Day 10 (10/08/2023): (Tuples, Set, Dict)

1) Tuples:
Tuples are used to store multiple values in a single variable.
Tuples are created by parenthesis or round bracket. (Different from List)
Tuples are ordered.
Tuples allow duplicates.
Tuples are unchangeable (immutable). (Different from List)
Tuples need more than one element. If not, it will be considered as string.

Code:
```
nums=("integers","real","decimal","imaginery")
print(type(nums)) #ordered,allow duplicates
nums[2]="float"
print(nums) #unchangable
```

Output:
```
<class 'tuple'>
TypeError: 'tuple' object does not support item assignment
```

2) Sets:
Sets are used to store multiple items in a single variable.
Sets are created by curly brackets. (Different from List)
Sets are unordered. (Different from List)
Sets don’t allow duplicates. (Different from List)
Set is unchangeable (Immutable). (Different from List)

Code:
```
set={"Takes","Fandom","Brains","Fork","Fork"}
print(type(set))
print(set) #multiplevalues/items,unordered,noduplicates,unchangable
set[1]="Tickets"
print(set)
```

Output:
```
<class 'set'>
{'Fork', 'Takes', 'Brains', 'Fandom'}
TypeError: 'set' object does not support item assignment
```

Set Constructor:
Set can also be constructed by defining a variable as a set before the entries of item/values.

Code:
```
set2=set(("Takes","Fandom","Brains","Fork","Fork"))
print(type(set2))
print(set2)
```

Output:
```
<class 'set'>
{'Fandom', 'Fork', 'Brains', 'Takes'}
```

Remove Duplicates from List:
Duplicates of a set or list can be removed by using *set() function.

Code:
```
list1=[21,25,65,21,59,25,21,26,59]
nodups=[*set(list1)]
print(nodups)
```

Output:
```
[65, 21, 25, 26, 59]
```

3) Dictionary (Mapping Type):
All the details of the users are entering from front to back end to database in the dictionary form.
Dictionaries are used to store data values in (key:value) pairs.
Dictionaries are created by curly brackets.
Dictionaries are ordered.
Dictionaries don't allow duplicates. (First duplicate will be deleted
While running the code.)
Dictionaries are changeable.

Code:
```
data={"int":55.58,"real":56,"dec":25.56,"flt":36.9,"flt":26.9}
print(type(data))
print(data)
```

Output:
```
<class 'dict'>
{'int': 55.58, 'real': 56, 'dec': 25.56, 'flt': 26.9}
```

Dictionary Key Value change: Value of a key can be changed by using key equating method.

Code:

```
data["flt"]=46.9
print(data)
```

Output:

{'int': 55.58, 'real': 56, 'dec': 25.56, 'flt': 46.9}

keys() & values():

Keys() & Values are used to identify all the keys, and the values of the dictionaries.

Code:

```
x=data.keys()
y=data.values()
print(x,y)
```

Output:

dict_keys(['int', 'real', 'dec', 'flt']) dict_values ([55.58, 56, 25.56, 46.9])

4) Logical:

Logical operators (AND/OR/NOT) are used to create several outcomes with different inputs on user input functions.

Code:

```
opr=input("Type of Arithmetic Operation (in words/symbols): ")
if opr=="ADD" or opr=="add" or opr=="Add" or opr=="+" or opr=="SUBTRACT" or
opr=="subtract" or opr=="Subtract" or opr=="-" or opr=="MULTIPLY" or
opr=="Multiply" or opr=="multiply" or opr == "*" or opr=="x" or opr=="X" or
opr=="Divide" or opr=="DIVIDE" or opr=="divide" or opr=="/":
    print(f"{opr}:")
    g=int(input("Enter the First Value: "))
    h=int(input("Enter the Second value: "))
    add=g+h
    subtract=g-h
    multiply=g*h
    divide=g/h
    if opr=="add" or opr=="ADD" or opr=="Add" or opr=="+":
        print(add)
    elif opr=="subtract" or opr=="SUBTRACT" or opr=="Subtract" or opr=="-":
        print(subtract)
    elif opr=="multiply" or opr=="Multiply" or opr=="Multiply" or opr=="x" or
opr=="X" or opr=="*":
        print(multiply)
    elif opr=="divide" or opr=="DIVIDE" or opr=="Divide" or opr=="/":
        print(divide)
```

Output:
Type of Arithmetic Operation (in words/symbols): /
Enter the First Value: 36
Enter the Second value: 5
7.2

5) Boolean (Bool):
Boolean is used to perform an action on the basis of if conditioning using (True/False).

Code:
```
r=True
s=False
if r:
    print("The Statement is true")
else:
    print("The Statement is False")
print(type(r))
print(r)
```

Output:
The Statement is true
<class 'bool'>
True

6) None Type:
None type is used in the places where the values are not intended to be give as an input to a variable. None type instructs the code to take the variable value as nothing/empty.

Code:
```
z=None
print(type(z))
print(z)
```

Output:
6) None Type:
<class 'NoneType'>
None

Type Conversion: Items can be changed by converting the tuple into list by using the method of type conversion. From one type to another: Use data.split(“,”) [in case of splitting based on comma] to create a data. Split will always create a list.

Len():
It shows how many datas inside a file.

Count():
It helps us to count the wanted files.