Python Assignment No 1

Pratik Wani

- Datatypes:
  - Numeric Types:
    - int: Integer type.
    - float: Floating-point type
  - Boolean:
    - bool: Boolean type (True or False)
  - Sequence:
    - List: List type
    - Set: Set type
    - Tuple: Tuple type
    - Dict: Dictionary type

Practice.py | operators.py | Datatypes.py | file.txt

Datatypes.py > ...

```python

#Numeric datatypes
x = 5
y = 3.14

print(type(x))
print(type(y))
print("\n*******************")

#boolean
flag = True

print(type(flag))
print("\n*******************")


#Sequence
L= [1, 2, 3, "four"]
T= (1, 2, 3, "four")
S= {1, 2, 3, 3, 2, 1}
D= {"name": "John", "age": 25}

print(type(L))
print(type(T))
print(type(S))
print(type(D))
```

```
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder> & C:/Users/de
s/dell/OneDrive/Desktop/Pratik Wani/Data Engineering/New folder/Datatypes.py"
<class 'int'>
<class 'float'>

*******************
<class 'bool'>

*******************
<class 'list'>
<class 'tuple'>
<class 'set'>
<class 'dict'>
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder>
```

- Operators:
  - Arithmetic Operators:
    - + → addition
    - - → subtraction
    - * → multiplication
    - / → division
    - % → modulus
    - ** → exponentiation
  - Comparison Operators:
    - == → equal to
    - != → not equal to
    - < → less than
    - > → greater than
    - <= → less than or equal to
    - >= → greater than or equal to
  - Bitwise Operators:
    - & → bitwise AND
    - | → bitwise OR
    - ^ → bitwise XOR
    - ~ → bitwise NOT
    - << → left shift
    - >> → right shift

Practice.py | operators.py | file.txt

operators.py > ...

```python

#Arithmetic Operators:
a = 10
b = 30
print(a+b)
print(a-b)
print(a*b)
print(a/b)
print(a%b)
print(a**b)
print(a//b)

#Comparison Operators:
x=50
y=20
print(x==y)
print(x≠y)
print(x<y)
print(x>y)
print(x≤y)
print(x≥y)

#bitwise operators:

a=10
b=12

print(a&b)
print(a|b)
print(a^b)
print(~a)
print(a<<1)
print(b>>1)

```

Ln 33, Col 10 Space

TERMINAL

```
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder> & C:/Users/dell/
s/dell/OneDrive/Desktop/Pratik Wani/Data Engineering/New folder/operators.py"
40
-20
300
0.3333333333333333
10
1000000000000000000000000000000
0
******************
False
True
False
True
False
True
******************
8
14
6
-11
20
6
******************
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder>
```

- Conditions:
    - If statements:
        - used to execute a block of code if a certain condition is True
    - If – Else statements:
        - used to execute one block of code if the condition is True and another block of code if the condition is False
    - If – Elif – Else statements:
        - used when you have multiple conditions to check.
        - It checks the blocks one by one if condition is 'true' it will execute that block
        - Otherwise it will execute the else block

Practice.py | operators.py | Datatypes.py | Conditional_statement.py | file.

Conditional_statement.py > ...

```python
#If statement

var='A'

if var=='A':
    print(f"value of var is {var}")

#If - Else statement

num=15

if num%2==0:
    print("num is even")
else:
    print("num is odd")

#If - Elif - Else

check=0

if check>0:
    print("check is positive")
elif check<0:
    print("check is negative")
else:
    print("check is zero")
```

PROBLEMS | OUTPUT | DEBUG CONSOLE | TERMINAL

TERMINAL

```
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder> & C:/Users/d
s/dell/OneDrive/Desktop/Pratik Wani/Data Engineering/New folder/Conditional_statement.p
value of var is A
num is odd
check is zero
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder>
```

- Control Structure:
  - For loop:
    - For loop is use to iterate over a specific range or sequence like list, tuple, string
  - While loop:
    - continues to execute a block of code as long as a specified condition is True
  - While – Else loop:
    - the else block in a while loop is executed when the loop condition becomes False.
    - If while loop prematurely exits then only else block is skip

Control_Structure.py > ...

```python
#for loop
for i in range(6):
    print(i)

print("\n**********************")

L=['a','b','c','d','e']

for i in L:
    print(i)
print("\n**********************")

#while loop
flag=0

while flag<6:
    flag+=2
    print(flag)

print("\n**********************")

#while - else
flag=0

while flag<6:
    flag+=2
    print(flag)
else:
    print("The final value of flag is "+ flag)

print("\n**********************")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

TERMINAL

```
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder> & C:/Users/dell/App
s/dell/OneDrive/Desktop/Pratik Wani/Data Engineering/New folder/Control_Structure.py"
0
1
2
3
4
5

**********************
a
b
c
d
e

**********************
2
4
6

**********************
2
4
6
The final value of flag is  6

**********************
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder>
```

- Break- Continue- Pass:
  - Pass:
    - Null operation
  - Continue:
    - The current iteration stops and next iteration will start executing
  - Break:
    - Terminate the loop

Practice.py | operators.py | Datatypes.py | Conditional_statement.

Break-Cont-Pass.py > ...

```python
#pass statement
for i in range(0,8):
    pass
print("No action taken!!")
print("\n********************")


# continue statement
for i in range(0,5):
    if i==2:
        continue
    print("Printing only odd values ", i)


# break statement

for i in range(0,10):
    if i==7:
        break
    print(i)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

TERMINAL

```
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder> & C:/Users/
s/dell/OneDrive/Desktop/Pratik Wani/Data Engineering/New folder/Break-Cont-Pass.py"
No action taken!!

********************
Printing only odd values  0
Printing only odd values  1
Printing only odd values  3
Printing only odd values  4
0
1
2
3
4
5
6
```

- String Functions:

stringfunction.py > ...

```python

var="pratik"

#length
len=len(var)
print(len)

#captal and lower
capital=var.capitalize()
lower=var.lower()
print(capital)
print(lower)

#split
text="Pratik,Arun,Wani"
Name=text.split(',')
print(Name)

#replace
new_var=var.replace('pra', 'ru')
print(new_var)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

TERMINAL

```
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder> & C:/Use
s/dell/OneDrive/Desktop/Pratik Wani/Data Engineering/New folder/stringfunction.py"
6
Pratik
pratik
['Pratik', 'Arun', 'Wani']
rutik
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder>
```

- Number Functions

stringfunction.py | NumberFunction.py

NumberFunction.py > ...

```python
#roundup value
x=1.5
x=round(x)
print(x)
print("\n********")
#absolute value
x=-3
print(abs(x))
print("\n********")

#power
x=3
y=2

print(pow(x,y))
print("\n********")

# Min and Max

print(min(x,y))
print("\n********")

print(max(x,y))
print("\n********")
```

```
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder> & C:/Us
s/dell/OneDrive/Desktop/Pratik Wani/Data Engineering/New folder/NumberFunction.py"
2

********
3

********
9

********
2

********
3

********
```

- Map function and Lambda expression:

  - Map:
    - The map function is used to apply a specified function to all items in an iterate able.
  - Lambda:
    - also known as anonymous functions
    - they don't have a name like regular functions defined with the def keyword

stringfunction.py | NumberFunction.py | map_Lambda.py

map_Lambda.py > ...

```python
#map

L=['1','2','3']

new_L=list(map(int,L))

print(new_L[0]+new_L[1])

print("\n*********")


#lambda


numbers=[1,2,3,4]

squared_numbers=map(lambda x: x**2, numbers)

print(list(squared_numbers))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

TERMINAL

```
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder> & C
s/dell/OneDrive/Desktop/Pratik Wani/Data Engineering/New folder/map_Lambda.py"
3

*********
[1, 4, 9, 16]
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder>
```

- Functions:
    - Functions starts with keyword def
    - Function is block of code which can be reuse
    - It will make code more readable also reduce the lines of codes

stringfunction.py | NumberFunction.py | map_Lambda.py | Practice.py | function.py

function.py > ...

```python
from math import sqrt
def checkprime(number):
    flag=1
    if number>1:
        for i in range(2,int(sqrt(number))+1):
            if(number%i==0):
                flag=0
        if(flag==0):
            return False
        else:
            return True

    else:
        return False

number=int(input("Enter the number: "))
if checkprime(number):
    print(f"{number} is prime number")
else:
    print(f"{number} is not prime number")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

TERMINAL

```
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder> & C:/Users/dell/AppData/Loc
s/dell/OneDrive/Desktop/Pratik Wani/Data Engineering/New folder/function.py"
Enter the number: 10
10 is not prime number
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder>
```

- List methods and slicing:
    - List can store multiple elements of different datatypes
    - List are mutable and iterate able
    - List Methods:
        - Append: Add an Element to the End
        - Insert: Insert an Element at a Specific Position
        - Remove: Remove First Occurrence of a Value
        - Pop: Remove Last Element
        - Sort: Sort the list

stringfunction.py | NumberFunction.py | map_Lambda.py | Control_Structure.py | Lists.py

Lists.py > ...

```python
my_list=[1,2,3,4,5]
#append
my_list.append(4)
print(my_list)

#pop
popped_element=my_list.pop()
print(my_list)

#insert
my_list.insert(0,0)
print(my_list)

#length
print(len(my_list))

#sort
my_list.sort()
print(my_list)

#slicing
print(my_list[0:3])
print(my_list[-2:-1])
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

TERMINAL

```
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder> & C:/Users/dell/AppD
s/dell/OneDrive/Desktop/Pratik Wani/Data Engineering/New folder/Lists.py"
[1, 2, 3, 4, 5, 4]
[1, 2, 3, 4, 5]
[0, 1, 2, 3, 4, 5]
6
[0, 1, 2, 3, 4, 5]
[0, 1, 2]
[4]
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder>
```

Ln 5, Col 1 Spa

- Tuple:
    - tuple is an ordered, immutable collection of elements.
    - they cannot be modified once created.
    - Tuple methods:
        - Count: Count Occurrences
        - Index: Find Index of First Occurrence

stringfunction.py | NumberFunction.py | map_Lambda.py | Control_Structure.py | Lists.py

tuple.py > ...

```python
my_tuple = (1,2,3,"pratik")
print(my_tuple[0])

#slicing
sub=my_tuple[0:2]
print(sub)

#count
count=my_tuple.count(2)
print(count)

#index
index=my_tuple.index(2)
print(index)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

TERMINAL

```
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder> & C:/Users/d
s/dell/OneDrive/Desktop/Pratik Wani/Data Engineering/New folder/tuple.py"
1
(1, 2)
1
1
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder>
```

- Dictionary:
    - Dictionaries are collections of key-value pairs
    - key must be unique
    - defined using curly braces {}
    - Dictionary methods:
    - Keys: Get Keys
    - Values: Get Values
    - Items: Get Key-Value Pairs
    - Update: Update Dictionary

Dict.py > ...

```python
my_dict={"name": "pratik",
         "age": 21,
         "city": "Nashik"}

print(my_dict)
print("\n********")

#keys
print(my_dict.keys())
print("\n********")

#values
print(my_dict.values())
print("\n********")

#items
print(my_dict.items())
print("\n********")

#update
my_dict.update({"occupation": "Engineer"})
print(my_dict)
print("\n********")
```

TERMINAL

```
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder> python -u "c:\Us
 folder\Dict.py"
{'name': 'pratik', 'age': 21, 'city': 'Nashik'}

********
dict_keys(['name', 'age', 'city'])

********
dict_values(['pratik', 21, 'Nashik'])

********
dict_items([('name', 'pratik'), ('age', 21), ('city', 'Nashik')])

********
{'name': 'pratik', 'age': 21, 'city': 'Nashik', 'occupation': 'Engineer'}

********
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder>
```

- Sets:
    - Unordered collections of unique elements.
    - Defined using curly braces {}
    - Set methods:
    - Add: To add a single element
    - Update: To add multiple elements
    - Remove: To remove an element
    - Union: Add to sets
    - Intersection: Common elements in sets

Dict.py | sets.py

sets.py > ...

```python
my_set1={1, 2, 3, 3, 4, 5}

my_set2={3, 4, 5, 6}
print(my_set1)

#add
my_set1.add(6)
print(my_set1)

#update
my_set1.update([7, 8, 9])
print(my_set1)

#remove
my_set1.remove(3)
print(my_set1)

#union
new_uset=my_set1.union(my_set2)
print(new_uset)

#intersection
new_iset=my_set1.intersection(my_set2)
print(new_iset)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

TERMINAL

```
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder> & C:/Users/d
s/dell/OneDrive/Desktop/Pratik Wani/Data Engineering/New folder/sets.py"
{1, 2, 3, 4, 5}
{1, 2, 3, 4, 5, 6}
{1, 2, 3, 4, 5, 6, 7, 8, 9}
{1, 2, 4, 5, 6, 7, 8, 9}
{1, 2, 3, 4, 5, 6, 7, 8, 9}
{4, 5, 6}
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder>
```

- Exception Handling:
    - exceptions are events that occur during the execution of a program that stops the normal flow of code.
    - The program that can handle the exception to prevent the program known as exception handling
    - Python is done using the try, except, else, and finally blocks

Dict.py | sets.py | Exception.py | Practice.py | excetion.py

excetion.py > ...

```python
try:
    a=int(input("Enter the a: "))
    b=int(input("Enter the b: "))
    result=a/b
except ZeroDivisionError:
    print("Cannot divide by zero.")
else:
    print("Division successful.")
finally:
    print("This will always execute.")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

TERMINAL

```
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder> & C:/Users/dell
s/dell/OneDrive/Desktop/Pratik Wani/Data Engineering/New folder/excetion.py"
Enter the a: 10
Enter the b: 0
Cannot divide by zero.
This will always execute.
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder> & C:/Users/dell
s/dell/OneDrive/Desktop/Pratik Wani/Data Engineering/New folder/excetion.py"
Enter the a: 10
Enter the b: 5
Division successful.
This will always execute.
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder>
```

- OOP:
    - It is a Programming technique that uses classes and objects, for designing and organizing code.
    - Class:
        - Blueprints for the objects
    - Objects:
        - The real world entity around which our code is going to be revolved
    - OOP's Pillers:
        - Inheritance: allows a derived class to inherit the attributes and methods of a base class
        - Polymorphism: allows us to create same function name but having different signatures
        - Encapsulation: prevents the accidental modification of data.

Dict.py | sets.py | Exception.py | Practice.py | excetion.py | oop.py

oop.py > ...

```python

class Car:
    def __init__(self,brand,model):
        self.__private_attr='Car'
        self.brand=brand
        self.model=model

    def display_info(self):
        print(f'''Brand Name: {self.brand}\nModel Name: {self.model}''')


C1=Car('Tata','Punch')
C2=Car('Audi','A4')

C1.display_info()
C2.display_info()

```

Dict.py | sets.py | Exception.py | Practice.py | excetion.py | oop.py

oop.py > SuperCar > display_info

```python
#inheritance and method overriding

print("*** Inheritance ***")

class SuperCar(Car):
    def __init__(self,brand,model,max_speed):
        super().__init__(brand,model)
        self.max_speed=max_speed

    def display_info(self):
        super().display_info()
        print(f'''Max speed: {self.max_speed}''')


C3=SuperCar('BMW','M1','270 km/h')
C3.display_info()

#encapsulation and data abstarction
#not accessible because private_attr is private attribute
# _ means protected (only accessible within base class and derived class)
# __ means private (only accessible within the base class)

print(C1.private_attr)
```

PROBLEMS | OUTPUT | DEBUG CONSOLE | TERMINAL

TERMINAL

```
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder> & C:/Users/dell/AppData/Local/Programs/P
s/dell/OneDrive/Desktop/Pratik Wani/Data Engineering/New folder/oop.py"
Brand Name: Tata
Model Name: Punch
Brand Name: Audi
Model Name: A4
*** Inheritance ***
Brand Name: BMW
Model Name: M1
Max speed: 270 km/h
*** Encapsulation ***
Traceback (most recent call last):
  File "c:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder\oop.py", line 43, in <module>
    print(C1.private_attr)
AttributeError: 'Car' object has no attribute 'private_attr'
PS C:\Users\dell\OneDrive\Desktop\Pratik Wani\Data Engineering\New folder>
```