

Python Experiment 1

Aim: To study & implement diff. data types & operations in python.

Learning: 1. Python allows variety of data types allowing flexibility in handling different kinds of data.

2. Structures like lists, tuple, dictionary are suitable for various use cases.
3. Lists are created using square brackets, they are ordered, changeable & allow duplicate values.
4. Tuple is a collection which is ordered & immutable written in parenthesis.
5. ~~Dict~~ Dictionary is a collection of key-values pairs, accessed using keys.
6. Set is unordered, immutable & unindexed.
7. Besides these, other data types are Boolean (True/False), None Type, int, float, complex, string, etc.

→ There are various operators to perform operations with variables. Among them are Arithmetic (+, -, *, /, %, **), Assignment op. (=, +=, -=, *=, /=, etc.) to assign values to variables, Comparison operators for comparison like (==, !=, >, <, >=, <=), Conditional operators for handling conditional statements like 'and', 'or', 'not'. 'is' & 'is not' are identity operators whereas OR (^), XOR (^), AND (&), NOT (~) zero fill left shift (<<), signed Right shift (>>) are bitwise logical operators.

Code (Experiment1) :

1) Data Types in Python :

```
a = "hello world"
print('\nStrings implemenation : ',a,type(a))

b = 2
print('\ninteger implemenation : ',b,type(b))

c = 29.87
print('\nfloat implemenation : ',c,type(c))

com = 15+98j
print('\nComplex numbers implemenation : ',com,type(com))

l = [1,'one',1.00,1+2j]
print('\nlist implemenation : ',l,type(l))

ran = range(10)
print('\nRange implemenation : ',ran,type(ran))

tup = ('abc',12,0.92)
print('\ntuple implemenation : ',tup,type(tup))

dic = {1:'hi',2:'you'}
print('\nDictionary implemenation : ')
print(dic,type(dic))
print('Dictionary values : ',dic.values())
print('Dictionary items : ',dic.items())
print('Dictionary keys : ',dic.keys())

s = {'hello','world','python'}
print('\nSet Implementation : ',s,type(s))

by = b'hi'
print('\nByte implemenation : ',by,type(by))

n = None
print('\nNone type : ',n,type(n))
```

```
id1 = 1223

print('id of id1 : ',id(id1),'is id of ',id1)
```

Output for above code :

```
PS D:\SEM-5\PYTHON\EXPERIMENTS> python -u "d:\SEM-5\PYTHON\EXPERIMENTS\EXP1\dataTypes.py"

Strings implemenation :  hello world <class 'str'>

integer implemenation :  2 <class 'int'>

float implemenation :  29.87 <class 'float'>

Complex numbers implemenation :  (15+98j) <class 'complex'>

list implemenation :  [1, 'one', 1.0, (1+2j)] <class 'list'>

Range implemenation :  range(0, 10) <class 'range'>

tuple implemenation :  ('abc', 12, 0.92) <class 'tuple'>

Dictionary implemenation :
{1: 'hi', 2: 'you'} <class 'dict'>
Dictionary values :  dict_values(['hi', 'you'])
Dictionary items :  dict_items([(1, 'hi'), (2, 'you')])
Dictionary keys :  dict_keys([1, 2])

Set Implementation :  {'world', 'hello', 'python'} <class 'set'>

Byte implemenation :  b'hi' <class 'bytes'>

None type :  None <class 'NoneType'>
id of id1 :  1973578753968 is id of  1223
PS D:\SEM-5\PYTHON\EXPERIMENTS>
```

2) Operators in Python :

```
a = 18
b = 3

print("\nArithmetic Operators :")
print(f'Addition of {a} and {b} is ',a+b)
print(f'subtraction of {a} and {b} is ',a-b)
print(f'Modulo of {a} and {b} is ',a%b)
print(f'Multiplication of {a} and {b} is ',a*b)
print(f'Division of {a} and {b} is ',a/b)
print(f'Remainder of {a} by {b} is ',a//b)
print(f'{a} raise to {b} is ',a**b)

c = 14

print("\nAssignment Operators :")
print("Inital value of c :")
```

```
print(c)

c+=6

print("Adding 6 to it gives ",c)

c-=6

print("subtracting 6 to it gives ",c)

c*=6

print("multiplying 6 to it gives ",c)

c/=6

print("divding by 6 to it gives ",c)

c%=6

print("taking modulo gives ",c)

c//=6

print("c raised to 6 gives ",c)

c**=6


d = 15
e = 10

print('\nComparison Operators :')

print(f"is {d} equal to {e}",d==e)

print(f"is {d} not equal to {e}",d!=e)

print(f"is {d} greater than or equal to {e}",d>=e)

print(f"is {d} smaller than or equal to {e}",d<=e)

print(f"is {d} greater than {e}",d>e)

print(f"is {d} smaller than {e}",d<e)


f = 10
g = 19

print('\nLogical Operators :')

print(f"if {f}>=10 and {g} <20 then : ", f>=10 and g<20)

print(f"if {f}>=10 or {g} >20 then : ", f>=10 or g>20)


h = 9
i = 19

print('\nidentity Operators :')

print(f"h is i ", h is i)

print(f"h is not i ", h is not i)


x = 'java'
y = 'javascript'

print("\nMembership Operators : ")
```

```
print(f'is {x} in {y} : ',x in y)
print(f'is {x} not in {y} : ',x not in y)

a = 5
b = 7

print("\nBitwise Operators : ")
print(f"{a} and {b} is : ",a&b)
print(f"{a} or {b} is : ",a|b)
print(f"{a} ^ {b} is : ",a^b)
print(f"{a} << 3 is : ",a<<3)
print(f"{b} >> 3 is : ",b>>3)
print(f"~ {a} is : ",~a)
print(f"~ {b} is : ",~b)
print(f"{a} >> 2 is : ",b>>2)
```

Output for above code :

```
PS D:\SEM-5\PYTHON\EXPERIMENTS> python -u "d:\SEM-5\PYTHON\EXPERIMENTS\EXP1\operators.py"
```

Arithmetic Operators :

```
Addition of 18 and 3 is 21
subtraction of 18 and 3 is 15
Modulo of 18 and 3 is 0
Multiplication of 18 and 3 is 54
Division of 18 and 3 is 6.0
Remainder of 18 by 3 is 6
18 raise to 3 is 5832
```

Assignment Operators :

```
Initial value of c :
14
Adding 6 to it gives 20
subtracting 6 to it gives 14
multiplying 6 to it gives 84
divding by 6 to it gives 14.0
taking modulo gives 2.0
c raised to 6 gives 0.0
```

Comparison Operators :

```
is 15 equal to 10 False
is 15 not equal to 10 True
is 15 greater than or equal to 10 True
is 15 smaller than or equal to 10 False
is 15 greater than 10 True
is 15 smaller than 10 False
```

Logical Operators :

```
if 10>=10 and 19 <20 then : True
if 10>=10 or 19 >20 then : True
```

identity Operators :

```
h is i False
h is not i True
```

Membership Operators :

```
is java in javascript : True
is java not in javascript : False
```

Bitwise Operators :

```
5 and 7 is : 5
5 or 7 is : 7
5 ^ 7 is : 2
5 << 3 is : 40
7 >> 3 is : 0
~ 5 is : -6
~ 7 is : -8
5 >> 2 is : 1
```

```
PS D:\SEM-5\PYTHON\EXPERIMENTS>
```


Python Experiment 2

Aim : To study & implement input-output statement & control and loop statement

Learnings / Understandings : 1. There are diff. ways & syntax statements to take input from user.

2. There are ways to take input & display using functions : `print()`, `append()`, `add()`, `map()`, `split()` from which `append`, `map` & `split` are used for list & `add()` is used for set.

3. 'Append' adds element at the end of list ; 'map' returns map object of results after applying the function to each element in iterable ; 'split' method splits string input into list.

Control & Loop :

1. Following are ways to control flow of execution & loop types to repeat block of code : `while`, `for`, `if...else`, `if...elif...else`.
2. With the help of loops, we can print patterns. By default, `print` ends with `"\n"`, so if we want to continue in same line, we use `print(text, end=" ")`.
3. In python, there are no switch statements.
4. Within loop, 'continue' keyword is used to skip the next lines in the loop while 'break' is used to exit the loop permanently.
5. Additionally, 'pass' keyword is a null operation ; mostly used as a placeholder for future code.

Code (Experiment 2) :

```
a = int(input("Enter the number : "))
print('square root of ', a, a**0.5)

l, b = float(input("Enter the length : ")), float(input("Enter the breadth : "))
print("Area : ", l*b, " ,", "Perimeter : ", 2 * (l + b))

a,b = int(input("Enter 1st numbers : ")), int(input("Enter 2nd numbers : "))
c = a
a = b
b = c

print("Swapping with third variable : ",a,b)

a,b = int(input("Enter 1st numbers : ")), int(input("Enter 2nd numbers : "))
b,a = a,b

print("swapping without third variable : ", a,b)

# another way

a,b = int(input("Enter 1st numbers : ")), int(input("Enter 2nd numbers : "))
print("a :",a," b :",b)

a = a+b
b = a-b
a = a-b

print("After Swapping ")
print("a:",a,"b:",b)

# adding elements to list , set , tuples

n = int(input("Enter the size of list : "))
l = list()

for i in range(n):
    l.append(int(input(f"Enter the {i} element : ")))

print("list :",l)

n = int(input("Enter the size of list : "))
s = set()

for i in range(n):
    s.add(int(input(f"Enter the {i} element : ")))

print("set :", s)

n = int(input("Enter size "))
t = tuple()
l = list(t)

for i in range(n):
    l.append(int(input(f"Enter the {i} element : ")))

t = tuple(l)
```



```
print("tuple :", t)
```

Output :

```
PS D:\SEM-5\PYTHON\EXPERIMENTS> python -u "d:\SEM-5\PYTHON\EXPERIMENTS\EXP2\exp2.1.py"
Enter the number : 5
square root of 5 2.23606797749979
Enter the length : 10
Enter the breadth : 45
Area : 450.0 , Perimeter : 110.0
Enter 1st numbers : 3
Enter 2nd numbers : 5
Swapping with third variable : 5 3
Enter 1st numbers : 65
Enter 2nd numbers : 44
swapping without third variable : 44 65
Enter 1st numbers : 37
Enter 2nd numbers : 34
a : 37 b : 34
After Swapping
a: 34 b: 37
Enter the size of list : 4
Enter the 0 element : 1
Enter the 1 element : 3
Enter the 2 element : 6
Enter the 3 element : 8
list : [1, 3, 6, 8]
Enter the size of list : 4
Enter the 0 element : 99
Enter the 1 element : 77
Enter the 2 element : 44
Enter the 3 element : 22
set : {99, 44, 77, 22}
Enter size 3
Enter the 0 element : 7
Enter the 1 element : 2
Enter the 2 element : 87
tuple : (7, 2, 87)
PS D:\SEM-5\PYTHON\EXPERIMENTS> █
```

Code :

```
n = int(input("Enter the number of rows : "))

for i in range(1,n+1):
    for j in range(1,i+1):
        print(i, end=" ")
    print()

# factorial

print("\nFactorial example : ")

n = int(input("Enter the number for factorial : "))

fact = 1

if n<0 : print("factorial does not exist !!")
elif n==0: print("Factorial of 0 is 1 ")
```

```

else :
    for i in range(1,n+1):
        fact *= i
print(f"Factorial of {n} with for loop : {fact}")
n = int(input("Enter the number for factorial : "))
fact = 1
i=1
if n<0 : print("factorial does not exist !!")
elif n==0: print("Factorial of 0 is 1 ")
else :
    while i <=n:
        fact *=i
        i+=1
print(f"Factorial of {n} with while loop : {fact}")

#Even odd with if else
print("\nEven odd with if else example : ")
n = int (input("Enter a number :"))
if (n%2)==0: print(f"{n} is even")
else: print(f"{n} is odd")

# checking the number
print("\nchecking the number with if elif else example : ")
number = 2+9j
if isinstance(number, int):
    print("Number is an integer.")
elif isinstance(number, list):
    print("Number is a list.")
elif isinstance(number, tuple):
    print("Number is a tuple.")
elif isinstance(number, set):
    print("Number is a set.")
elif isinstance(number, float):
    print("Number is a float data type.")
elif isinstance(number, str):
    print("Number is a string.")
else:
    print("Number is complex.")

# break continue pass

```

```

print("\nbreak continue pass example : ")
for letter in 'python':
    if letter == 't' or letter == '0':
        continue
    print('current letter:',letter)
for letter in 'python':
    if letter == 'y':
        break
    print('current letter:',letter)
for letter in 'python':
    if letter in ['p', 'y', 't', 'h']:
        print('hi')
        pass
print('pass letter:',letter)

```

Output for above code :

```

PS D:\SEM-5\PYTHON\EXPERIMENTS> python -u "d:\SEM-5\PYTHON\EXPERIMENTS\EXP2\exp2.2.py"
Enter the number of rows : 4
1
2 2
3 3 3
4 4 4 4

Factorial example :
Enter the number for factorial : 5
Factorial of 5 with for loop : 120
Enter the number for factorial : 6
Factorial of 6 with while loop : 720

Even odd with if else example :
Enter a number :34
34 is even

checking the number with if elif else example :
Number is complex.

break continue pass example :
current letter: p
current letter: y
current letter: h
current letter: o
current letter: n
current letter: p
hi
hi
hi
hi
pass letter: n
PS D:\SEM-5\PYTHON\EXPERIMENTS>

```

Python Experiment 3

Aim : To implement functions in python

Learning / Understanding : 1. It is a block of code, which only runs when it is called. We can also pass data as params & get return data values as result.

2. The main idea is to implement the principle : DRY (Do Not Repeat Yourself).

3. Syntax : $\langle \text{function-name} \rangle (\text{Param}_1, \dots, \text{Param}_n) :$
 $\leftarrow \text{Statements} \rightarrow$
 $\text{return } \langle \text{something} \rangle$

4. Functions are both inbuilt & user defined

5. Inbuilt functions for list :

- a) `append()` - to add element in list
- b) `extend()` - to extend to new list
- c) `insert()` - to add to a specific index
- d) `pop()` - to remove last element (if position not specified)

6. ~~4~~ Tuple have functions : `len()`, `min()`, `max()`, `count()`, `sort()`, `index()`, `sum()`.

7. Set functions are : `add()`, `discard()`, `remove()`, `pop()`, `clear()`.

8. Dictionary have functions as `copy()` (from keys), `clear()`.

9. Additionally, there are many inbuilt functions in python.

Code (Experiment 3):

List in built function :

```
# append()-appends element to list
print("append() function")
list1 = [10, 20, 30]
print("Original list:", list1)
list1.append(50)
print("List After Appending 50:", list1)

# extend()- extends by adding on with new list
print("extend() function")
list1 = [10, 20, 30]
print("Original list:", list1)
list2 = [222, 333]
list1.extend(list2)
print("After extending to list2, the original list is : ", list1)

# insert() - inserts element at particular location
print("insert() function")
list1 = [10, 20, 30]
print("Original list:", list1)
list1.insert(2, 15)
print("List After Appending 15:", list1)

# pop()- pops element from list
print("pop() function")
list1 = [10, 20, 30]
print("Original list:", list1)
list1.pop(2)
print("List after popping from index 2:", list1)

# copy()-copies a list
print("copy() function")
list1 = [10, 20, 30]
print("Original list:", list1)
list2 = list1.copy()
print("Copied list is:", list2)

# clear()-clears the list
print("Clear() function")
list1 = [10, 20, 30]
print("Original list:", list1)
list2 = list1.clear()
```



```
print("List after clearing:", list2)
```

Output for above code :

```
PS D:\SEM-5\PYTHON\EXPERIMENTS> python -u "d:\SEM-5\PYTHON\EXPERIMENTS\EXP3\list.py"
append() function
Original list: [10, 20, 30]
List After Appending 50: [10, 20, 30, 50]
extend() function
Original list: [10, 20, 30]
After extending to list2, the original list is : [10, 20, 30, 222, 333]
insert() function
Original list: [10, 20, 30]
List After Appending 15: [10, 20, 15, 30]
pop() function
Original list: [10, 20, 30]
List after popping from index 2: [10, 20]
copy() function
Original list: [10, 20, 30]
Copied list is: [10, 20, 30]
Clear() function
Original list: [10, 20, 30]
List after clearing: None
PS D:\SEM-5\PYTHON\EXPERIMENTS> 
```

Tuple in built function :

```
tuple1 = (10, 20, 30, 50, 50, 70, 90, 80, 100)

print("Original tuple:", tuple1)

# len()- length of tuple
print("len() function")

print("The length of tuple is:", len(tuple1))

# count()-repetition of element in tuple
print("count() function")

print("The count of 50 in tuple is:", tuple1.count(50))

# index()- gives index of element in tuple
print("Index() function")

print("The index of 80 in tuple is:", tuple1.index(80))

# sort()-sorts the elements in tuple
print("sort() function")

print("The sorted tuple is:", sorted(tuple1))

# min()- gives minimum element of tuple
print("min() function")

print("The minimum element of tuple is:", min(tuple1))

# max()- gives maximum element of tuple
print("max() function")

print("The maximum element of tuple is:", max(tuple1))

# sum()-gives sum of elements in tuple
```



```
print("sum() function")

print("The sum of elements in tuple is:", sum(tuple1))
```

```
PS D:\SEM-5\PYTHON\EXPERIMENTS> python -u "d:\SEM-5\PYTHON\EXPERIMENTS\EXP3\tuple.py"
Original tuple: (10, 20, 30, 50, 50, 70, 90, 80, 100)
len() function
The length of tuple is: 9
count() function
The count of 50 in tuple is: 2
Index() function
The index of 80 in tuple is: 7
sort() function
The sorted tuple is: [10, 20, 30, 50, 50, 70, 80, 90, 100]
min() function
The minimum element of tuple is: 10
max() function
The maximum element of tuple is: 100
sum() function
The sum of elements in tuple is: 500
PS D:\SEM-5\PYTHON\EXPERIMENTS>
```

Sets in built function :

```
set1 = {'a','b', 'c', 'd', 'e'}

print("Original set is:", set1)

# add()- adds element to set

set1.add('f')

print("add() function")

print("Set after adding 'f'", set1)

# discard() - discards element from set

set1.discard('e')

print("discard() function")

print("Set after discarding/Updating 'e': ",set1)

# remove() - removes element from set

set1.remove('a')

print("remove() function")

print("Set after removing 'a':", set1)

# pop()- pops element from the set

set1.pop()

print("pop() function")

print("Set after popping elements:", set1)

# clear()-clears the set

set1.clear()

print("clear() function")

print("Set after clearing:",set1)
```

```

PS D:\SEM-5\PYTHON\EXPERIMENTS> python -u "d:\SEM-5\PYTHON\EXPERIMENTS\EXP3\set.py"
Original set is: {'d', 'b', 'e', 'c', 'a'}
add() function
Set after adding 'f' {'d', 'b', 'e', 'c', 'a', 'f'}
discard() function
Set after discarding/Updating 'e': {'d', 'b', 'c', 'a', 'f'}
remove() function
Set after removing 'a': {'d', 'b', 'c', 'f'}
pop() function
Set after popping elements: {'b', 'c', 'f'}
clear() function
Set after clearing: set()
PS D:\SEM-5\PYTHON\EXPERIMENTS> 

```

Dictionary in built function :

```

dict1 = {'1': 'One', '2': 'Two', '3': 'Three'}
print("Original dictionary:", dict1)
# copy()- copies the dictionary
dict2 = dict1.copy()
print("Copied Dictionary :", dict2)
# fromkeys() - gives details from dictionary
seq = ('1', '2', '3')
print("fromkeys() method")
print(dict1.fromkeys(seq, None))
# clear()- clears the dictionary
dict1.clear()
print("clear() function")
print("The dictionary after clearing it:", dict1)

```

```

PS D:\SEM-5\PYTHON\EXPERIMENTS> python -u "d:\SEM-5\PYTHON\EXPERIMENTS\EXP3\dictionary.py"
Original dictionary: {'1': 'One', '2': 'Two', '3': 'Three'}
Copied Dictionary : {'1': 'One', '2': 'Two', '3': 'Three'}
fromkeys() method
{'1': None, '2': None, '3': None}
clear() function
The dictionary after clearing it: {}
PS D:\SEM-5\PYTHON\EXPERIMENTS> 

```

User defined function :

```

def histogram(input_list :list) -> list[int]:
    sorted_list = list(set(input_list))
    hist = []

```

```

for i in sorted_list:
    hist.append((i,input_list.count(i)))

histogram = sorted(hist,key=lambda x:x[1])

return histogram

h = histogram([13,7,12,7,11,13,14,13,7,11,13,14,12,14,14,7])

print(h)

def hanoi(disks, source, aux, target):
    if disks==1:
        print(f"Move 1 disk from peg{source} to peg{target}")
        return
    hanoi(disks-1,source,target,aux)
    print(f"Move disk {disks} from peg{source} to peg{target}")
    hanoi(disks-1,aux,source,target)

disk_count = int(input("Enter num of disks : "))
hanoi(disk_count,'A','B','C')

def perfect_number(num):
    """Checks if a number is perfect or not."""
    sum = 0
    for i in range(1,num//2+1):
        if num % i == 0: sum+= i

    return True if sum==num else False

n = int(input("\nEnter a num : "))
isPerfect = perfect_number(n)
print(f"{n} is a perfect number\n") if isPerfect else print(f"{n} is a not a perfect number\n")

# code to print greatest of two number using lamda

a = int(input("Enter 1st number : "))
b = int(input("Enter 2nd number : "))

maximum = lambda a,b : a if a>b else b

```

```
print(f"Maximum btw {a},{b} is {maximum(a,b)}")
```

Output :

```
PS D:\SEM-5\PYTHON\EXPERIMENTS> python -u "d:\SEM-5\PYTHON\EXPERIMENTS\EXP3\userDefinedFunctions.py"
[(11, 2), (12, 2), (7, 4), (13, 4), (14, 4)]
Enter num of disks : 4
Move 1 disk from pegA to pegB
Move disk 2 from pegA to pegC
Move 1 disk from pegB to pegC
Move disk 3 from pegA to pegB
Move 1 disk from pegC to pegA
Move disk 2 from pegC to pegB
Move 1 disk from pegA to pegB
Move disk 4 from pegA to pegC
Move 1 disk from pegB to pegC
Move disk 2 from pegB to pegA
Move 1 disk from pegC to pegA
Move disk 3 from pegB to pegC
Move 1 disk from pegA to pegB
Move disk 2 from pegA to pegC
Move 1 disk from pegB to pegC

Enter a num : 6
6 is a perfect number

Enter 1st number : 45
Enter 2nd number : 55
Maximum btw 45,55 is 55
PS D:\SEM-5\PYTHON\EXPERIMENTS>
```


Python Experiment 4

Aim: To implement classes, objects and Inheritance

Learning/ understanding:

1. Classes serve as blueprints for creating objects, It provides means of bundling data and functionality together.
2. 'class' keyword is used for creation of class, class instances have methods and attributes.
3. Objects are instances of classes, representing tangible entities with specific attributes and methods.
4. Objects are created based on blueprint provided by their respective classes.
5. Each object has its own state, defined by its attributes.
6. Inheritance facilitates the reuse of code by allowing a class to inherit properties and methods from another class.
7. Superclass provides a foundation for common attributes and behaviour, Super subclasses can extend or override inherited functionalities.
8. For every class there is `__init__()` function. It is known as constructor, it is called when an object is created from class and access is required to initialize attributes of class. `super()` keyword is used to call a method from superclass in subclass.
9. It is commonly used in context of inheritance when subclass wants to intake a method defined in superclass.

Code (Experiment 4):

OOPs concept in banking transaction :

```
class BankAccount:

    def __init__(self, account_number, holder_name, balance=0):

        self.account_number = account_number

        self.holder_name = holder_name

        self.balance = balance


    def deposit(self, amount):

        if amount > 0:

            self.balance += amount

            print(f"Deposited ${amount} into account {self.account_number}.")

        else:

            print("Invalid deposit amount.")


    def withdraw(self, amount):

        if 0 < amount <= self.balance:

            self.balance -= amount

            print(f"Withdrew ${amount} from account {self.account_number}.")

        else:

            print("Insufficient funds or invalid withdrawal amount.")


    def display_balance(self):

        print(f"Account {self.account_number} balance: ${self.balance}")


class SavingsAccount(BankAccount):

    def __init__(self, account_number, holder_name, balance=0, interest_rate=0.01):

        super().__init__(account_number, holder_name, balance)

        self.interest_rate = interest_rate


    def apply_interest(self):

        interest = self.balance * self.interest_rate

        self.balance += interest

        print(f"Interest of ${interest} applied to account {self.account_number}.")


class CheckingAccount(BankAccount):

    def __init__(self, account_number, holder_name, balance=0, overdraft_limit=100):

        super().__init__(account_number, holder_name, balance)

        self.overdraft_limit = overdraft_limit
```



```

def withdraw(self, amount):
    if 0 < amount <= (self.balance + self.overdraft_limit):
        self.balance -= amount
        print(f"Withdrew ${amount} from account {self.account_number}.")
    else:
        print("Insufficient funds or invalid withdrawal amount.")

class Bank:
    def __init__(self):
        self.accounts = {}

    def add_account(self, account):
        self.accounts[account.account_number] = account

    def get_account(self, account_number):
        return self.accounts.get(account_number)

# Create instances of BankAccount, SavingsAccount, and CheckingAccount
account1 = BankAccount("001", "Alice", 1000)
savings_account1 = SavingsAccount("002", "Bob", 5000, 0.02)
checking_account1 = CheckingAccount("003", "Charlie", 2000, 500)

# Create a Bank and add accounts to it
bank = Bank()
bank.add_account(account1)
bank.add_account(savings_account1)
bank.add_account(checking_account1)

# Perform transactions
account1.deposit(500)
account1.withdraw(200)
account1.display_balance()

savings_account1.deposit(1000)
savings_account1.apply_interest()
savings_account1.display_balance()

checking_account1.withdraw(2500)
checking_account1.display_balance()

```

Output :

```
PS D:\SEM-5\PYTHON\EXPERIMENTS> python -u "d:\SEM-5\PYTHON\EXPERIMENTS\EXP4\bankingTransaction.py"
Deposited $500 into account 001.
Withdrew $200 from account 001.
Account 001 balance: $1300
Deposited $1000 into account 002.
Interest of $120.0 applied to account 002.
Account 002 balance: $6120.0
Withdrew $2500 from account 003.
Account 003 balance: $-500
PS D:\SEM-5\PYTHON\EXPERIMENTS> []
```

OOPs concept in inventory management :

```
class InventoryItem:

    def __init__(self, item_id, description, price):

        self.item_id = item_id

        self.description = description

        self.price = price

    def __str__(self):

        return f"{self.item_id}: {self.description}, Price: ${self.price:.2f}"

class Electronics(InventoryItem):

    def __init__(self, item_id, description, price, brand, power_consumption):

        super().__init__(item_id, description, price)

        self.brand = brand

        self.power_consumption = power_consumption

    def __str__(self):

        return super().__str__() + f", Brand: {self.brand}, Power Consumption: {self.power_consumption}W"

class Clothing(InventoryItem):

    def __init__(self, item_id, description, price, size, color):

        super().__init__(item_id, description, price)

        self.size = size

        self.color = color

    def __str__(self):

        return super().__str__() + f", Size: {self.size}, Color: {self.color}"

class Furniture(InventoryItem):

    def __init__(self, item_id, description, price, material, dimensions):

        super().__init__(item_id, description, price)

        self.material = material
```

```
        self.dimensions = dimensions

    def __str__(self):
        return super().__str__() + f", Material: {self.material}, Dimensions: {self.dimensions}"

class StoreInventory:

    def __init__(self):
        self.inventory = []

    def add_item(self, item):
        self.inventory.append(item)

    def remove_item(self, item_id):
        for item in self.inventory:
            if item.item_id == item_id:
                self.inventory.remove(item)
                return True
        return False

    def display_inventory(self):
        if not self.inventory:
            print("Inventory is empty.")
        else:
            print("Inventory Items:")
            for item in self.inventory:
                print(item)

# Usage example:
store_inventory = StoreInventory()

# Add items to the inventory
tv = Electronics("E001", "Smart TV", 799.99, "Sony", 120)
shirt = Clothing("C001", "Men's Shirt", 29.99, "M", "Blue")
table = Furniture("F001", "Coffee Table", 149.99, "Wood", "40x20x18 inches")

store_inventory.add_item(tv)
store_inventory.add_item(shirt)
store_inventory.add_item(table)

# Display the inventory
```

```
store_inventory.display_inventory()

# Remove an item
item_id_to_remove = "C001"
if store_inventory.remove_item(item_id_to_remove):
    print(f"Item {item_id_to_remove} removed.")
else:
    print(f"Item {item_id_to_remove} not found.")

# Display the updated inventory
store_inventory.display_inventory()
```

Output :

```
PS D:\SEM-5\PYTHON\EXPERIMENTS> python -u "d:\SEM-5\PYTHON\EXPERIMENTS\EXP4\inventoryManagement.py"
Inventory Items:
E001: Smart TV, Price: $799.99, Brand: Sony, Power Consumption: 120W
C001: Men's Shirt, Price: $29.99, Size: M, Color: Blue
F001: Coffee Table, Price: $149.99, Material: Wood, Dimensions: 40x20x18 inches
Item C001 removed.
Inventory Items:
E001: Smart TV, Price: $799.99, Brand: Sony, Power Consumption: 120W
F001: Coffee Table, Price: $149.99, Material: Wood, Dimensions: 40x20x18 inches
PS D:\SEM-5\PYTHON\EXPERIMENTS> 
```

OOPs concept in flight transaction :

```
class Flight:

    def __init__(self, flight_number, destination, departure_time, capacity):

        self.flight_number = flight_number

        self.destination = destination

        self.departure_time = departure_time

        self.capacity = capacity

        self.passengers = []

    def display_flight_info(self):

        return f"Flight {self.flight_number} to {self.destination} at {self.departure_time}"

    def available_seats(self):

        return self.capacity - len(self.passengers)

    def book_ticket(self, passenger):

        if self.available_seats() > 0:

            self.passengers.append(passenger)

            return True

        else:

            return False

    def reschedule_flight(self, new_departure_time):

        self.departure_time = new_departure_time

    def cancel_reservation(self, passenger):

        if passenger in self.passengers:

            self.passengers.remove(passenger)

            return True

        else:

            return False

    def refund_ticket(self, passenger):

        if passenger in self.passengers:

            self.passengers.remove(passenger)

            return True

        else:

            return False

    def prioritize_reservations(self):

        # Sort passengers by a priority criterion (e.g., frequent flyer status)

        self.passengers.sort(key=lambda passenger: passenger.priority, reverse=True)

class Passenger:

    def __init__(self, name, age, priority=0):

        self.name = name
```

```

        self.age = age

        self.priority = priority

    def __str__(self):
        return f"{self.age}, Age: {self.age}, Priority: {self.priority}"

class Reservation:

    def __init__(self, flight, passenger):
        self.flight = flight
        self.passenger = passenger

    def display_reservation(self):
        return f"Reservation: {self.passenger.name} on {self.flight.display_flight_info()}"

# Usage example:
if __name__ == "__main__":

    # Create a Flight
    flight1 = Flight("F001", "New York", "12:00 PM", 150)

    # Create Passengers
    passenger1 = Passenger("Alice", 30, 5)
    passenger2 = Passenger("Bob", 25, 2)
    passenger3 = Passenger("Charlie", 45, 4)

    # Book Tickets
    flight1.book_ticket(passenger1)
    flight1.book_ticket(passenger2)
    flight1.book_ticket(passenger3)

    # Display Flight Info and Passengers
    print(flight1.display_flight_info())
    for passenger in flight1.passengers:
        print(passenger)

    # Reschedule Flight
    flight1.reschedule_flight("2:00 PM")
    print("Rescheduled to:", flight1.departure_time)

    # Cancel Reservation
    if flight1.cancel_reservation(passenger2):
        print(f"{passenger2.name}'s reservation has been canceled.")

    # Refund Ticket
    if flight1.refund_ticket(passenger3):
        print(f"Ticket refunded for {passenger3.name}.")

    # Display Passengers after changes
    print("Passengers after changes:")
    for passenger in flight1.passengers:

```



```
print(passenger)

# Prioritize Reservations

flight1.prioritize_reservations()

print("Passengers after prioritizing:")

for passenger in flight1.passengers:

    print(passenger)

# Create Reservation

reservation = Reservation(flight1, passenger1)

print(reservation.display_reservation())
```

Output :

```
PS D:\SEM-5\PYTHON\EXPERIMENTS> python -u "d:\SEM-5\PYTHON\EXPERIMENTS\EXP4\flightTransaction.py"
Flight F001 to New York at 12:00 PM
30, Age: 30, Priority: 5
25, Age: 25, Priority: 2
45, Age: 45, Priority: 4
Rescheduled to: 2:00 PM
Bob's reservation has been canceled.
Ticket refunded for Charlie.
Passengers after changes:
30, Age: 30, Priority: 5
Passengers after prioritizing:
30, Age: 30, Priority: 5
Reservation: Alice on Flight F001 to New York at 2:00 PM
PS D:\SEM-5\PYTHON\EXPERIMENTS> █
```

Jigar Siddhura

60004210155

Python Experiment 5

Aim: To understand and implement exception handling

Learning / understanding:

1. Exception handling is used to manage and respond to unexpected or exceptional situations that may arise during the execution of a program.
2. Mostly it is used for error detection, graceful error recovery, debugging, resource management.
3. There are two types of errors:
 - a) Syntax error (due to improper syntax implementation)
 - b) Logical error (when in runtime an error that occurs after passing the syntax test is called exception or logical).
4. There are various types of error which can be handled through exception. Syntax for except, exception handling:

```
< code snippet >
except
    < any exception error > as e:
        .print(e)
```
5. Some of the errors are value error, arithmetic error, import error, lookup error, keyboard interrupt.
6. Besides these you can also create a user defined exception handling for this example is given in the code.

Code (Experiment 5) :

Inbuild error and their handling :

```
import math

import time


def catchKeyError():

    print("\nKey Error implemenation")

    try:

        dic = {'1':'params','2':'query'}

        print(dic['3'])

    except KeyError as ke:

        print('Key error exception : ',ke)


def catchImportError():

    print("\nimport error implemenation")

    try:

        import nonexistent_module

    except ImportError:

        print("Import error: The module does not exist or cannot be imported.")


def catchZeroDivError():

    print("\nzero Division Error implemenation")

    try:

        print(9/0)

    except ZeroDivisionError as e:

        print(e)


def catchRangeError():

    print("\nout range error handling implemenation")

    try:

        l = []

        print(l[1])

    except Exception as e:

        print(e)


def catchKeyboardInterrupt():

    print('\nkeyboard interrupt ')

    try:
```

```

print(input("Enter something : "))

while True:

    print('print')

    time.sleep(1)

except Exception as kbe:

    print(kbe)

```

create a custom exception

```

catchKeyError()

catchImportError()

catchZeroDivError()

catchRangeError()

catchKeyboardInterrupt()

```

Output :

```

Key Error implemenation
Key error exception : '3'

```

```

import error implemenation
Import error: The module does not exist or cannot be imported.

```

```

zero Division Error implemenation
division by zero

```

```

out range error handling implemenation
list index out of range

```

```

keyboard interrupt
Enter something : hey
hey
print
print

```

```

Traceback (most recent call last):

```

```

  File "d:\SEM-5\PYTHON\EXPERIMENTS\EXP5\inbuiltErrors.py", line 88, in <module>
    catchKeyboardInterrupt()

```

```

  File "d:\SEM-5\PYTHON\EXPERIMENTS\EXP5\inbuiltErrors.py", line 75, in catchKeyboardInterrupt
    time.sleep(1)

```

```

KeyboardInterrupt

```

```

PS D:\SEM-5\PYTHON\EXPERIMENTS> █

```

User defined Error Handling :

```
class BaseError(Exception):pass
class HighValueError(Exception):pass
class LowValueError(Exception):pass
value = 81
while(1):
    try:
        n=int(input("Enter number:"))
        if n>value:
            raise HighValueError
        elif n < value:
            raise LowValueError
        else:
            print("Nice!Correct answer")
            break
    except LowValueError:
        print("Very Low Value, Give input again")
        print()
    except HighValueError:
        print("Very High value , give input again")
        print()
```

Output :

```
PS D:\SEM-5\PYTHON\EXPERIMENTS> python -u "d:\SEM-5\PYTHON\EXPERIMENTS\EXP5\userDefErrors.py"
Enter number:99
Very High value , give input again

Enter number:22
Very Low Value, Give input again

Enter number:81
Nice!Correct answer
PS D:\SEM-5\PYTHON\EXPERIMENTS> █
```


Python Experiment 6 :

Aim: To understand and implement file handling.

Learning

1. Python provides file handling, which allows users to handle files i.e. to perform read and write operation along with many other file handling options to operate on files.
2. File handling is easy in python. Here it treats files differently text or binary and this is important. Each line of code includes a sequence of characters and they form a text line.
3. Each line of file is terminated with special character called EOL or new line character ($\backslash n$).
4. It ends current line and tells the interpreter a new one has begun. The various mode in file handling:
 - r (Read): opens file and does not allow writing.
 - w (write): opens file for writing, overrides existing data or creates new file if not present.
 - a (append): opens file for appending. Does not overrides.
 - $r+$ (read and write): opens file for both reading and writing overrides existing data.
 - $w+$ (write and read): Same as above but here if file does not exist a new file is created.
 - $a+$ (append and read): opens file for both appending and reading does not override existing data.
5. To understand this various mode sorting of elements in two different and their results written in new file.

Code (Experiment 6):

File1.txt contains :

```
EXP6 > file1.txt
You, 20 hours ago |
1 3
2 8
3 15
4 20
5 58
6 69
7 78
8 82
9
```

File2.txt contains :

```
EXP6 > file2.txt
You, 20 hours ago |
1 4
2 17
3 28
4 50
5 62
6 82
7 91
```

Sorting code :

```
def getValueAtIndex(content,index):
    return int(content[index].strip())

with open('file1.txt','r') as f, open('file2.txt','r') as g, open('sort.txt','w') as s:
    i = j = 0
    k = 1

    content1, content2 = f.readlines(), g.readlines()

    len1 = len(content1)
    len2 = len(content2)

    while(i<len1 and j<len2):
        value1, value2 = int(content1[i].strip()), int(content2[j].strip())
        if value1 < value2:
            s.write(str(value1)+"\n")
            i+=1
```

```

elif value1 > value2:

    s.write(str(value2)+"\n")

    j+=1

else:

    s.write(str(value1)+"\n")

    s.write(str(value2)+"\n")

    i+=1

    j+=1


while(i<len1):

    value1 = content1[i].strip()

    s.write(str(value1)+"\n")

    i+=1


while(j<len2):

    value2 = content2[j].strip()

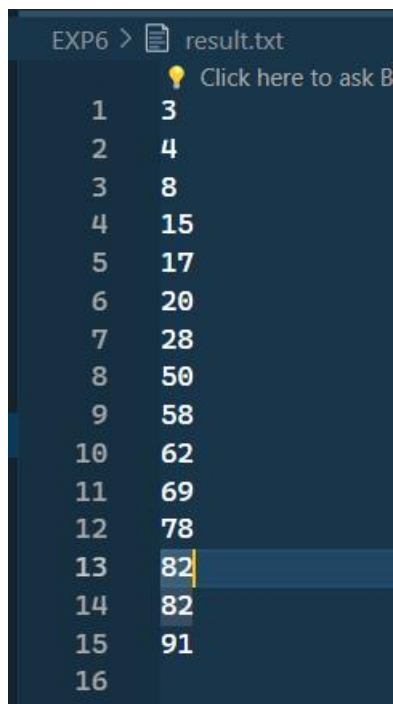
    s.write(str(value2)+"\n")

    j+=1

```

Result :

A new file named as file3.txt is created and contains element as :



```

EXP6 > result.txt
Click here to ask B
1 3
2 4
3 8
4 15
5 17
6 20
7 28
8 50
9 58
10 62
11 69
12 78
13 82
14 82
15 91
16

```

Python Experiment 7

Aim: To understand & implement regular expressions in python.

Learning / Understandings: 1. A regular expression is a special sequence of characters that uses a search pattern to find a string or set of strings.
2. It can detect the presence or absence of a text by matching it with a particular pattern & also can split a pattern into 1 or more sub patterns:

- 're' module - Supports the use of regex in python. Its primary function is to perform a search, where it takes regex & a string.
- `re.findall()` - Return all non-overlapping matches of a string as a list of strings.
- `re.compile()` - Regex are compiled into pattern objects which have methods for various objects - operations such as searching for pattern matches or string substitutions.
- `re.split()` - Split string by occurrences of character or pattern; upon finding the pattern; remaining is returned as list.

Examples:

	Special seq.	Description
1.	<code>\A</code>	Matches if the string begins with given character.
2.	<code>\b</code>	Matches if the word begins or ends with a given char.

\B It is the opposite of \b i.e. should not start / end with given regex

\d Matches any decimal digit in [0-9]

\D Matches any non-digit character

\s Matches any white space character

\S Matches any non-white space char.

\w Match any alpha numeric char.

\W Match any non-alpha numeric char.

~~\Z~~

~~\M~~

• Conclusion: Thus, we implemented regex in python

Code (Experiment 7):

Sample text where re is to be applied :

```
EXP7 > sample.txt
Click here to ask Blackbox to help you code faster
1 fakeemail@gmail.com python@hotmail.com
2 Mrs. pusha
3 https://amazon.com/
4 295058305
5 4*335*95
6 91-1234098756
7 Mr. X
8 X_Y@gmail.com
```

Code for re :

```
import re

file = open('sample.txt')
text = file.read()

email_pat = r'[a-zA-Z0-9\.\_]+\@[a-zA-Z0-9\.\_]+\com'
mob_pat = r'[0-9]+[#\-*]*[0-9]+[0-9]+[#\-*]*[0-9]+'

url_pat = r"(?i)\b(?:https?://|www\d{0,3}[.]|[a-z0-9.-]+[.][a-z]{2,4})/)(?:[^\s()<>+|\\(\[^\s()<>+|\\(\[^\s()<>+|\\)])))+(?:\\(\[^\s()<>+|\\(\[^\s()<>+|\\)]))|([^\s'!()\\[\]{};: '\", <>?«»‘’])"

name_pat = r'M(?:r|s|. |rs|. |s|. ) [a-zA-Z]+'

print(f"Emails in text field are : {re.findall(email_pat, text)}")
print(f"Phone numbers are : {re.findall(mob_pat, text)}")
print(f"Names are : {re.findall(name_pat, text)}")
print(f"Urls are : {re.findall(url_pat, text)}")
```

Output :

```
PS D:\SEM-5\PYTHON\EXPERIMENTS> python -u "d:\SEM-5\PYTHON\EXPERIMENTS\EXP7\regex.py"
Emails in text field are : ['fakeemail@gmail.com', 'python@hotmail.com', 'X_Y@gmail.com']
Phone numbers are : ['295058305', '4*335*9', '91-1234098756']
Names are : ['Mrs. pusha', 'Mr. X']
Urls are : [('https://amazon.com/', '', '', '', '')]
PS D:\SEM-5\PYTHON\EXPERIMENTS> █
```


Python Experiment 8

Aim: To implement database connectivity in python

Theory: Learnings / Understandings:

1. MySQL connector enables python program to connect to MySQL databases, using an api that is compliant with python database — API specification v2.0 (PEP 249).
2. It is written in pure python & does not have any dependencies except for Python standard library.
3. Steps to create connection of python application to database:
 - a) Import mysql connector module.
 - b) Create a connection object
 - c) Create the cursor object
 - d) Execute the query

Creating cursor object: 1. It can be specified defined as an abstraction specified in the python DB-API 2.0.

2. It facilitates us to have multiple separate working environments through same connection to the database.
3. One can create cursor object by calling cursor function of the connection obj
4. Cursor obj is an important aspect of executing queries to the db.

Code (Experiment 8) :

1) code for database creation and table creation:

```
import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    password="xxx",
)

mycursor.execute("CREATE DATABASE electronics")

mycursor = mydb.cursor()

mycursor.execute("""
        CREATE TABLE LAPTOP(
            Id int(11) NOT NULL,
            Name varchar(250) NOT NULL,
            Price float NOT NULL,
            Purchase_date Date NOT NULL,
            PRIMARY KEY(Id)
        )
    """)

mydb.commit()
```

2) code for inserting values in table :

Single values query :

```
import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    password="xxx",
    database="electronics"
)

mycursor.execute("""
        INSERT INTO LAPTOPS(Id, Name, Price, Purchase_date) VALUES(15,'Acer aspire 7','52000','2019-
08-17')
    """)

mydb.commit()
```

Multiple values query :

```

import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    password="xxx",
    database="electronics"
)

mycursor = mydb.cursor()

sql = """
    INSERT INTO LAPTOP(Id, Name, Price, Purchase_date, Rating) VALUES(%s, %s, %s, %s, %s)
    """

val = [(31, 'Microsoft Go', '120000', '2019-08-22', 4.1), (29, 'Macbook M2 pro', '2370000', '2019-08-27', 4.7)]

mycursor.executemany(sql, val)

mydb.commit()

```

3) code for displaying rows in table :

```

import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    password="xxx",
    database="electronics"
)

mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM Laptop")

res = mycursor.fetchall()

for x in res:
    print(x)

```

Output :

```
PS D:\SEM-5\PYTHON\EXPERIMENTS> python -u "d:\SEM-5\PYTHON\EXPERIMENTS\EXP8\displayDB.py"
(15, 'Acer aspire 7', 52000.0, datetime.date(2019, 8, 17))
(26, 'Windows', 144000.0, datetime.date(2020, 11, 23))
(29, 'Macbook M2 pro', 2370000.0, datetime.date(2019, 8, 27))
(31, 'Microsoft Go', 120000.0, datetime.date(2019, 8, 22))
(39, 'Macbook M2 ', 6670000.0, datetime.date(2012, 8, 27))
PS D:\SEM-5\PYTHON\EXPERIMENTS> █
```

4) code for deleting specific row in table :

```
import mysql.connector

mydb = mysql.connector.connect(

    host="localhost",

    user="root",

    password="xxx",

    database="electronics"

)

mycursor = mydb.cursor()

mycursor.execute("DELETE FROM Laptop WHERE Id = 17")

mydb.commit()

res = mycursor.fetchall()

for x in res:

    print(x)
```

Table after deletion :

```
PS D:\SEM-5\PYTHON\EXPERIMENTS> python -u "d:\SEM-5\PYTHON\EXPERIMENTS\EXP8\displayDB.py"
(26, 'Windows', 144000.0, datetime.date(2020, 11, 23))
(29, 'Macbook M2 pro', 2370000.0, datetime.date(2019, 8, 27))
(31, 'Microsoft Go', 120000.0, datetime.date(2019, 8, 22))
(39, 'Macbook M2 ', 6670000.0, datetime.date(2012, 8, 27))
PS D:\SEM-5\PYTHON\EXPERIMENTS> █
```

5) code for upadting specific row in table :

```
import mysql.connector

mydb = mysql.connector.connect(

    host="localhost",

    user="root",

    password="xxx",

    database="electronics"

)
```

```

mycursor = mydb.cursor()

print("Eariler Records : ")

mycursor.execute("SELECT * FROM Laptop")

res = mycursor.fetchall()

for x in res:

    print(x)

mycursor.execute("UPDATE Laptop SET Price = 150000 WHERE Id = 16")

mydb.commit()

print("Updated Records : ")

mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM Laptop")

res = mycursor.fetchall()

for x in res:

    print(x)

```

Output :

```

PS D:\SEM-5\PYTHON\EXPERIMENTS> python -u "d:\SEM-5\PYTHON\EXPERIMENTS\EXP8\updateRow.py"
Eariler Records :
(26, 'Windows', 144000.0, datetime.date(2020, 11, 23))
(29, 'Macbook M2 pro', 2370000.0, datetime.date(2019, 8, 27))
(31, 'Microsoft Go', 120000.0, datetime.date(2019, 8, 22))
(39, 'Macbook M2 ', 6670000.0, datetime.date(2012, 8, 27))
Updated Records :
(26, 'Windows', 144000.0, datetime.date(2020, 11, 23))
(29, 'Macbook M2 pro', 2370000.0, datetime.date(2019, 8, 27))
(31, 'Microsoft Go', 120000.0, datetime.date(2019, 8, 22))
(39, 'Macbook M2 ', 150000.0, datetime.date(2012, 8, 27))
PS D:\SEM-5\PYTHON\EXPERIMENTS> 

```

6) code for searching specific row in table:

```

import mysql.connector

mydb = mysql.connector.connect(

    host="localhost",

    user="root",

    password="xxx",

    database="electronics"

)

mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM Laptop WHERE Rating > 3")

res = mycursor.fetchall()

for x in res:

    print(x)

```

```
PS D:\SEM-5\PYTHON\EXPERIMENTS> python -u "d:\SEM-5\PYTHON\EXPERIMENTS\EXP8\searchRow.py"
(31, 'Microsoft Go', 120000.0, datetime.date(2019, 8, 22))
(39, 'Macbook M2 ', 150000.0, datetime.date(2012, 8, 27))
PS D:\SEM-5\PYTHON\EXPERIMENTS> █
```

7) code for altering table:

```
import mysql.connector

mydb = mysql.connector.connect(

    host="localhost",

    user="root",

    password="xxx",

    database="electronics"

)

mycursor = mydb.cursor()

mycursor.execute("ALTER TABLE Laptop ADD payment_mode varchar(250)")

mydb.commit()

mycursor.execute("SELECT * FROM Laptop")

res = mycursor.fetchall()

for x in res:

    print(x)
```

Output where a new column is added :

```
PS D:\SEM-5\PYTHON\EXPERIMENTS> python -u "d:\SEM-5\PYTHON\EXPERIMENTS\EXP8\alterTable.py"
(26, 'Windows', 144000.0, datetime.date(2020, 11, 23), None)
(29, 'Macbook M2 pro', 2370000.0, datetime.date(2019, 8, 27), None)
(31, 'Microsoft Go', 120000.0, datetime.date(2019, 8, 22), None)
(39, 'Macbook M2 ', 150000.0, datetime.date(2012, 8, 27), None)
PS D:\SEM-5\PYTHON\EXPERIMENTS> █
```

8) code for deleting table and database :

```
import mysql.connector

mydb = mysql.connector.connect(

    host="localhost",

    user="root",

    password="xxx",

    database="electronics"

)

mycursor = mydb.cursor()

mycursor.execute("DROP TABLE LAPTOPS")
```

```
print('\nTable Dropped\n')  
mydb.commit()  
mycursor.execute("DROP DATABASE electronics")  
mydb.close()
```

Output :

```
PS D:\SEM-5\PYTHON\EXPERIMENTS> python -u "d:\SEM-5\PYTHON\EXPERIMENTS\EXP8\delTable.py"  
Table Dropped  
PS D:\SEM-5\PYTHON\EXPERIMENTS> █
```


Python Experiment 9

Aim: Design a GUI application to show I/O operations using Tkinter. (calculator)

Learning / Understanding: 1. It is the most commonly used application for developing GUI (Graphical User Interface)

2. It is a standard python interface to the Tk GUI toolkit shipped with python.

3. As Tk & Tkinter are available on most of the unix platforms as well as windows, development becomes fast & easy.

4. Steps to create Tkinter module:

- Import Tkinter module
- Create GUI application main window.
- Add one or more widgets to the GUI application.
- Enter main event loop to make loop against each event triggered by the user.

→ Tkinter widgets:

- Button - To display in app.
- Label - Provide single line caption for other widgets
- Text - Display text in multiple lines
- Frame - Used as a container to organize other widget

→ Geometry management:

- pack() - Organize widget in blocks before placing in parent widget
- grid() - organize widget in table like structure in parent widget
- place() - organize by placing in specific parent.

Code (Experiment 9) :

```
from tkinter import *

class MyWindow:

    def __init__(self, win):

        self.lbl1 = Label(win, text='Select Shape:')

        self.lbl2 = Label(win, text='Select Calculation:')

        self.t1 = StringVar()

        self.t2 = StringVar()

        shapes = ['Circle', 'Rectangle', 'Sphere', 'Cone']

        calculations = ['Area', 'Perimeter', 'Curved Area', 'Volume', 'Total Surface Area']

        self.shape_dropdown = OptionMenu(win, self.t1, *shapes)

        self.calculation_dropdown = OptionMenu(win, self.t2, *calculations)

        self.shape_dropdown.place(x=200, y=10)

        self.calculation_dropdown.place(x=200, y=50)

        self.lbl1.place(x=100, y=10)

        self.lbl2.place(x=100, y=50)

        self.dimension_label_1 = Label(win, text='Enter 1st Dimension:')

        self.dimension_label_1.place(x=100, y=90)

        self.dimension_entry_1 = Entry(win)

        self.dimension_entry_1.place(x=200, y=90)

        self.dimension_label_2 = Label(win, text='Enter 2nd Dimension:')

        self.dimension_label_2.place(x=100, y=110)

        self.dimension_entry_2 = Entry(win)

        self.dimension_entry_2.place(x=200, y=110)

        self.dimension_label_3 = Label(win, text='Enter 3rd Dimension:')

        self.dimension_label_3.place(x=100, y=130)

        self.dimension_entry_3 = Entry(win)

        self.dimension_entry_3.place(x=200, y=130)

        self.result_label = Label(win, text='')

        self.result_label.place(x=200, y=170)

        self.btn = Button(text='Submit', command=self.calculate)

        self.btn.place(x=200, y=210)

    def calculate(self):

        selected_shape = self.t1.get()

        selected_calculation = self.t2.get()

        d1 = self.dimension_entry_1.get()

        d2 = self.dimension_entry_2.get()

        d3 = self.dimension_entry_3.get()
```

```
result = "Result: "

if selected_shape == 'Circle':

    if selected_calculation == 'Area':

        res = str(3.14 * float(d1)*float(d1))

        result += res

    elif selected_calculation == 'Perimeter':

        result += "no Perimeter Calculation"

    elif selected_calculation == 'Curved Area':

        result += "Circle Curved Area Calculation"

elif selected_shape == 'Rectangle':

    if selected_calculation == 'Area':

        res = float(d1) * float(d2)

        result += str(res)

    elif selected_calculation == 'Perimeter':

        res = 2 * (float(d1) + float(d2))

        result += str(res)

    elif selected_calculation == 'Curved Area':

        result += "no Curved Area Calculation"

elif selected_shape == 'Sphere':

    if selected_calculation == 'Volume':

        res = 1.33 * 3.14 * (float(d1) * float(d1) * float(d1))

        result += str(res)

    elif selected_calculation == 'Perimeter':

        result += "no Perimeter Calculation"

    elif selected_calculation == 'Area':

        result += "no Area Calculation"

    elif selected_calculation == 'Total Surface Area':

        res = 4 * 3.14 * (float(d1) * float(d1))

        result += str(res)

elif selected_shape == 'Cone':

    if selected_calculation == 'Volume':

        res = 0.3333 * 3.14 * (float(d1) * float(d1)) * float(d2)

        result += str(res)

    elif selected_calculation == 'Perimeter':

        result += "no Perimeter Calculation"

    elif selected_calculation == 'Area':

        result += "no Area Calculation"

    elif selected_calculation == 'Total Surface Area':

        res = 3.14 * float(d1) * (float(d1) + float(d2))
```

```
        result += str(res)

        self.result_label.config(text=result)

window = Tk()

mywin = MyWindow(window)

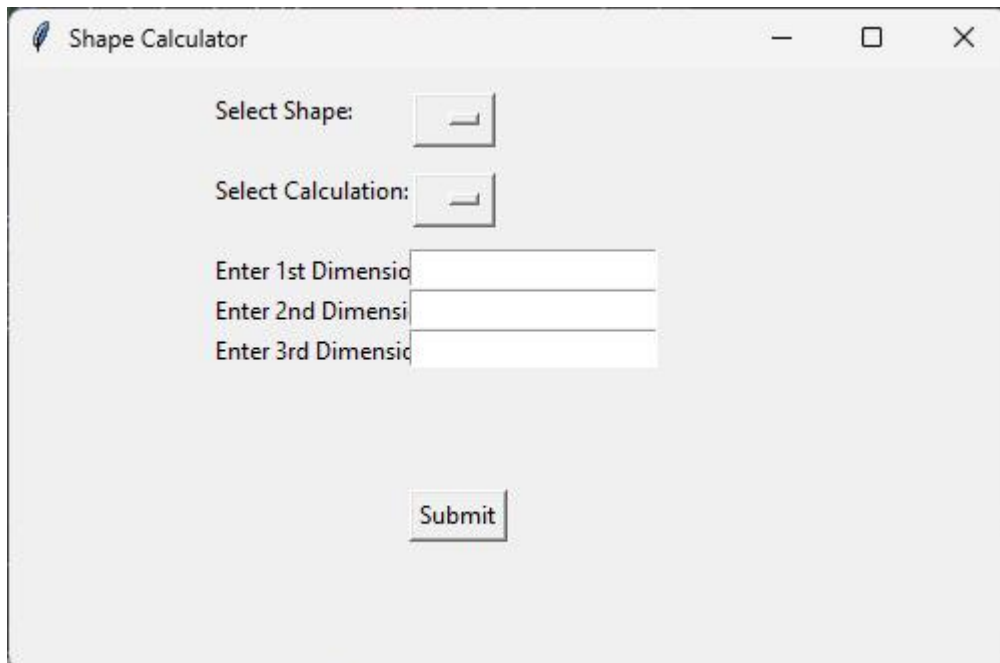
window.title('Shape Calculator')

window.geometry("500x300+10+10")

window.mainloop()
```

Output Screen :

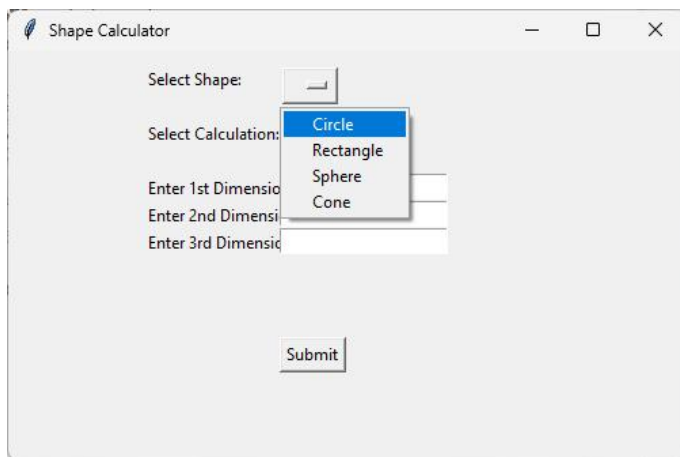
Initial screen :



The screenshot shows a window titled "Shape Calculator" with a standard macOS-style title bar (red, yellow, and green buttons). The window contains the following elements:

- Select Shape:** A dropdown menu with a small arrow icon on the right.
- Select Calculation:** A dropdown menu with a small arrow icon on the right.
- Enter 1st Dimension:** A text input field.
- Enter 2nd Dimension:** A text input field.
- Enter 3rd Dimension:** A text input field.
- Submit:** A button located at the bottom center of the window.

selecting Shape , calculation type and entering values :



Shape Calculator

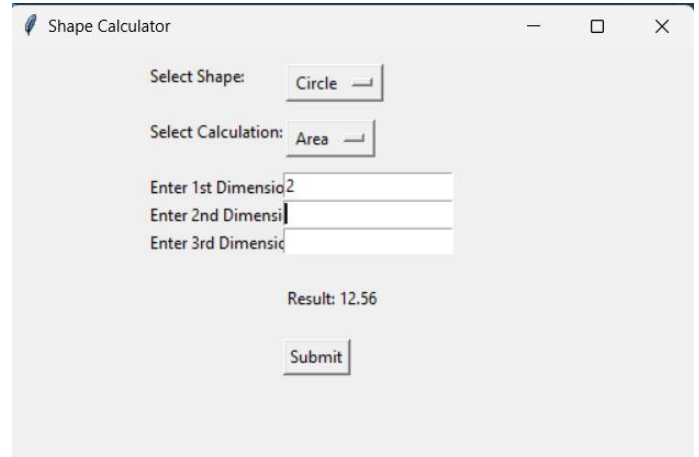
Select Shape:

Select Calculation:

Enter 1st Dimension:

Enter 2nd Dimension:

Enter 3rd Dimension:



Shape Calculator

Select Shape:

Select Calculation:

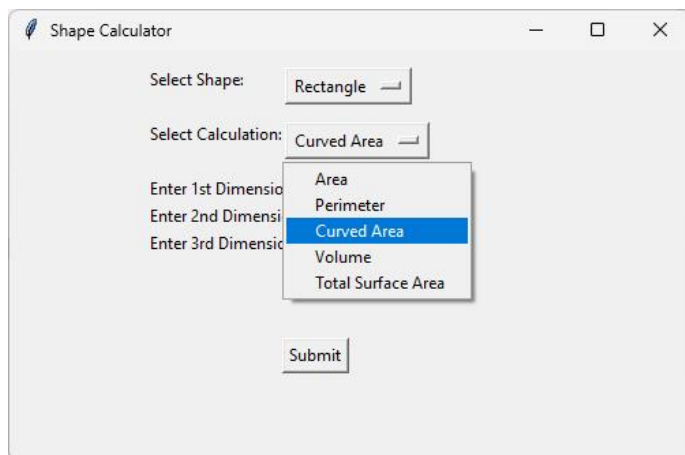
Enter 1st Dimension:

Enter 2nd Dimension:

Enter 3rd Dimension:

Result: 12.56

Selecting wrong configuration :



Shape Calculator

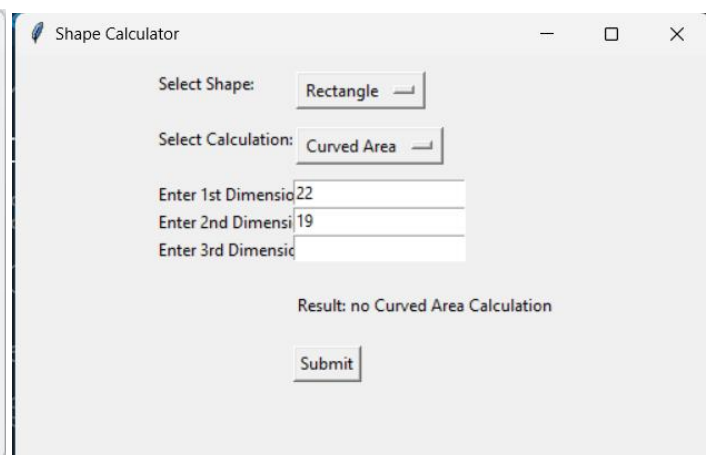
Select Shape:

Select Calculation:

Enter 1st Dimension:

Enter 2nd Dimension:

Enter 3rd Dimension:



Shape Calculator

Select Shape:

Select Calculation:

Enter 1st Dimension:

Enter 2nd Dimension:

Enter 3rd Dimension:

Result: no Curved Area Calculation

Python - Experiment 10

Aim: To implement various operations on dataset

Learning/Understanding:

1. For performing various operations on dataset (for analysis purpose), there are many libraries like pandas, numpy, matplotlib, seaborn & many more.
2. They are used for analyzing, cleaning, exploring & manipulating data.
3. Pandas allows us to make big data & make conclusion based on statistical theories.
4. It can clean messy data sets & make them readable & relevant; it can find correlation b/w 2 or more column, also it can delete column, rows that are not relevant.
5. Matplotlib is a 2D library that gives geometric figures, graphs in variety of format & interactive environments. It is used to generate plots, histogram, power spectrum, bar charts, scatterplots, etc.
6. Numpy is a power numerical library which provides support for large multi-dimensional arrays & matrices along mathematical function to operate on these arrays. It is mostly foundation for many other scientific computing libraries.

Conclusion:

Hence, we understand various operations that can be performed on a dataset with python libraries like numpy, pandas, seaborn, etc. We learnt various patterns in dataset & also how to use libraries for this.

Code (Experiment 10) :

```
import pandas as pd  
list1=[]  
var1=pd.Series(list1)  
print(var1)
```

```
0    21  
1    18  
2     5  
3     3  
4     4  
dtype: int64
```

Giving custom indexes :

```
var1=pd.Series(list1,index=['a','b','x','y','z'])  
print(var1)
```

```
a    21  
b    18  
x     5  
y     3  
z     4  
dtype: int64
```

series for dictionary :

```
dist1={'1':'abc','2':'dfg'}  
var1=pd.Series(dist1)  
print(var1)
```

```
1    abc  
2    dfg  
dtype: object
```

creating a DataFrame and printing value of index 0:

```
data={  
    'sapid':[157,159,161,None],  
    'name':['abc','dfg','xyz',None]}
```

```
}  
df=pd.DataFrame(data)  
df.loc[0]
```

```
sapid    157.0  
name      abc  
Name: 0, dtype: object
```

Finding the Index of Rows in a DataFrame Where the 'name' Column is 'abc' :

	sapid	name
0	12.0	abc
1	14.0	dfg
2	15.0	xyz
3	NaN	erf

```
row=df.loc[df['name']=='xyz'].index  
row
```

```
Int64Index([2], dtype='int64')
```

Use of head ,tail, info, finding null values in Dataframe :

```
df.head(2)
```

	sapid	name
0	157.0	abc
1	159.0	dfg

```
df.tail(1)
```

	sapid	name
3	NaN	None

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 2 columns):
#   Column  Non-Null Count  Dtype
---  ------  -
0    sapid    3 non-null        float64
1    name     3 non-null        object
dtypes: float64(1), object(1)
memory usage: 192.0+ bytes
```

filling Null Values with mean :

```
data={
    'sapid':[156,159,160,None],
    'name':['abcd','efgh','xyzq','erff']
}
df=pd.DataFrame(data)

df.fillna(df.mean())
```

	sapid	name
0	156.000000	abcd
1	159.000000	efgh
2	160.000000	xyzq
3	158.333333	erff

filling Null Values with median:

```
data={
    'sapid':[156,159,160,None],
    'name':['abcd','efgh','xyzq','erff']
}
df=pd.DataFrame(data)
```

```
df.fillna(df.median())
```

	sapid	name
0	156.0	abcd
1	159.0	efgh
2	160.0	xyzq
3	159.0	erff

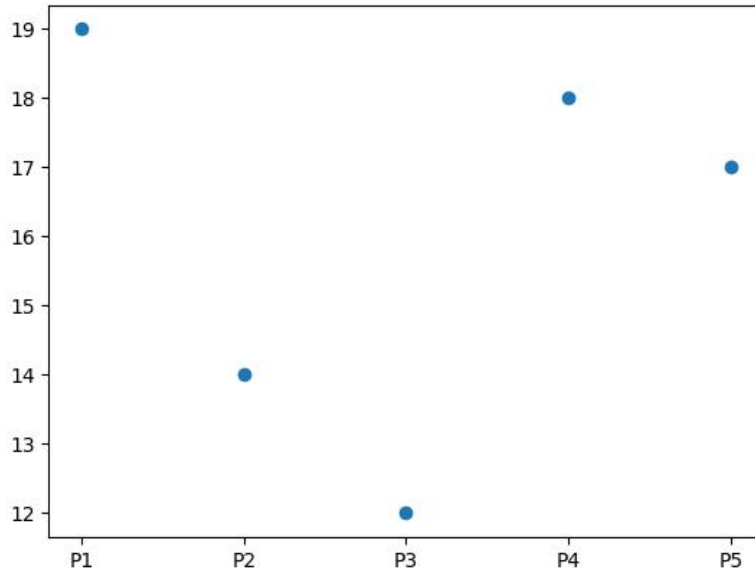
Use of Describe method :

```
df.describe()
```

	sapid
count	3.000000
mean	158.333333
std	2.081666
min	156.000000
25%	157.500000
50%	159.000000
75%	159.500000
max	160.000000

Plotting a graph :

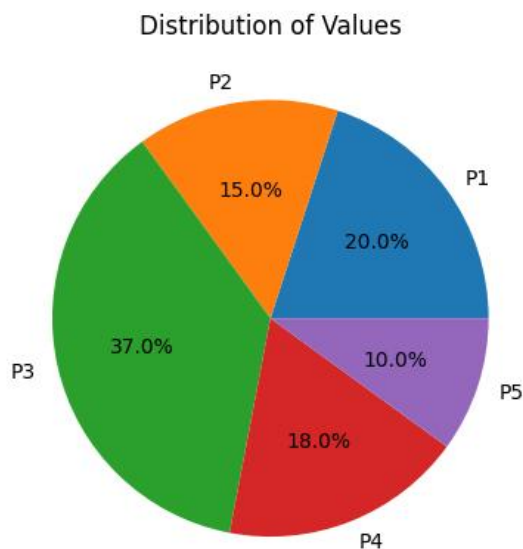
```
import matplotlib.pyplot as plt
import numpy as np
x=np.array(['P1', 'P2', 'P3', 'P4', 'P5'])
y=np.array([19, 14, 12, 18, 17])
plt.scatter(x,y)
plt.show()
```



Plotting a Piechart :

```
x = np.array(['P1', 'P2', 'P3', 'P4', 'P5'])
y = np.array([20, 15, 37, 18, 10])

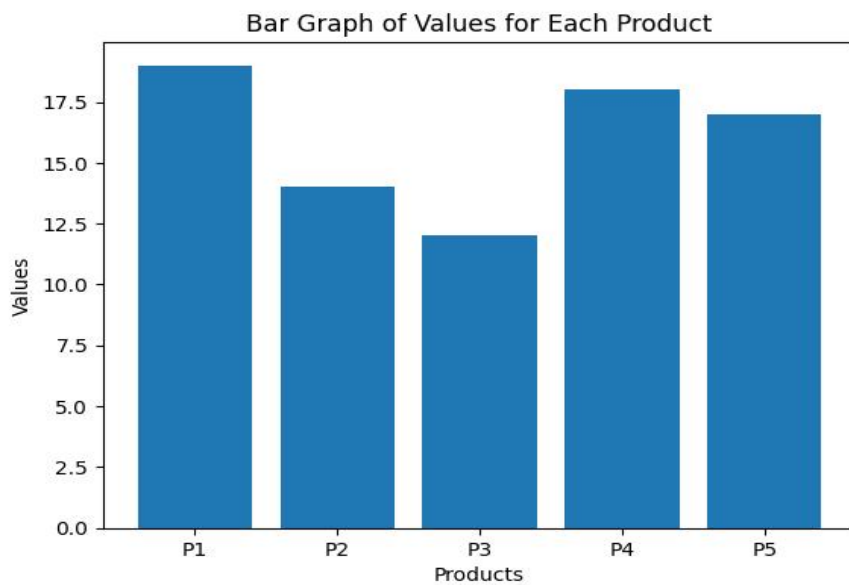
plt.pie(y, labels=x, autopct='%1.1f%%')
plt.title('Distribution of Values')
plt.show()
```



Plotting a bargraph:

```
x = np.array(['P1', 'P2', 'P3', 'P4', 'P5'])
y = np.array([19, 14, 12, 18, 17])

plt.bar(x, y)
plt.xlabel('Products')
plt.ylabel('Values')
plt.title('Bar Graph of Values for Each Product')
plt.show()
```



Plotting a histogram:

```
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(42)
data = np.random.randn(1000)

plt.hist(data, bins=20, color='skyblue', edgecolor='black')

plt.xlabel('Random Values')
plt.ylabel('Frequency')
plt.title('Histogram of Random Values')

plt.show()
```

Histogram of Random Values

