A BOOK FOR BEGINNERS

PYTHON 3 WORKBOOK

Quick Revision of Python
With Exercises And Solutions.

BY RITIK SHUKLA

Preface:

A Book for those who have already completed the python course and now want to practice or revise their python skills by solving questions on topics such as Data Types, Loops, Decision Making, Functions, and Different Data Structures in python.

It contains 80 exercises only for python revision and practice. Each exercise that you complete will strengthen your understanding and will enhance your ability to tackle different types of problems.

Some solutions may have a different approach to solve and the major ones will be provided. I hope you will try to solve them on your own before looking into the solutions and also will compare your solutions with the actual ones.

If you get stuck on any exercise try to revise the concept or try looking at the syntaxes again and then try to solve it, a quick peek at the solutions will also be fine.

I also will recommend you to try one-liner solutions as well and try to make your code as short as possible and have fun with all the tweaks that you can make into it.

The Book starts with covering all basics of python concepts along with the exercises which may be random so revisit the concepts if needed.

80 Example Exercises For Quick Python Revision.

Basics of Python

Data Types, Operations, Casting and Input – Output.

1. Write a program to print “Hello World”.

```
print("Hello World")
```

Hello World

print() is used to display the information to the screen just like printf() does in c.

2. Write a program to input a string value.

```
input("Enter any string")
```

Enter any string: ‘Hello’

‘Hello’

input() is used to take input from the user and by default it takes te value as a string.

Note: The text is always treated as a string inside ‘ ‘ single quotation or “ “ double quotation.

3. Write a program to input an integer value.

```
int(input("Enter any string"))
```

Enter any string: ‘500’

500

int() is used to convert the information given in the parameters into an integer. If instead of number other values would be given then it will show ValueError.

4. Write a program to input a float value.

```
float(input("Enter any string"))
```

Enter any string: ‘5’

5.0

Similar to int(), float() is used to convert the information given in the parameters into a float value. If instead of number other values would be given then it will show ValueError.

5. Write a program to convert given value x = 5.05 to string.

```
X = 5.5
print(str(x))
```

‘5.5’

6. Write a program to store integer 7 in variable x and ‘Hello’ in variable y and print the types of x and y.

```
x=7
y='Hello'
print(type(x))
print(type(y))
```

<class 'int'>

<class 'str'>

type() is used to print the type of value, since x is integer 7 and y is string ‘Hello’ both of their respective class type has been printed.

7. Write a program to input two value in x,y and perform addition, subtraction, multiplication, float and floor division, modulo division and exponent of x and y.

```
X = int(input('Enter Number:'))
y = int(input('Enter Number:'))
print(x+y)
print(x-y)
print(x*y)
print(x/y)
print(x//y)
print(x%y)
print(x**y)
```

Enter Number: 2

Enter Number: 5

7

-3

10

0.4

0

2

32

8. Write a program to input two value in x,y and perform all comparison operations between them.

```
x=int(input('Enter Number:'))
y=int(input('Enter Number:'))
print(x==y)
print(x!=y)
print(x>y)
print(x<y)
print(x>=y)
print(x<=y)
```

Enter Number: 5
Enter Number: 10
False #5==10
True #5!=10
False #5>10
True #5<10
False #5>=10
True #5<=10

9. Write a program to show case all logical operations.

```
x=3
print(x<5 and x>3)
print(x<5 or x>5)
print(not(x<5 and x<10))
```

False
True
False

10. Write a program to check if x refers to same object as y does (Identity operations).

```
x,y = 5,5
print(x is y)
print(x is not y)
```

```
True        #x is y - True if both are same object
False       #x is not y - True if both are not same object
```

11. Write a program to check if character ‘s’ is in string ‘string’ (Membership operator).

```
print('s' in 'string')
print('s' not in 'string')
```

True
False

12. Use bool() function to evaluate any string, integer, boolean values, None, 0, and empty string “ “.

```
bool("String")
bool(1)
bool(False)
bool(None)
bool(0)
bool("")
```

```
True
True
False
False
False
False
```

bool() evaluates any value to bool values True and False. 1, True and any other filled values will be evaluated to true and False for empty values such as 0, None, False, Empty lists, etc.

LIST

Lists are used to store multiple items of different data types in a single variable. It’s one of 4 built-in data types in Python used to store collections of data.

Syntax: ListName = [ ]

13. Write a program to input values in a List and print it.

```
List1 = input().split()
print(List1)
```

['7', '8', '9', '10']

.split() method is used to split the values separated by space or by given values in its parameter just like an example given below.

14. Write a program to convert this string “Hello, I, Love, Python “ into a list separating comma ‘ , ’ values.

```
List1 = "Hello, I, Love, Python".split(',')
print(List1)
```

['Hello', ' I', ' Love', ' Python']

15. Write a program to create a List and store/append all data types values in it.

```
L = []
L.append('String')
L.append(True)
L.append(False)
L.append(10)
L.append(2.5)
L.append(None)
print(L)
```

['String', True, False, 10, 2.5, None]

append() is used to add the elements to the list.

16. Write a program to output length of a List L = [1,2,3,4,'Hello',True].

```
L = [1,2,3,4,'Hello',True]
print(len(L))
```

6

len() is used to return the length of a given iterable object like lists, strings, tuples, etc.

17. Write a program print the first element of a list.

```
L = [1,2,3,4,'Hello',True]
print(L[0])
```

1

[ ] is used for accessing the elements from an iterable objects.

Note: List slice syntax List[Initial : End : Step]

18. Write a program to print all elements of a list.

```
L = [1,2,3,4,'Hello',True]
print(L[::])
```

[1, 2, 3, 4, 'Hello', True]

[::]

19. WAP to print elements from 1st to 3rd index.

```
L = [1,2,3,4,'Hello',True]
print(L[1:4])
```

[2, 3, 4]

20. Write a program to print all elements in a reverse order.

```
L = [1,2,3,4,'Hello',True]
print(L[::-1])
```

[True, 'Hello', 4, 3, 2, 1]

21. Write a program to print all elements by step 2 .

```
L = [1,2,3,4,'Hello',True]
print(L[::2])
```

[1, 3, 'Hello']

22. Write a program to change the 1st element to ‘7’.

```
L = [1,2,3,4,'Hello',True]
L[0] = 7
print(L)
```

[7, 2, 3, 4, 'Hello', True]

23. Write a program to sort the list `L = [1,22,13,14,55]` in ascending order.

```
L=[1,22,13,14,55]
L.sort()
print(L)
```

[1, 13, 14, 22, 55]

sort() is used to sort the list by default in ascending order and if revere is specified to True then descending.

24. Write a program to sort the list `L=[1,22,13,14,55]` in descending order.

```
L=[1,22,13,14,55]
L.sort(reverse=True)
print(L)
```

[55, 22, 14, 13, 1]

25. Write a program to count integer 7 in a list `L=[1,7,7,2,3,7,5]`.

```
L=[1,7,7,2,3,7,5]
print(L.count(7))
```

7

count() is used to count the number of occurrences of a specified element in an iterable object.

26. Write a program to find the index of integer 7 in a list L=[1,7,7,2,3,7,5].

```
L=[1,7,7,2,3,7,5]
print(L.index(7))
```

1

index() is used to return the first index occurrence of an item in a list.

27. Write a program to remove the last element of a list.

```
L=[1,7,7,2,3,7,5]
L.pop()
print(L)
```

[1, 7, 7, 2, 3, 7]

pop() is used to remove the last element from a list.

28. Write a program to extend the given list L = [7,8,9] by adding multiple values 1,2,3, and 4.

```
L=[7,8,9]
L.extend([1,2,3,4])
print(L)
```

[7, 8, 9, 1, 2, 3, 4]

extend() is used to add multiple values to te list.

29. Write a program to convert a list `L=['Hello','Python']` into a string.

```
L=['Hello','Python']
print("".join(L))
```

Hello Python

join() is used to join all list string items adjoining specified string value.

30. Write a program to insert integer 9 at index 3.

```
x='Ritik'
y=20
print(f"{x}{y}")
print("{}{}".format(x,y))
```

Ritik20

31. Write a program to replace element 9 by 99.

```
x='Ritik'
y=20
print(f"{x}{y}")
print("{}{}".format(x,y))
```

Ritik20

Tuples

Second built-in data type in Python used to store items and is ordered, immutable and allows indexing and duplicate values.

It’s similar to lists except we cannot change values of tuples, some operations and methods are also same like slicing , accessing elements, count() and index().

32. Write a program to create a tuple `T=('A',1,'B',2,'C',3,True,['List'])` , access the first element and print elements from index 1 to 3.

```
T=('A',1,'B',2,'C',3,True,['List'])
print(T[0])
print(T[1:4])
```

A
(1, 'B', 2)

33. Write a program to convert a list into a tuple.

```
L=('A',1,'B',2,'C',3,True,['List'])
print(tuple(L))
```

('A', 1, 'B', 2, 'C', 3, True, ['List'])

34.1. Write a program to unpack and store this tuple values into three different variables.

```
T=(1,2,3)
A,B,C=T
print(A,B,C)
```

1 2 3

34.2. Write a program to pack the same three variables into a tuple.

```
A,B,C=1,2,3
T=(A,B,C)
print(T)
```

(1, 2, 3)

Sets

Third built-in data type in Python which is used to store multiple items, and is unordered, unchangeable and unindexed and it do not allows duplicate values. So sets cannot be sorted, Indexed, or sliced.

Syntax: S = set()

35. Write a program to create an empty set and add values to it including duplicates.

```
S=set()
S.add('A')
S.add(1)
S.add('A')
S.add(1)
print(S)
```

{1, 'A'}

add() is used to add items into set as similar to append was for list.

36. Write a program to add multiple items `[2,'B']`

in an existing above referred set.

```
S={1,'A'}
S.update([2,'B'])
print(S)
```

{'A', 1, 2, 'B'}

update() is used for adding multiple values from iterable object into set.

37. Write a program to remove ‘B’ from above set.

```
S={'A',1,2,'B'}
S.discard('B')
print(S)
```

{1, 'A', 2}

OR

```
S={'A',1,2,'B'}
S.remove('B')
print(S)
```

{1, 'A', 2}

Only difference between remove and discard is that remove will raise an error if the element which we’re trying to remove is not present and discard will not raise any error even if the element is not present.

38. Write a program to clear the set.

```
S={'A',1,2,'B'}
S.clear()
print(S)
```

set()

clear() is used for clearing the set by deleting whole elements of a set, and del() is used for deleting the set variable from the memory.

Some of other built-in methods of sets are:

difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets

Dictionary

Dictionaries are used to store data values in key: value pairs. A dictionary is a collection which is ordered/unordered, changeable and does not allow duplicates key.

Form Python 3.7 dictionaries are ordered but in all previous versions dictionaries were unordered.

Syntax: D = { }

Example: `D={1:'A',2:'B',3:'C'}`

39. Write a program to print the type and length of this dictionary `D={1:'A',2:'B',3:'C'}`.

```
D={1:'A',2:'B',3:'C'}
print(type(D))
print(len(D))
```

<class 'dict'>
3

As there is 3 key:value pairs items so is the length of the dictionary.

40. Write a program to print the value of key ‘3’ in this dictionary `D={1:'A',2:'B',3:'C'}`.

```
D={1:'A',2:'B',3:'C'}
print(D[3])
```

‘C’

We access and change elements using [ ], just like we did in other data types such as List and Tuples.

41. Write a program to insert key 4 with value ‘D’ and also change the value of key 33 to ‘CC’.

```
D={1:'A',2:'B',3:'C'}
D[4]='D'    OR   D.update({4:'D'})
D[3]='CC'
print(D)
```

{1: 'A', 2: 'B', 3: 'CC', 4: 'D'}

42. Write a program to remove the pair 3:’CC’ and then remove the last pair of a dictionary D={1:'A',2:'B',3:'CC',4:'D'}.

```
D={1:'A',2:'B',3:'CC',4:'D'}
D.pop(3)
print(D)
D.popitem()
print(D)
```

{1: 'A', 2: 'B', 4: 'D'}
{1: 'A', 2: 'B'}

pop() removes the specified key and popitem() is used to remove the last item from the dictionary.

43. Write a program to print all keys, values, and key:value pairs/items of this dictionary `D={1:'A',2:'B',3:'CC',4:'D'}` in a separate line.

```
D={1:'A',2:'B',3:'CC',4:'D'}
print(D.keys())
print(D.values())
print(D.items())
```

dict_keys([1, 2, 3, 4])
dict_values(['A', 'B', 'CC', 'D'])
dict_items([(1, 'A'), (2, 'B'), (3, 'CC'), (4, 'D')])

44. Write a program to print the value of 5 if not present in a dictionary then print 0.

```
D={1:'A',2:'B',3:'CC',4:'D'}
print(D.get(5,0))
```

0

get() is used to return the value of the specified key and also allows to specify the default value which will be returned if the specified key is not present in the dictionary.

IF – ELSE, Nested IF – Else:

We use logical conditions which includes use of boolean, comparison and logical operators to check for certain conditions and then to follow certain approach depending on the given conditions.

Boolean Operations used: `True & False`,`1 & 0`, Empty Values (`[], {}, (), None`) & Non Empty Values (`[1,2],1,'String',etc`).

Logical Conditions used : `and, or, not`

```
Equals(a==b)
Not Equals (a!=b)
Less than (a<b)
Greater than (a>b)
Less than Equal to (a<=b)
Greater than Equal to (a>=b).
```

Syntax for IF - Else:

```
if (condition):
     #Statements to execute
else:
     #Other Statements
```

Syntax for Nested IF - Else:

```
if (condition):
     #Statements to execute
elif (condition):
     #Other Statements
else:
    #Other Statements
```

45. Write a program to check and print the greater number among a = 10, b = 20 and c =15.

```
a,b,c=10,20,15
if a>b and a>c:
     print(a)
elif b>a and b>c:
     print(b)
else:
     print(c)
```

20

46. Write a program to check if A's value = 22 is greater then 20 and if it is the check if B's value = 21 is less or greater than 20 if greater print 'Welcome' if not print 'Not Welcome'.

```
A,B=22,21

if A>20:
  if B > 20:
    print("Welcome")
  else:
    print("Not Welcome")
```

Welcome

OR

```
if A > 20 and B>20:
  print("Welcome")
else:
  print("Not Welcome")
```

OR (One Liner)

```
print("Welcome") if A>20 and B>20 else print("Not
Welcome")
```

While Loop

The statements under while loop is executed as long as the specified condition is True.

Syntax:

```
while (condition):
  #Execute Statement
```

With `break` statement we can stop the loop once the break statement becomes true.

With `continue` statement we can stop the current iteration and continue with the next loop.

47. Write a program to print from 0 to 4.

```
i=0
while i<=4:
     print(i)
     i+=1 #i+=1 is same as i=i+1 which is
     incrementing i by 1 each loop.
```

```
0
1
2
3
4
```

48. Write a program to print from 0 to 4 but skip 3.

```
i=0
while i<=4:
  print(i)
  i+=1
  if i == 3:
    continue
```

```
0
1
2
3
4
```

49. Write a program to print all numbers but stop if the number is divisible by 6.

```
i=0
while True:
 print(i)
 i+=1
 if i%6==0:
   break
```

```
0
1
2
3
4
5
```

For Loop

Similar to while loop but we do not need to increment the manual increment condition here.

For loop is used for iterating over a sequence / iterables like string, list, tuples, set, dictionaries.

For Loop Syntax:

```
for _ in sequence:
  #Statements
```

Nested For Loop:

```
for _ in sequence:
     for _ in sequence:
          #Statements
```

Usage of break and continue statements is same as it's in while loop.

Else statement at the end and outside of the for loop is executed when neither of the conditions in for loop satisfies.

```
For _ in sequence:
  #Statements
else:
```

50. Write a program to count the sum of first 5 numbers.

```
s=0
for i in range(6):
  s+=i
print(s)
```

15

range() is used to generate sequence of numbers starting from 0 by default and increments by 1 (by default) and stops before a specified number.

Here range(6) is same as range(0,6).

51. Write a program to print all odd numbers from 1 to 10 using for loop.

```
for i in range(1,10,2):
  print(i)
```

1
3
5
7
9

range(1,10,2) – from 1 to 10 with step 2

52. Write a program to print all fruits present in this list fruits = ["apple", "banana", "cherry"].

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
  print(fruit)
```

apple
banana
cherry

53. Write a program to print each letter of this string `cherry`.

```
for letter in "cherry":
  print(letter)
```

```
c
h
e
r
r
y
```

Nesteds and Comprehensions

List Comprehension:

54. Write a program to add odd numbers from 1 to 10 into a list using list comprehension.

```
L = [odd for odd in range(1,10,2)]
print(L)
```

[1, 3, 5, 7, 9]

55. Write a program to add all letter ‘a’ from this string ‘banaanaa’ using list comprehension.

```
L=[letter for letter in 'banaanaa' if letter=='a']
print(L)
```

['a', 'a', 'a', 'a', 'a']

Set Comprehension:

56. Write a program to add all even numbers present in this list `input_list = [1,2,3,4,4,5,6,6,6,7,7]` using set comprehension.

```
input_list = [1,2,3,4,4,5,6,6,6,7,7]
S = {var for var in input_list if var%2 == 0}
print(S)
```

{2, 4, 6}

Dictionary Comprehension:

57. Write a program to add cubes of all odd elements from 1 to 10 in a key:value pairs using dictionary comprehension.

```
D={var:var**3 for var in range(1,10) if var % 2 != 0}
print(D)
```

{1: 1, 3: 27, 5: 125, 7: 343, 9: 729}

Generator Comprehension:

Its similar to list comprehension only difference is that generator use circular brackets ( ) like tuples. Generators don’t allocate memory for the whole list, Instead they generate each value one by one which is why they are memory efficient.

58. Write a program to add and print all even numbers from 1 to 10 using generator comprehension.

```
G=(var for var in range(1,11) if var % 2 ==0)
print(G)
for i in G:
  print(i)
```

```
<generator object <genexpr> at 0x00FE55D8>
2
4
6
8
10
```

First output is the generator object through which we can only iterate thus memory efficient.

Nested List: [[ ], [ ], [ ]]

59. Write a program to create a matrix like below: By taking inputs

```
matrix = [[0, 1, 2, 3, 4],
          [0, 1, 2, 3, 4],
          [0, 1, 2, 3, 4]]
```

```
matrix=[]
for i in range(3):
  matrix.append([int(n) for n in input().split()])
print(matrix)
```

OR (Using List Comprehension)

```
matrix=[[int(n) for n in input().split()] for i in range(3)]
```

OR

```
matrix=[list(map(int,input().split()))for i in range(1)]
```

```
0 1 2 3 4
0 1 2 3 4
0 1 2 3 4
[[0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]
```

Nested Tuple: ((),(),())

Since we cannot modify contents of tuples we have to convert the objects such as list, sets, generators, etc into a tuple in order to achieve the nested tuples through inputs.

60. Write a program to create nested tuple like this T = ((5,10),(20,30),(40,50)) through inputs.

```
T=tuple()
T=tuple(tuple(int(n) for n in input().split()) for i
in range(3))
print(T)
```

OR

```
T=tuple(tuple(map(int,input().split())) for i in
range(1))
```

```
5 10
20 30
40 50
((5, 10), (20, 30), (40, 50))
```

Nested Sets: (frozenset({ }), frozenset({ }))

Note: frozenset() is similar to a normal set except that it’s immutable.

61. Write a program to create nested sets of {1,2}, {3,4}, {5,6} by adding frozensets.

```
S=set()
S.add(frozenset((1,2)))
S.add(frozenset((3,4)))
S.add(frozenset((5,6)))
print(S)
```

{frozenset({3, 4}), frozenset({5, 6}), frozenset({1, 2})}

Nested Dictionary:

62. Write a program to create nested dictionary containing information as ‘Name’, & ‘Age’ of 2 students, Also access and print the Name of Student 2.

```
D={"Student1":{
   "Name":"ABC",
   "Age":18},
   "Student2":{
   "Name":"DEF",
   "Age":20},}

print(D)
print(D["Student2"]["Name"])
```

{'Student1': {'Name': 'ABC', 'Age': 18}, 'Student2': {'Name': 'DEF', 'Age': 20}}

DEF

Function

Its a block of code which is executed only when its called, Defined using def keyword, and also can have multiple arguments inside its parameters.

Syntax:

```
def function_name(argument1,argument2,.....argN):
  //Statements
```

63. Write a program to create a function SquareSum() with two arguments a and b, calculate their square of sum and store it into variable c and print it.

```
def SquareSum(a,b):
  return (a+b)**2
c=SquareSum(2,5)
print(c)
```

49

64. Write a program to count the number of arguments passed into a functions parameters.

```
def Count(*args):
  count=0
  for i in args:
    count+=1
  print(count)

Count(1,2,3,4,"Hello",6,['List'],8)
```

8

65. Write a program to print from 1 to 5 using recursion function.

```
def series(n):
  if n==0:
    return 0
  series(n-1)
  print(n)
series(5)
```

```
1
2
3
4
5
```

Lambda / Anonymous Function

While normal functions are defined using the def keyword, anonymous functions are defined using lambda keyword.

Syntax: lambda arguments: expression

*Example: lambda x : x*2*

66. Write a program to print the product of two numbers using lambda function with and without using the identifier.

```
product=lambda a,b:a*b
print(product(5,4))

print((lambda a,b:a*b)(5,5))
```

```
20
25
```

filter()

filter() function takes in any function and a list of arguments, it basically filters out the elements according to the conditions specified in a function.

67. Write a program to filter only the even numbers from this list [1,2,3,4,5,6,8,9] using filter and lambda function.

```
List1=[1,2,3,4,5,6,8,9]
print(list(filter(lambda x:x%2 == 0,List1)))
```

[2, 4, 6, 8]

It iterates through the list, applies the function checks for the condition given inside the function and if it evaluates to True then only it filters out the iterated elements.

map()

It also takes in a function and a list, unlike filter it doesn't filters out the elements instead it applies the changes to the elements according to the condition specified in a given function.

68. Write a program to get the squares of each number present in this list [1,2,3,5,6,7,8,9] using map and lambda function.

```
L=[1,2,3,5,6,7,8,9]
print(list(map(lambda x: x**2,L)))
```

[1, 4, 9, 25, 36, 49, 64, 81]

reduce()

This function performs a repetitive operations over the pairs of the iterable. It comes under functools module so we need to import it in order to used this function.

69. Write a program to calculate the cumulative sum of all the numbers in this list [1,2,5,6,4] using reduce and lambda function.

```
from functools import reduce
L=[1,2,5,6,4]
print(reduce(lambda a,b:a+b,L))
```

18

Object Oriented Programming

Class and Object

Class is a blueprint or a collection of an objects, and object is an entity that has state and behaviour for example: Animal is a class and dog, lion, and other animals are its objects.

Class keyword is used to define the class. Once defined we can create its object (instance).

Syntax for Class:

```
class Animal:
  pass
```

Syntax for Object:

```
dog = Animal()
```

Constructor

The method __init__() is used to create the constructor for a class. The self keyword is the first argument that is passed into its parameters which allows accessing the attributes and methods of a class so it can be considered as the reference.

Syntax:

```
def __init__(self,other_attr):
  pass
```

70. Write a program to create a class Dog, Initialize constructor with the dogs name and age and create method bark() which prints out “Woof!”+”Name of Dog” when called by its object and also print the name and age of the dog.

```
class Dog:
  def __init__(self, name, age):
    self.name=name
    self.age=age
  def bark(self):
    print("Woof!",self.name)

dog1=Dog("Hilo",5)
dog2=Dog("Timi",2)
dog1.bark()
dog2.bark()
```

Woof! Hilo
Woof! Timi

Inheritance

In Inheritance, the child class acquires the properties and can access all the data members and functions defined in the parent class.

Syntax:

```
class child_class(parent_class):
  pass
```

71. Write a program to create a class Animal, initialize local variable name and constructor to print ‘Friendly Animal’, create method info() which prints “Animal”+name then create another class as Dog inheriting the class Animal with its constructer printing “Friendly Dog” and method info() which prints “Dog”+name, Create one instance of class Dog and use its methods.

```
class Animal:
  name = 'XYZ'
  def __init__(self):
    print("Friendly Animal")
  def info(self):
    print("Animal",self.name)

class Dog(Animal):
  def __init__(self,name):
    self.name=name
    super().__init__()
    print("Friendly Dog")
  def info(self):
    print("Dog",self.name)

dog1 = Dog("Tilo")
dog1.info()
```

Friendly Animal
Friendly Dog
Dog Tilo

we used super() function inside child’s class init() method which allows to run the init() method or any variable or methods of the parent’s class.

Further the info() method of the parent class has benn overridden by the child’s class info() method.

Encapsulation

It restricts access to methods and variables, which helps in preventing data from direct manipulation. Underscore (_) or Double Underscore (__) is used to declare the private data.

72. Write a program to create a class Info and make password variable as private, try to change the value of password by accessing it directly and then by its method.

```
class Info:
  def __init__(self):
    self.__password ='Passwd'
  def printpass(self):
    print(self.__password)
  def changepass(self,newpass):
    self.__password=newpass

I1=Info()
I1.printpass()
I1.__password="Pass123"
I1.printpass()
I1.changepass("Pass123")
I1.printpass()
```

Passwd
Passwd
Pass123

So, as shown in the example above, its clear that the private member cannot be changed or accessed directly outside the class without its getters and setters, which in this case getters and setters are printpass() and changepass() respectively.

Polymorphism

Polymorphism is an ability to be used in more than one forms. Using method of same name for different classes but performing different tasks for each of them is an example of polymorphism.

73. Write a program to create two classes Parrot, Dog both having fly() as their method and call this method of both the classes through interface flying_test().

```
class Parrot:
  def fly(self):
    print("Parrot can fly")

class Dog:
  def fly(self):
    print("Dog can't fly")

def flying_test(obj):
  obj.fly()

P1=Parrot()
D1=Dog()

flying_test(P1)
flying_test(D1)
```

Parrot can fly
Dog can't fly

Exception handling

When an error occurs python breaks the entire program flow and shows the error logs. We can handle these errors or exceptions by using some built-in python exceptions, which will normally stop, generate some error message handle the exception according to what is specified and continues the running flow of program.

try: This block lets you to test a block of code which generally will throw some errors.

except: This block lets you handle the error.

finally: This block executes code, regardless of the try and except block results.

74. Write a program to divide a number by 0 and handle the exception with suitable defined error using try and catch statement and also print “Finished” at last using finally.

```
try:
  print(10//0)
except ZeroDivisionError:
  print("Number Cannot be divided by 0")
finally:
  print("Finished")
```

Number Cannot be divided by 0
Finished

75. Write a program to divide a string ‘10’ by 2, catch the exception and fix the code by converting the string into integer and then try to divide it by 2.

```
n='10'
try:
  print(n//2)
except TypeError:
  print("String Cannot be divided")
  print("Trying to convert string to int.....")
  print(int(n)//2)
```

String Cannot be divided
Trying to convert string to int.....
5

76. Write a program to accept an input and print the given input by convert it into integer, catch exception if there’s any else print the square of a number.

```
n=input("Enter Number: ")
try:
  print(int(n))
except ValueError:
  print("Cannot Convert It Into Int")
else:
  print(int(n)**2)
```

Enter Number: 5
25

Enter Number: five
Cannot Convert It Into Int

77. Write a program to input positive integer and print the square root of it, if it’s negative integer then raise an exception.

```
Import math
n=int(input("Enter Number: "))
try:
  if n<0:
    raise ValueError("This is not a positive
number")
except ValueError as e:
  print(e)
else:
  print(math.sqrt(n))
```

Enter Number: 16
4.0

Enter Number: -16
This is not a positive number

78. Write a program to input a number and show case exception handling for errors such as ValueError, IndexError and TypeError if it doesn’t lies for above stated errors then print ‘Error Occurred’.

```
#n=input("Enter Number: ")
#List1 = input("Enter List: ").split()
try:
  print(int(n))
  print(n%2)
  for i in range(3):
    print(List1[i])
  print(eval("Hello"**2))
except ValueError:
```

```
  print("Value Error: Cannot Convert String Into
Integer")
except TypeError:
  print("Type Error: Cannot Divide String by
Integer")
except IndexError:
  print("IndexError: Index Range Exceeded, No
Further Items In List")
except:
  print("Unknown Error")
```

Enter Number: two
Value Error: Cannot Convert String Into Integer

Enter Number: '2'
Type Error: Cannot Divide String by Integer

Enter List: 1 2
Index Error: Index Range Exceeded, No Further Items In List

Local and Global Scope

79. Write a program to print global variable x=200 as well as local scope function variable x=100.

```
x=200
def localfunc():
  x=100
  print("Local x:",x)
print("Global x:",x)
localfunc()
```

Global x: 200
Local x: 100

80. Write a program to change the value of global variable inside a function then print it at the end of the program.

```
x = 200
def localfunc():
  global x
  x=100
  print("Global x changed:",x)
print("Before Running Function, x:",x)
localfunc()
print("Global x:",x)
```

Before Running Function, X: 200
Global x changed: 100
Global x: 100