

▼ Python - Remove Set Items

Remove Item

```
✓ 0s [7] thisset = {"apple", "banana", "cherry"}  
    thisset.remove("banana")  
    print(thisset)  
  
{'apple', 'cherry'}
```

Remove Item using discard

```
✓ 0s ① thisset = {"apple", "banana", "cherry"}  
    thisset.discard("banana")  
    print(thisset)  
  
⇒ {'apple', 'cherry'}
```

+ Code + Text

▼ Python - Loop Sets

Loop Items

```
✓ 0s [9] thisset = {"apple", "banana", "cherry"}  
    for x in thisset:  
        print(x)  
  
apple  
cherry  
banana
```

Python 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**.

Python - Access Dictionary Items

Accessing Items

- You can access the items of a dictionary by referring to its key name, inside square brackets
- There is also a method called **get()** that will give you the same result.

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.

Python Dictionaries

```
[10] thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964,
    "year": 2020
}
print(thisdict)
print(thisdict["brand"]) #Print the "brand" value of the dictionary
{'brand': 'Ford', 'model': 'Mustang', 'year': 2020}
Ford
```

Python - Access Dictionary Items

Accessing Items

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

Mustang

get() method

```
[12] x = thisdict.get("model")
print(x)
```

Mustang

Get Keys

```
[13] x = thisdict.keys()
print(x)
```

dict_keys(['brand', 'model', 'year'])

Python - Change Dictionary Items

Change Values

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

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.

Python - Change Dictionary Items

Change Values

```
[14] thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
thisdict["year"] = 2018  
print(thisdict)  
  
{'brand': 'Ford', 'model': 'Mustang', 'year': 2018}
```

Update Dictionary

```
[15] thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
thisdict.update({"year": 2020})  
print(thisdict)  
  
{'brand': 'Ford', 'model': 'Mustang', 'year': 2020}
```

Python - Add Dictionary Items

Adding Items

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

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.
- The argument must be a dictionary, or an iterable object with key:value pairs.

Python - Add Dictionary Items

Add Items

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

update() Method

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

Python - Remove Dictionary Items

Removing Items

- There are several methods to remove items from a dictionary
- The `pop()` method removes the item with the specified key name.
- The `popitem()` method removes the last inserted item (in versions before 3.7, a random item is removed instead).
- The `del` keyword removes the item with the specified key name.
- The `del` keyword can also delete the dictionary completely.
- The `clear()` method empties the dictionary.

Python - Remove Dictionary Items

Removing Items

```
[18] thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
thisdict.pop("model") #pop Method  
print(thisdict)  
  
{'brand': 'Ford', 'year': 1964}
```

```
[19] thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
thisdict.popitem() #popitem Method  
print(thisdict)  
  
{'brand': 'Ford', 'model': 'Mustang'}
```

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

```
[21] thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
thisdict.clear() #clear method  
print(thisdict)  
  
{}
```

Lab Tasks:**Task # 1:**

Write a Python script to print a dictionary where the keys are numbers between 1 and 10 (both included) and the values are square of keys. Sample Dictionary: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}

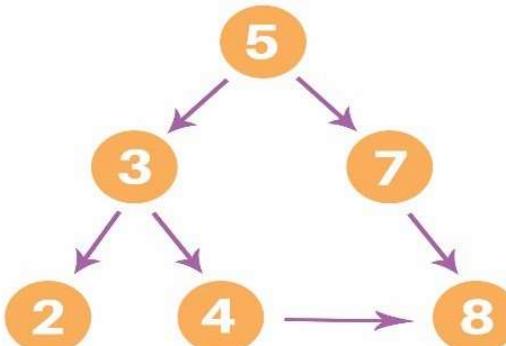
Task # 2:

Create a simple book inventory system using tuples. Each book in the inventory should be represented as a tuple containing the following information:

- Title
 - Author
 - Publication Year
 - ISBN (International Standard Book Number)
 - Quantity in Stock
1. Define an empty list named `book_inventory` to store tuples representing books.
 2. Implement a function `add_book` that takes input for a new book and adds it to the `book_inventory`. The function should prompt the user for the book details (title, author, publication year, ISBN, quantity) and create a tuple to represent the book. Append this tuple to the `book_inventory`.
 3. Implement a function `search_book` that takes the ISBN as input and searches for the book in the inventory. If the book is found, print its details; otherwise, print a message indicating that the book is not in the inventory.
 4. Implement a function `display_inventory` that prints the details of all books in the inventory. Each book's information should be printed on a new line.
 5. Implement a function `sell_book` that takes the ISBN and quantity as input and updates the stock quantity of the corresponding book in the inventory. If the quantity is greater than the available stock, print a message indicating insufficient stock.
 6. Test your inventory system by adding a few books, searching for them, displaying the inventory, and selling books.

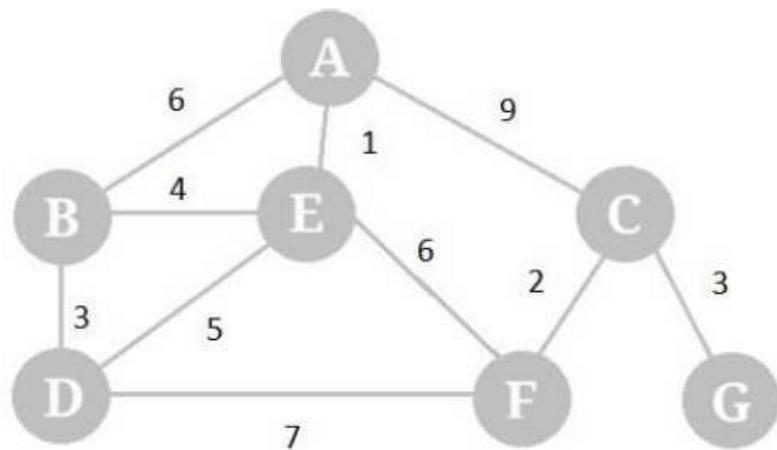
Task # 3:

Add the following graph into the dictionary.



Task # 4:

Add the following graph into the dictionary.



Lab # 02 Marks distribution

	ER1	ER6	ER8
Task	3 points	3 points	4 points

Lab # 02 Rubric Evaluation Guideline:

#	Qualities & Criteria	0 < Poor <= 1	1 < Satisfactory <= 2	2 < Excellent <=3
ER1	Task Completion	Minimal or no program functionality was achieved.	Some tasks were completed, but the program has errors or incomplete functionalities.	All tasks were completed, and the program runs without errors.
#	Qualities & Criteria	0 < Poor <= 1	1 < Satisfactory <= 2	2 < Excellent <=3
ER6	Program Output	Output is inaccurate or poorly presented.	Output is mostly accurate but may lack labels, captions, or formatting.	Output is clear, accurate, and well presented with labels, captions, and proper formatting.
#	Qualities & Criteria	0 < Poor <= 1	1 < Satisfactory <= 2.5	2.5 < Excellent <= 4
ER8	Question & Answer	Answers some questions but not confidently or based on lab task knowledge.	Answers most questions confidently and based on lab task knowledge.	Answers all questions confidently and demonstrates a deep understanding of the given lab task.