day1

[1]:
```
a=1
```

[2]:
```
a
```

[2]:
```
1
```

[3]:
```
print(a)
```

```
1
```

1 heading 1 by selecting markdown, using hashtag (#), we for description and heading for details

1.1 using double hashtag (##)

using four hashtag (####)

1. sjfl;
2. dlldje

[11]:
```
## double hash compiler understand its a single line comment
# single comment for programmer to see, writing on code
""" multi line comment
we use double goutes 3 times
"""
a
```

[11]:
```
1
```

[12]:
```
print("skjdf  ")
```

```
skjdf
```

[17]:
```
b = 3
##  only typing b is not showing output on output console its just in code␣
 ↪doing nothing

b
```

[17]: 3

[16]:
```
# ptint inbuilt funtion will give output on output console
print(b)
```

```
3
```

[23]:
```
print("jshdjkfdjd \n '\ n' for next line statement")
```

```
jshdjkfdjd
 '\ n' for next line statement
```

2 assigning variable

[25]:
```
name= "abcd"
name
```

[25]: 'abcd'

[28]:
```
name = "djhfd"
# overwriting or riding name variable with new data
name
```

[28]: 'djhfd'

[29]:
```
type(name)
# type function print type of written variable
```

[29]: str

[32]:
```
type("djfhdk:")
```

[32]: str

[33]:
```
type(23)
```

[33]: int

[34]:
```
type(20.3)
```

[34]: float

[36]:
```
c=34
c
```

[36]: 34

[37]:
```
c= 20
c
```

[37]: 20

[38]:
```
type(c)
```

[38]: int

[39]:
```
decimel_num= 3.4
decimel_num
```

[39]: 3.4

[40]:
```
type(decimel_num)
```

[40]: float

[41]:
```
complex_num= 1+2j
complex_num
```

[41]: (1+2j)

[42]:
```
type(complex_num)
```

[42]: complex

[44]:
```
#variable names are case sensitive
company= "steam"
Company="treding"
print(company)
print(Company)
```

```
steam
treding
```

[45]:
```
# reserved inbuilt keywords
""" int, float, len, complex, return, bool, str, yeild """
```

[45]: ' int, float, len, complex, return, bool, str, yeild '

3 boolen

1. true or not false
2. false or not true ### logic gates
3. and
4. or

5. not

[51]:
```
bool(0)
```

[51]:
```
False
```

[52]:
```
bool(1)
```

[52]:
```
True
```

[53]:
```
type(True)
```

[53]:
```
bool
```

type(not True)

[56]:
```
a=10
if bool(a)==True:
    print('true')
```

```
true
```

4 Typecasting

changing type of a variable by using inbuilt funtion str, bool, int, float

[58]:
```
type(str(100))
```

[58]:
```
str
```

[59]:
```
type(int('23'))
```

[59]:
```
int
```

5 dynamic type

on run time compiler understands variable is integer, float, string

[62]:
```
a= 10
str1 = "sjhhf"
# we are not defining variable type like C, C++, java, python automatically␣
  ↪understand on run run time
print(type(a))
print(type(str1))
```

```
<class 'int'>
<class 'str'>
```

[63]:
```
a= 'djhf'
#dynamically changing variable type
type(a)
```

[63]:
```
str
```

[64]:
```
int(1.54)
```

[64]:
```
1
```

6 concatination b/w different type

[65]:
```
'1'+'1'
```

[65]:
```
'11'
```

[66]:
```
'1'*'1'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[66], line 1
----> 1 '1'*'1'

TypeError: can't multiply sequence by non-int of type 'str'
```

[68]:
```
'1'*5
```

[68]:
```
'11111'
```

[69]:
```
int('1')+int('1')
```

[69]:
```
2
```

[ ]:

day2 if else

[6]:
```
price= int(input("enter product price"))
if price>1000:
    dis=price*0.8
    print(f'new discount price is {dis}')
if price <=1000:
    dis = price*0.7
    print("new discount price is {}".format(dis))
```

```
enter product price 2000

new discount price is 1600.0
```

[7]:
```
product_price=int(input("Enter the price"))
if product_price>1000:
    print("The price of the product is {}".format(product_price*0.8))
else:
    print("The price of the product is {}".format(product_price*0.7))
```

```
Enter the price 100

The price of the product is 70.0
```

[8]:
```
product_price=int(input("Enter the price"))
if product_price>1000:
    print(f"The price of the product is {product_price*0.8}")
else:
    print("The price of the product is {}".format(product_price*0.7))
```

```
Enter the price 100

The price of the product is 70.0
```

[19]:
```
product_price=int(input("enter product price"))

if product_price>3000:
    if product_price==4000:
        print("congratulation you've won a trip to goa")
    print(f'discounted price is {product_price*0.8}')
```

```
elif product_price>=2000 and product_price<=3000:
    if product_price==2999:
        print("you'll get an additional gift")
    print("discounted price is {}".format(product_price*0.70))

elif product_price>=100 and product_price<2000:
    print('discounted price is',product_price*0.6)

else :
    print("no product available")
```

```
enter product price 99
no product available
```

[20]:
```
#loops
## while loop

### atm machine with thousand rupees

total_amount= 1000
while total_amount!=0:
    print(total_amount)
    total_amount= total_amount-100
else:
    print("atm doesn't have cash")
```

```
1000
900
800
700
600
500
400
300
200
100
atm doesn't have cash
```

[ ]:

day2 starloops

[20]:
```
n= 7
for i in range(0,n):
    for j in range(1,i+1):
        print(j,end=' ')
    print()
```

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
```

[74]:
```
n= 5
#x = n-1
for i in range(0,n):
    x = n-j
    for j in range(0,x):

        print(end=" ")
    #x=x-1
    for j in range(1,i+1):
        print(j,end=' ')
    print()
```

```
    1
   1 2
  1 2 3
 1 2 3 4
```

[77]:
```
n= 5
x = n-1
for i in range(0,n):
    #x = n-j
    for j in range(0,x):

```

```
        print(end=" ")
    x=x-1
    for j in range(1,i+1):
        print(j,end=' ')
    print()
```

```
   1
  1 2
 1 2 3
1 2 3 4
```

[46]:
```
#python3 code to implement above approach
height = 5
for row in range(1, height+ 1):
    print(" " * (height - row) +"*" * row)
```

```
    *
   **
  ***
 ****
*****
```

[61]:
```
for i in range(0,4):
    print(i)
```

```
0
1
2
3
```

[62]:
```
for i in range(0,4): print(i)
```

```
0
1
2
3
```

[123]:
```
num_row=5
row=0
while(row<num_row):
    row +=1
    space = num_row-row
    while(space>0):
        print(end=" ")
        space-=1
   # rc=1
```

```
    star =  2*row-2
    while(star>0):
        print(row, end="")
        star -= 1
       # rc+=1

    print(row)
```

```
    1
   222
  33333
 4444444
555555555
```

[ ]:

[ ]:

day3_4

[13]:
```
start= True
stop = False

#print value as it is
print(f"the Defined value of start is {start}")
print(f"Defined value of stop is {stop} \n")

# also be written as
print(f"negating defined value of start is :{not start}")
#not able to understand negating, now got its nulling or oppositing defined␣
 ↪value
print("defined value of start is {} \n".format(start))

#vice versa
print("defined value of stop is {} ".format(stop))
print('negating value of stop is {} '.format(not stop))
```

```
the Defined value of start is True
Defined value of stop is False

negating defined value of start is :False
defined value of start is True

defined value of stop is False
negating value of stop is True
```

[25]:
```
not int(bool(0))
```

[25]:
```
True
```

[26]:
```
some_negative_integer = -3
some_positive_integer=+3

print(f'boolean value of {some_negative_integer} is␣
 ↪{bool(some_negative_integer)}')
print(f'boolean value of {some_positive_integer} is␣
 ↪{bool(some_positive_integer)} \n')
```

```
print(f'negation of {some_negative_integer} is {not␣
 ↪bool(some_negative_integer)} \
    and negation of {some_positive_integer} is {not␣
 ↪bool(some_positive_integer)}')

print('\n---------------------------------\n')
```

```
boolean value of -3 is True
boolean value of 3 is True

negation of -3 is False     and negation of 3 is False

---------------------------------
```

[28]:
```
#logic gate and,or,not\

not(False)
```

[28]:
```
True
```

[30]:
```
not(True) * False
```

[30]:
```
True
```

[40]:
```
course_name= "data sciEnce masters"
print(len(course_name))
course_name.title()
```

```
20
```

[40]:
```
'Data Science Masters'
```

[41]:
```
course_name.split('s')
```

[41]:
```
['data ', 'ciEnce ma', 'ter', '']
```

[42]:
```
course_name.partition('s')
```

[42]:
```
('data ', 's', 'ciEnce masters')
```

[43]:
```
course_name.upper()
```

[43]:
```
'DATA SCIENCE MASTERS'
```

[44]:
```
course_name.lower()
```

[44]:
```
'data science masters'
```

[45]:
```
course_name.swapcase()
```

[45]:
```
'DATA SCIeNCE MASTERS'
```

[47]:
```
course_name.count('s')
```

[47]:
```
3
```

[48]:
```
#slice or slicing

course_name[5:12]
```

[48]:
```
'sciEnce'
```

[50]:
```
course_name[-9:4:-1]
```

[50]:
```
'ecnEics'
```

[52]:
```
course_name[-9:-16:-1]
```

[52]:
```
'ecnEics'
```

[53]:
```
course_name[11:4:-1]
```

[53]:
```
'ecnEics'
```

[54]:
```
course_name
```

[54]:
```
'data sciEnce masters'
```

[20]:
```
num_row=5
for i in range(0,num_row):
    print(" "*(num_row-i),end="")
    for j in range(i):
        print(j,end=" ")
    print(i)
```

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

[10]:
```
num_row=5
for i in range(0,num_row):
```

```
    print(" "*(num_row-i),end="")
    for j in range(i*2):
        print(j,end="")
    print(i+1)
```

```
     1
    012
   01233
  0123454
 012345675
```

[135]:
```
a="dkj12fkOP The"
a
```

[135]:
```
'dkj12fkOP The'
```

[39]:
```
a.isalnum()
```

[39]:
```
False
```

[40]:
```
a.isalpha()
```

[40]:
```
False
```

[42]:
```
a.capitalize()
```

[42]:
```
'Dkj12fkop the'
```

[51]:
```
a.find('1')
```

[51]:
```
3
```

[52]:
```
a.split()
```

[52]:
```
['dkj12fkOP', 'The']
```

[53]:
```
b="Iam a Super Hero"
```

[54]:
```
b.split()
```

[54]:
```
['Iam', 'a', 'Super', 'Hero']
```

[55]:
```
a.
```

[55]:
```
False
```

[3]:
```
lst_1=["apple","banana","mango","grapes",1,12]
tupl_1=(5,78,"egg","fish","chicken")
set_1={}
```

[69]:
```
type(set_1)
```

[69]:
```
dict
```

[70]:
```
set_1={"ginger","garlic","chilli",66 ,69}
type(set_1)
```

[70]:
```
set
```

[9]:
```
dic_1={"a":"manga", "b":"bongo", "e":"salsa", "d":90,"5":88}
```

[10]:
```
print(dic_1)
```

```
{'a': 'manga', 'b': 'bongo', 'e': 'salsa', 'd': 90, '5': 88}
```

[11]:
```
dic_1.keys()
```

[11]:
```
dict_keys(['a', 'b', 'e', 'd', '5'])
```

[13]:
```
#extra to get keys of dictionary, it creates immutable and
# not duplicate set of keys
key=frozenset(dic_1)
key
```

[13]:
```
frozenset({'5', 'a', 'b', 'd', 'e'})
```

[ ]:
```

```

[87]:
```
dic_1.items()
```

[87]:
```
dict_items([('a', 'manga'), ('b', 'bongo'), ('e', 'salsa'), ('d', 90), ('5',
88)])
```

[89]:
```
print(lst_1)
print(tupl_1)
print(set_1)
print(dic_1)
```

```
['apple', 'banana', 'mango', 'grapes', 1, 12]
(5, 78, 'egg', 'fish', 'chicken')
{66, 'ginger', 69, 'garlic', 'chilli'}
{'a': 'manga', 'b': 'bongo', 'e': 'salsa', 'd': 90, '5': 88}
```

[99]:
```
lst_1
```

[99]:
```
['apple', 'banana', 'mango', 'grapes', 1, 12]
```

[102]:
```
for i in lst_1:
    print(i)
```

```
apple
banana
mango
grapes
1
12
```

[8]:
```
for i in range(len(lst_1)):
    print(       lst_1[i],end="")
    if i in range(len(tupl_1)):
        print(" \t", tupl_1[i])
   # print(" \t", set_1[i],end="")   error
  #  print(" \t", dic_1[i],end="")   # error
```

```
apple    5
banana   78
mango    egg
grapes   fish
1        chicken
12
```

[6]:
```
a
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[6], line 1
----> 1 a

NameError: name 'a' is not defined
```

[126]:
```
#iterating reverse

#for i in range(le
ch=len(a)-1
while(ch>=0):
    print(a[ch])
    #ch= ch-1
    ch-=1
```

```
e
```

```
h
T

P
O
k
f
2
1
j
k
d
```

[130]:
```
a[12:9:-2]
```

[130]:
```
'eT'
```

[141]:
```
a
```

[141]:
```
'dkj12fkOP The'
```

[134]:
```
a
```

[134]:
```
'dkj12fkOP Thedjkdjk'
```

[138]:
```
lst_1[-4]
```

[138]:
```
'mango'
```

[147]:
```
for i in range(len(a)):
    print(a[len(a)-1-i])
#    print(a[len(a)-(1+i)])
```

```
e
h
T

P
O
k
f
2
1
j
k
d
```

[155]:
```
name="maharaja partape"
vowels="AaEeIiOoUu"

for i in name:
    if i in vowels:
        print(i,"its a vowel")
    else:
        print(i,"isn't voewl")
```

```
m isn't voewl
a its a vowel
h isn't voewl
a its a vowel
r isn't voewl
a its a vowel
j isn't voewl
a its a vowel
  isn't voewl
p isn't voewl
a its a vowel
r isn't voewl
t isn't voewl
a its a vowel
p isn't voewl
e its a vowel
```

[175]:
```
[i+' is vowel' if i in ("aAeEiIoOuU") else i+"  not vwls" for i in name]
```

[175]:
```
['m  not vwls',
 'a is vowel',
 'h  not vwls',
 'a is vowel',
 'r  not vwls',
 'a is vowel',
 'j  not vwls',
 'a is vowel',
 '   not vwls',
 'p  not vwls',
 'a is vowel',
 'r  not vwls',
 't  not vwls',
 'a is vowel',
 'p  not vwls',
 'e is vowel']
```

[176]:
```
lst_1
```

[176]:
```
['apple', 'banana', 'mango', 'grapes', 1, 12]
```

[179]:
```
if "apple" in lst_1:
    print('present')
```

```
present
```

[186]:
```
"""for i in range(len(lst_1)):
    if "apple" in lst_1:
        print(lst_1[i])"""

for element in lst_1:
    if element=="apple":
        print(element)
```

```
apple
```

[194]:
```
#list comprehention
[element for element in lst_1 if element=="apple"]
```

[194]:
```
['apple']
```

[195]:
```
# append only single value or single list or tuple add at the end of
#    given list
# concatenation + create new list and merge them
# extend merge list with order iteration
# insert
```

append(): append the element to the end of the list.

insert(): inserts the element before the given index.

extend(): extends the list by appending elements from the iterable.

List Concatenation: We can use the + operator to concatenate multiple lists and create a new list.

1 assignment

[202]:
```
# sum of even and odd no.
lst_2 = [1,2,3,4,5,6,7,8,9,10]
sum_e=0
sum_o=0
for i in lst_2:
    print(i)
    if i%2==0:
```

```
        sum_e=sum_e+i #sum_e += i
    else:
        sum_o=sum_o+i
print(sum_e,'even add')
print(sum_o,'odd add')
```

```
1
2
3
4
5
6
7
8
9
10
30 even add
25 odd add
```

[203]:
```
lst_2
```

[203]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

[234]:
```
sum_e=0
sum_o=0

[i+sum_e for i in lst_2 if i%2==0]
```

[234]:
```
[2, 4, 6, 8, 10]
```

[236]:
```
sum_eve= sum([i for i in lst_2 if i%2==0])
sum_eve
```

[236]:
```
30
```

[244]:
```
odd_sum=sum([od for od in lst_2 if od%2!=0])
odd_sum
```

[244]:
```
25
```

[245]:
```
# formula to convert celsius to fahernheit(9/5)*temp+32

# Example 4: Convert a list of temperatures from Celsius to Fahrenheit uing␣
 ↪list comprehesnion
celsius_temperatures = [0, 10, 20, 30, 40, 50]
```

[246]:
```
[(9/5)*i+32 for i in celsius_temperatures]
```

[246]: [32.0, 50.0, 68.0, 86.0, 104.0, 122.0]

[247]:
```
[(9/5)*temp+32 for temp in celsius_temperatures]
```

[247]: [32.0, 50.0, 68.0, 86.0, 104.0, 122.0]

[251]:
```
# Example 3: Create a list of only the first letters of words in a list
words = ['apple', 'banana', 'cherry', 'date']

[word[0] for word in words]
```

[251]: ['a', 'b', 'c', 'd']

[272]:
```
# Example 5: Flatten a list of lists into a single list
lists = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
#x=[]
print(lists[1][2])

[cl for rw in lists for cl in rw ]
```

```
6
```

[272]: [1, 2, 3, 4, 5, 6, 7, 8, 9]

[ ]:
```
## Assignment
## Using both code and list comprehesnion
# Example 2: Create a list of only the prime numbers from a given list
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

for i in range(2,max(numbers)+1):
    #print(i)
    cnt=0
    for j in numbers:
        if j%i==0:
            cnt+=1
            #print(j)
        if cnt==0:
            print(J)
```

```
2
3
3
4
5
4
5
6
7
```

```
5
6
7
8
9
6
7
8
9
10
7
8
9
10
8
9
10
9
10
10
```

[29]:
```python
## Assignment
## Using both code and list comprehesnion
# Example 2: Create a list of only the prime numbers from a given list
numbers = [1,2, 3, 4, 5, 6, 7, 8, 9, 10]

for num in numbers:
    cnt=0
    for iteratn in range(2,num+1):
        #print(iteratn)
        if num%iteratn==0:
            cnt+=1
           # print(num%iteratn)
    if cnt==1:
        print(num)
```

```
2
3
5
7
```

[33]:
```python
lis=[1,4,10,12,11,31,77,67] # num= int(input("enter no. to check prime")
prime=[]

for i in lis:
    c=0
```

```
    for j in range(1,i):#range from 1 to i means 1 to a number before value so␣
 ↪if it's only divisble by 1
                        # and other lower numbers are unable to get 0 remainder␣
 ↪than it's a prime no.
        #print(j)
        if i%j==0:
            c+=1
    if c==1:
        prime.append(i)

print(prime)
```

```
[11, 31, 67]
```

[34]:
```
# quora copied

def prime(n):
    if n <= 0:
        return "Not defined"
    elif n == 1:
        return "Not prime"
    for i in range(2, n): # this range is giving one lower number of n so if␣
 ↪not any no. divides, it wont return 0
        if n%i == 0:
            return "not prime" #range from 1 to i means 1 to a number before␣
 ↪value so if it's only divisble by 1
                        # and other lower numbers are unable to get 0 remainder␣
 ↪than it's a prime no.
    return "prime"



def list_prime(lst):
        prime_list =[ ]
        for i in lst:
                x = prime(i)
                if x == "prime":
                        prime_list.append(i)
        return prime_list

print(list_prime([1,2,3,4,5,6,7,8,9]))
```

```
[2, 3, 5, 7]
```

[38]:
```
[i for i in range(2,4)]
```

[38]:
```
[2, 3]
```

[47]:
```
## Assignment
## Using both code and list comprehesnion
# Example 2: Create a list of only the prime numbers from a given list
numbers = [1,2, 3, 4, 5, 6, 7, 8, 9, 10]

#[num for num in numbers for itratn in range(2,num) if num%num==0 and num%1==0␣
 ↪and num%itratn!=0]

[num for num in numbers if all(num%num==0 and num%1==0 and num%itratn!=0 for␣
 ↪itratn in range(2,num))]
```

[47]:
```
[1, 2, 3, 5, 7]
```

[46]:
```
#prime number list comprehension
#quora copied
[x for x in range(2, 20) if all(x % y != 0 for y in range(2, x))]
```

[46]:
```
[2, 3, 5, 7, 11, 13, 17, 19]
```

[48]:
```
all # keyword
```

[48]:
```
<function all(iterable, /)>
```

[49]:
```
[x for x in range(4, 10) if all(x % y != 0 for y in range(2, x-1))]
```

[49]:
```
[5, 7]
```

[50]:
```
#prime numbers find
numbers = [1,2, 3, 4, 5, 6, 7, 8, 9, 10]

for num in numbers:
    cnt=0
    for iteratn in range(1,num+1):

        if num%iteratn==0:
            cnt+=1

    if cnt==2: # 2 count because range is 1 to number +1 so number will be␣
 ↪divided by 1 and by itself so 2counts
        print(num)
```

```
2
3
5
7
```

[ ]:

day5_2_feb

[1]:
```
d={}
```

[2]:
```
type(d)
```

[2]:
```
dict
```

[5]:
```
d1={'name':'resheph',"email_id":"dkfjdk@gmail.com","number":23435}
```

[7]:
```
d1
```

[7]:
```
{'name': 'resheph', 'email_id': 'dkfjdk@gmail.com', 'number': 23435}
```

[10]:
```
type(d1)
```

[10]:
```
dict
```

[9]:
```
d2={"name":"Raa","name":"resheph"}
```

[11]:
```
#key should be unique, name 'raa' get updated with 'resheph'
d2
```

[11]:
```
{'name': 'resheph'}
```

[ ]:
```
bb
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[2]:
```
import math
d21={i: math.log10(i) for i in range(1,11)}
d21
```

[2]:
```
{1: 0.0,
 2: 0.3010299956639812,
 3: 0.47712125471966244,
 4: 0.6020599913279624,
 5: 0.6989700043360189,
 6: 0.7781512503836436,
 7: 0.8450980400142568,
 8: 0.9030899869919435,
 9: 0.9542425094393249,
 10: 1.0}
```

[3]:
```
{i for i in d21 if i.keys()%2==0}
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[3], line 1
----> 1 {i for i in d21 if i.keys()%2==0}

Cell In[3], line 1, in <setcomp>(.0)
----> 1 {i for i in d21 if i.keys()%2==0}

AttributeError: 'int' object has no attribute 'keys'
```

[4]:
```
{i for i in d21.items if i.keys()%2==0}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[4], line 1
----> 1 {i for i in d21.items if i.keys()%2==0}

TypeError: 'builtin_function_or_method' object is not iterable
```

[18]:
```
d21.keys()
```

[18]:
```
dict_keys([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
```

[24]:
```
{i:d21[i] for i in d21.keys() if i%2==0}
```

[24]:
```
{2: 0.3010299956639812,
 4: 0.6020599913279624,
 6: 0.7781512503836436,
```

```
 8: 0.9030899869919435,
 10: 1.0}
```

[38]:
```
atup = ("otkhs")
atup2=("21")

print(type(atup))
print(type(atup2))

print(atup2 is int)
print(atup2.isnumeric())
```

```
<class 'str'>
<class 'str'>
False
True
```

[ ]:

[ ]:

day6_3_feb_functn

[1]:
```
print("its in built function")
```

```
its in built function
```

[2]:
```
l=[1,2,3,4,5]
```

[3]:
```
len(l)
```

[3]: 5

[4]:
```
def test(): # def is just defination / defined name function test
            #follow nameing convention but not use reserved keyword
```

```
  Cell In[4], line 3

    ^
SyntaxError: incomplete input
```

[5]:
```
#without direction or suppose to do it gave error
def test():
    pass    # if not want to write inside funct,
```

[8]:
```
def test1():
    print("this is first function and difined its funct to print")
```

[11]:
```
test1
```

[11]: <function __main__.test1()>

[12]:
```
test1()
```

```
this is first function and difined its funct to print
```

[13]:
```
test1() + "resheph"
```

this is first function and difined its funct to print

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[13], line 1
----> 1 test1() + "resheph"

TypeError: unsupported operand type(s) for +: 'NoneType' and 'str'
```

[22]:
```
# if outcome of function is print
#then it is not gonna return any datatype
# only NonType, and with nontype we wont be able perform and operation
```

[23]:
```
#so have to use return keyword
def test2():
    return "return keyword will return this string "
```

[24]:
```
test2()
```

[24]:
```
'return keyword will return this string '
```

[25]:
```
test2() + "resheph"
```

[25]:
```
'return keyword will return this string resheph'
```

[26]:
```
def test3():
    return 1,4,"pwskills", 23.34
```

[28]:
```
test3() #call test3 funct, able to get multiple return from test3 functn
```

[28]:
```
(1, 4, 'pwskills', 23.34)
```

[29]:
```
a= 1,2,3,4,5
```

[30]:
```
a # allocation multiple value to single variable
```

[30]:
```
(1, 2, 3, 4, 5)
```

[31]:
```
type(a)
```

[31]:
```
tuple
```

[32]:
```
a,b,c,d = 1,2,34.56,True
```

[34]:
```
# a is holding 1, b holding 2 .... at a time alocating multiple value to␣
↪multiple variable
```

```
# like we got multiple data type return from functn
a
```

[34]: 1

[35]:
```
b
```

[35]: 2

[36]:
```
c
```

[36]: 34.56

[37]:
```
d
```

[37]: True

[39]:
```
test3() #functn return multiple datatype at a time in form of tuple
```

[39]: (1, 4, 'pwskills', 23.34)

[40]:
```
test3()[0]
```

[40]: 1

[41]:
```
test3()[1]
```

[41]: 4

[43]:
```
test3()[2]
```

[43]: 'pwskills'

[44]:
```
test3()
```

[44]: (1, 4, 'pwskills', 23.34)

[45]:
```
a,b,c,d = test3()
```

[46]:
```
#assign test3() four values to four variable
```

[47]:
```
a
```

[47]: 1

[48]:
```
b
```

[48]:
```
4
```

[50]:
```
c
```

[50]:
```
'pwskills'
```

[51]:
```
d
```

[51]:
```
23.34
```

1 function logic

[2]:
```
def test4():
    a = 3*4+5
    return a
```

[3]:
```
test4()
```

[3]:
```
17
```

[4]:
```
type(test4())
```

[4]:
```
int
```

[5]:
```
def test5(a,b):
    c = a+b
    return c
```

[6]:
```
# calling and passing value to a and b variable in test5()
test5() #didnt pass 2 parameter that required
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[6], line 2
      1 # calling and passing value to a and b variable in test5()
----> 2 test5() #didnt pass 2 parameter that required

TypeError: test5() missing 2 required positional arguments: 'a' and 'b'
```

[7]:
```
test5(1,3) # perform addition
```

[7]:
```
4
```

[8]:
```
test5("resheph","Gani") # perform concationation
```

[8]:
```
'reshephGani'
```

[9]:
```
test5([1,2,3,4],[4,5,6,7,8])# perform concationation
```

[9]:
```
[1, 2, 3, 4, 4, 5, 6, 7, 8]
```

[10]:
```
test5(b="resheph",a="gani") # passing value with variable to print oppsite
```

[10]:
```
'ganiresheph'
```

[45]:
```
l= [1,2,3,4,'pw','skills',[11,21,31,41.51]]
```

[46]:
```
#create a function to take input as a list and give final value as a sigle list␣
↪containing only numeric
```

[47]:
```
def test6(a):
    #print(a)
    ls=[]
    for x in a:
        if type(x) is int or type(x) is float:
            #print(x)
            ls.append(x)
    return ls
```

[48]:
```
test6(l)
```

[48]:
```
[1, 2, 3, 4]
```

[57]:
```
def test7(a):
    ls=[]
    #print(a)
    for i in a:
        if type(i) == list:
            for j in i:
                if type(j) == int or type(j)== float:
                    ls.append(j)
        else:
            if type(i) == int or type(i) == float:
                ls.append(i)
    return ls
```

[58]:
```
test7(l)
```

[58]:
```
[1, 2, 3, 4, 11, 21, 31, 41.51]
```

[59]:
```
def test7(a):
    ls=[]
    #print(a)
    for i in a:
        if type(i) is int or type(i) is float:
            ls.append(i)
        elif type(i) is list:
            for j in i:
                 if type(j) is int or type(j)is float:
                    ls.append(j)

    return ls
```

[60]:
```
test7(l)
```

[60]:
```
[1, 2, 3, 4, 11, 21, 31, 41.51]
```

2 function part 2

[61]:
```
def test(a,b,c,d):
    pass
```

[62]:
```
test(1,2,3,4)
```

[63]:
```
test(4,5,6,7,8,9,2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[63], line 1
----> 1 test(4,5,6,7,8,9,2)

TypeError: test() takes 4 positional arguments but 7 were given
```

2.0.1 if want to give n number of argument to function then use '*'

[68]:
```
def test1(*args): #can write any word after *
    return args
```

[69]:
```
test1(3)
```

[69]:
```
(3,)
```

[70]:
```
test1(1,2,3,4,5)
```

[70]:
```
(1, 2, 3, 4, 5)
```

[71]:
```
test1('reshesph',[1,2,3,5,6],(1,2,3,4,5))
```

[71]:
```
('reshesph', [1, 2, 3, 5, 6], (1, 2, 3, 4, 5))
```

2.0.2 now can put n number of argument in a function

[72]:
```
def test2(*resh):
    return resh
```

[74]:
```
test2(2,3,4,5)
```

[74]:
```
(2, 3, 4, 5)
```

can write any word after * and it give n number of value return as tuple

[79]:
```
def test3(*args,a ):
    return args, a
```

[80]:
```
test3(3)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[80], line 1
----> 1 test3(3)

TypeError: test3() missing 1 required keyword-only argument: 'a'
```

for args can pass n number of data directly but after that we have to write variable name to pass data

[81]:
```
test3(2.3,4,5,6, a ='resheph')
```

[81]:
```
((2.3, 4, 5, 6), 'resheph')
```

[84]:
```
def test4(*args):
    l = []
    for i in args:
        if type(i) == list:
            l.append(i)
    return l
# only extract list from all argument passed
```

[85]:
```
test4(1.2,4,5, "er","resheph",[3,4,5,],(3,'5',[5,'f']))
```

[85]:
```
[[3, 4, 5]]
```

[87]:
```
test4(1,2,3,[1,2,3,4,4],(1,2,34,4),"resheph",[4,5],[6,7,8])
```

[87]:
```
[[1, 2, 3, 4, 4], [4, 5], [6, 7, 8]]
```

2.1 if want to pass n number of records or arguments in key and pair

use '**' double astrik

[88]:
```
def test5(**kwargs):
    return kwargs
```

[90]:
```
test5() # return empty dictionary
```

[90]:
```
{}
```

[93]:
```
type(test5())
```

[93]:
```
dict
```

[92]:
```
test5('kd','dj')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[92], line 1
----> 1 test5('kd','dj')

TypeError: test5() takes 0 positional arguments but 2 were given
```

[95]:
```
test5(a=35,b=23, c= [1,2,3,4], d=("resheph", "gani")) # now return key value␣
↪pair
```

[95]:
```
{'a': 35, 'b': 23, 'c': [1, 2, 3, 4], 'd': ('resheph', 'gani')}
```

[123]:
```
def test6(**kwargs):
    l=[]
    for i in kwargs:
        print(i)
        if type(kwargs[i])==list:
            l.append(kwargs[i])
    return l
```

[120]:
```
test6(a = 1, e = 4, b = [1,23,], c= (1,3,'2',4), d = "resheph")
```

```
a
e
```

```
b
c
d
```

[120]:
```
[[1, 23]]
```

[129]:
```
def test6(**kwargs):

    for i in kwargs.keys():

        if type(kwargs[i])==list:
            return i, kwargs[i]

# now will return key and value pair
```

[130]:
```
test6(a=34,b=23,c=[1,2,3,4],d=("resheph", "pwskills"))
```

[130]:
```
('c', [1, 2, 3, 4])
```

[135]:
```
def test7(*args, **kwargs):
    return args, kwargs

# now can pass n number of arguments and key value pair
```

[136]:
```
test7(2,3,4,5, a=35, b=98)
```

[136]:
```
((2, 3, 4, 5), {'a': 35, 'b': 98})
```

3 generator function

[137]:
```
range(1,10) # its only giving object not data
```

[137]:
```
range(1, 10)
```

[138]:
```
for i in range(1,10):
    print(i) # now able to get data
```

```
1
2
3
4
5
6
7
8
9
```

[139]:
```
l= [1,2,3,4,4,5,6,7,8,7,"resheph", "pwskills"]
```

[142]:
```
def test1(a):
    n=[]
    for i in a:
        if type(i)==int:
            n.append(i)
    return n

# this function take infinte number of time if list is big
# because its going to give a final single list after processing all data
# till than it wont terminate, so for millions of data it will take infinte time
# it wont terminate till it prepare whole single list
# so its prepare the data than it gives output

# so solution is range() kind of fumction, becasue its givig one by one output␣
 ↪unlike test1(),
# range() is generating output one by one, not holding all the data on memory
# range() type functn doesnt remember all data, just last one that is give as␣
 ↪output and again iterate
```

[143]:
```
test1(l)
```

[143]:
```
[1, 2, 3, 4, 4, 5, 6, 7, 8, 7]
```

fibonacci series:

```
0,1,1,2,3,4,5,13,21
# for fibonacci also generating series of data
```

range is also generator function so lets create a generator function

[145]:
```
def test_fib(n):
    a,b = 0 ,1
    for i in range(n):
        yield a
        a,b=b,a+b
```

[146]:
```
test_fib(10)
```

[146]:
```
<generator object test_fib at 0x7f58162faab0>
```

so its giving generator object because now its a generator functn we created

3.0.1 using yield keyword

have to inside for loop to access date inside generator function

3.0.2 *yielding keep on giving me data/outcome one by one when call it inside loop, instead of holding the entire data in a single list

4 wont hold many space for data, only last data in a memory, its more optimized

[148]:
```
for i in test_fib(10):
    print(i)
```

```
0
1
1
2
3
5
8
13
21
34
```

[155]:
```
def test_fib1(n):
    a = 0
    b = 1
    for i in range(n):
        yield a
        temp= a
        a=b
        b=temp+b
```

[159]:
```
for i in test_fib1(7):
    print(i)
```

```
0
1
1
2
3
5
8
```

havent write return because yield is giving result one by one or giving record one by one, one by one

4.0.1 The Yield keyword in Python is similar to a return statement used for returning values or objects in Python. However, there is a slight difference. The yield statement returns a generator object to the one who calls the function which contains yield, instead of simply returning a value.

[ ]:

day8_5_feb_oops

1 OOPS stands for Object-Oriented Programming System

[1]:
```
a = 1
```

[2]:
```
print(type(a))
```

```
<class 'int'>
```

[3]:
```
print(type("pwskills"))
```

```
<class 'str'>
```

[4]:
```
# why each type print class then type of variable
# class is blueprint- or classification of real world entity
```

[7]:
```
class test:
    pass

a =test()
print(type(a))
```

```
<class '__main__.test'>
```

[9]:
```
class pwskills:

    def welcome_msg():
        print("welcome to pwskills")

rohan = pwskills()
rohan.welcome_msg()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[9], line 7
      4         print("welcome to pwskills")
      6 rohan = pwskills()
----> 7 rohan.welcome_msg()
```

```
TypeError: pwskills.welcome_msg() takes 0 positional arguments but 1 was given
```

[10]:
```
class pwskills:

    def welcome_msg(self):   #if declaring function inside class have give␣
 ↪pointer or reference
                             #that this method/functn belong to me//// or to␣
 ↪bind that functn with class
                             #(self) have to use
        print("welcome to pwskills")

rohan = pwskills()
rohan.welcome_msg()
```

```
welcome to pwskills
```

[11]:
```
sohan=pwskills()
sohan.welcome_msg()
```

```
welcome to pwskills
```

[ ]:
```
# create a class will take different different data for diffn diffn class␣
 ↪variable
#just like
a = 1
b=2 # for int class, we are able to give data to variable in in t class
#then have to use constructor,
# constructor is something that help class to take data

class pwskills1:

    def __init__(self,phone_number, email_id, student_id): # __init__ is␣
 ↪constructor or in-buit method, 'self' have to write for declaring
                                                          #this method belong␣
 ↪to this class ,,, or give reference to class that this method belong to this␣
 ↪class with 'self'
        self.phone_number = phone_number
        self.email_id = email_id
        self.student_id = student_id
```

[18]:
```
my_dict = {"a": 1, "b": 2, "c": 3}

my_set = set(my_dict.keys())
```

```
my_set.add("d")

print(my_dict)
```

```
{'a': 1, 'b': 2, 'c': 3}
```

[24]:
```
print(my_set)

my_set2=my_dict.keys()
#my_set2[1]
my_dict['a']
```

```
{'a', 'b', 'd', 'c'}
```

[24]:
```
1
```

[39]:
```
my_dict.update({'e':''})
```

[40]:
```
my_dict
```

[40]:
```
{'a': 1, 'b': 2, 'c': 3, 'e': ''}
```

[ ]:
```

```

day9_8_feb_oops2

1 Polymorphism

[5]:
```
def test(a,b):
        return a+b
```

[6]:
```
test(4,5)
```

[6]:
```
9
```

[8]:
```
test("resheph","xxxx") # this addition function is performing concatination
        #not adding, this is polymorphism
```

[8]:
```
'reshephxxxx'
```

[ ]:
```
test([1,2,3,4],[3,4,5,6,7]) # now performing list concatination or list␣
↪appending
                # this is polymorphism, have multiple states or have multiple␣
↪characteristics
                # according to data , or as per the data
```

1.0.1 now how to achieve polymorphism concept in acase of class and object

[2]:
```
class data_science:
    def syllabus(self):
        print("this is my method for data science syllabus")
```

[3]:
```
class web_dev:
    def syllabus(self):
        print("this is my method for web dev")
```

[9]:
```
def class_parser(class_obj):
    for i in class_obj:
        i.syllabus()
```

[10]:
```
obj_data_science= data_science()
```

[11]:
```
obj_web_dev = web_dev()
```

[13]:
```
class_obj = [obj_data_science,obj_web_dev] # in a single list or wrapper
class_parser(class_obj) #pass list of object, as what object i pass it'll␣
 ↪change its behavior

# a function is behaving differently for different class-object, POLYMORPHISM
```

```
this is my method for data science syllabus
this is my method for web dev
```

[ ]:
```

```

2 Encapsulation

[14]:
```
class test:
    def __init__(self, a,b):
        self.a=a
        self.b=b
```

[18]:
```
t = test(45,5)
```

[19]:
```
t.a
```

[19]: 45

[20]:
```
t.b
```

[20]: 5

[23]:
```
t.a=3

t.a
```

[23]: 3

able to easily access variable inside class-object, which is porne to hack

so to secure code we use encapsulation

so no one can access and chage varible/properties/arguments of class-object

[ ]:
```

```

[ ]:
```

```

[61]:
```
class car:

    def __init__(self,year,make,model, speed):
        self.__name = year #with double underscore its now private variable,␣
↪not public
        self.__model= make # user cannot access this properties, on programmer␣
↪because he knows variable name
        self.__make = model # with getter and setter method user can see and␣
↪change value but couldnt do directly
        self.__year = speed # because user dont know variable name and class␣
↪name ('_car__year' can change value)

    def set_speed(self,speed): # if want to give user ability to change
        self.__speed = 0 if speed < 0 else speed #this if else is working

    def get_speed(self): # if want to give user to see object properties
        return self.__speed

    def get_model(self): # if want to give user to see object properties
        return self.__model
```

[62]:
```
obj_car = car(2023,"RR","Ghost",70)
```

[63]:
```
obj_car.year # because self.__year , user cannot access
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[63], line 1
----> 1 obj_car.year # because self.__year , user cannot access

AttributeError: 'car' object has no attribute 'year'
```

[64]:
```
obj_car._car__year # only by knowing variable and class name and using _ class␣
↪_ _ varible
                   # can change value inside, and only developer know it
```

[64]:
```
70
```

[65]:
```
obj_car._car__year = 15
```

[66]:
```
obj_car._car__year
```

[66]:
```
15
```

[67]:
```
obj_car.set_speed(56) # because we made method to change speed value for user
```

[68]:
```
obj_car._car__speed
```

[68]: 56

[49]:
```
obj_car._car__speed #only by knowing class and variable name can access so now␣
↪make
                    # a class method to view properties/value for user
```

[49]: -56

[71]:
```
obj_car.get_speed() # now user can see the change also
```

[71]: 56

[73]:
```
obj_car.get_model() # havent made any method to change model so user cant␣
↪change it, only view it
```

[73]: 'RR'

example 2

[81]:
```
class bank_account:

    def __init__(self, balance):
        self.__balance = balance # if someone know name of balance variable and␣
↪directily access it\
                                 # then they can manupulate entire system,␣
↪method, or purpose of making all this classes
                                 #to secure this we use encapsulation

    def deposit(self,amount):
        self.__balance = self.__balance + amount

    def withdraw(self, amount):
#         self.__balance = 'insufficien' if self.__balance - amount < 0
        if self.__balance >= amount:
            self.__balance = self.__balance - amount

            return True
        else:
            return False


    def get_balance(self): # not giving direct access to variable balance so␣
↪giving method to just see it
        return self.__balance # as creator i can access it because i know name,␣
↪but user cant
```

```
                                    # but through object no one can directly access␣
↪it, only by method which i made
```

[82]:
```
obj_bank_account = bank_account(1000)
```

[83]:
```
obj_bank_account.get_balance()
```

[83]:
```
1000
```

[84]:
```
obj_bank_account.deposit(6000)
```

[85]:
```
obj_bank_account.get_balance()
```

[85]:
```
7000
```

[86]:
```
obj_bank_account.withdraw(10000)
```

[86]:
```
False
```

[87]:
```
obj_bank_account.withdraw(2000)
```

[87]:
```
True
```

[88]:
```
obj_bank_account.get_balance()
```

[88]:
```
5000
```

[ ]:

[ ]:

3 Inheritance

[ ]:
```
class parent:

    def test_parent(self):
        print("this is parent class")
```

[ ]:
```
clas
```

[ ]:

[ ]:

[1]:
```
class class1:
    def test1(self):
        print("this is class1")


class class2:
    def test2(self):
        print("this is class2")
```

[3]:
```
class class3(class1, class2):
    def test3(self):
        pass
```

[4]:
```
obj_class3 = class3()
```

[5]:
```
obj_class3.test2()
```

```
this is class2
```

[102]:
```
obj_class3.test1()
```

```
this is class1
```

[94]:
```
obj_class2 = class2()
```

[ ]:
```
obj_class2.test2
```

[ ]:

[ ]:

[ ]:

[ ]:

4 Abstract

[10]:
```
import abc

class pwskills:

    @abc.abstractmethod
    def student_details(self):
        pass
```

```
    @abc.abstractmethod
    def student_assignment(self):
        pass

    @abc.abstractmethod
    def student_marks(self):
        pass
```

[11]:
```
class data_science(pwskills):

    def student_details(self):
        return "this return student details of Data Science Master"

    def student_assignment(self):
        return "this will return student assignment of Data Science Master"
```

[12]:
```
class web_dev(pwskills):

    def student_details(self):
        return "this will return student details of web development"

    def student_marks(self):
        return "this will return marks of web development class"
```

[13]:
```
obj_ds = data_science()
```

[15]:
```
obj_ds.student_details()
```

[15]:
```
'this return student details of Data Science Master'
```

[16]:
```
wdev = web_dev()
```

[17]:
```
wdev.student_marks()
```

[17]:
```
'this will return marks of web development class'
```

[ ]:
```

```

day10_10_feb_oops_concept

1 decorators

A decorator is a design pattern in Python that allows a user to add new functionality to an existing object without modifying its structure. Decorators are usually called before the definition of a function you want to decorate.

2 static method

Static methods in Python are extremely similar to python class level methods, the difference being that a static method is bound to a class rather than the objects for that class. This means that a static method can be called without an object for that class. # class method A class method is a method that is bound to the class and not the object of the class. They have the access to the state of the class as it takes a class parameter that points to the class and not the object instance. It can modify a class state that would apply across all the instances of the class.

3 special method (magic/dunder) methods

Python dunder methods are the special predefined methods having two prefixes and two suffix underscores in the method name. … Python dunder methods can be easily understood by visualizing a contract between your implementation and the Python interpreter.

example : **inti** (_ _ inti _ _), **new**, **add**, **len** etc

4 property decorators - setter, getter and deleter

A python decorator is a function that accepts another function as an argument. A decorator's point is to enhance the function passed to it. It returns the modified function, which can then be called normally.

[ ]:

day11_11_feb_files

1 1. open and write operation in file

[1]:
```
f = open("day11_11_feb_files.txt" , mode= 'w') #created variable with file name␣
 ↪and write mode

# or : f = open("day11_11_feb_files.txt", "w")
```

[2]:
```
pwd #present working directory
```

[2]:
```
'/home/jovyan/work/day11_11_feb_files'
```

[3]:
```
ls # gives list of file present in current directory
```

```
day11_11_feb_files.ipynb  day11_11_feb_files.txt
```

[4]:
```
f.write("this my first file created by using python default functions")



# only by write() it doesn't write anything on file, it needs a close()␣
 ↪function then it'll print string
# which we have passed with write function
```

[4]:
```
60
```

[5]:
```
f.close()


# unless and untill you are not calling close() it wont write anything
```

[6]:
```

```

```
f.write("Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse␣
 ↪dapibus fringilla viverra. Vestibulum facilisis sit amet diam et varius.␣
 ↪Duis felis ipsum, ultricies non volutpat eget, vehicula non justo. Phasellus␣
 ↪mattis posuere vestibulum. Morbi fermentum est ac nisl tincidunt lobortis.␣
 ↪Donec pharetra velit quis odio dignissim volutpat. Praesent et bibendum enim.
 ↪ Duis sed odio eu nunc ornare volutpat nec ut nulla. Sed mattis turpis nec␣
 ↪elit fringilla porttitor. Vestibulum euismod sagittis tempus. Aenean sed␣
 ↪egestas ipsum. Proin eleifend magna vel rutrum accumsan. Donec hendrerit␣
 ↪fermentum risus ut pretium. Maecenas blandit leo a metus porttitor rutrum␣
 ↪Aliquam dignissim, sapien at luctus ornare, augue quam ultricies metus, non␣
 ↪venenatis diam sapien eu justo. Nulla sit amet tortor at justo finibus␣
 ↪suscipit ac nec libero. Aenean dignissim magna neque, ac mollis dui␣
 ↪facilisis id. Praesent auctor malesuada lorem, vel imperdiet orci consequat␣
 ↪quis. Suspendisse vestibulum, lacus id volutpat blandit, nisl ligula aliquam␣
 ↪leo, et egestas purus augue posuere velit. Donec gravida ipsum sed dolor␣
 ↪lobortis faucibus et sed turpis. Nunc vulputate tincidunt mauris quis ornare.
 ↪ Proin condimentum vulputate massa nec consequat. Vestibulum ante ipsum␣
 ↪primis in faucibus orci luctus et ultrices posuere cubilia curae; Vestibulum␣
 ↪vitae egestas ligula.")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[6], line 1
----> 1␣
  ↪f.write("Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse dapibus fring

ValueError: I/O operation on closed file.
```

[ ]:
```
f.close()
```

[7]:
```
f.write("dkfdlkfh sjlkfjelkj")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[7], line 1
----> 1 f.write("dkfdlkfh sjlkfjelkj")

ValueError: I/O operation on closed file.
```

1.0.1 it wont work because file is already close, so if we want to edit or do something on file we have to open it again

[8]:
```
f = open("day11_11_feb_files.txt", "w")
```

[9]:
```
f.write("Lorem ipsum kdlfjd jsl fjdlkfj jldf")


# again writing in same file wont append/ add text or content to file, but␣
 ↪it'll truncate the content and
# write the passed string
```

[9]: 35

[10]:
```
f.close()
```

[11]:
```
f = open("day11_11_feb_files.txt", 'a')
# "a" is append mode it'll let us write after the end of saved file
```

[12]:
```
f.write("\n appended string using 'a' mode to append after the end of last␣
 ↪saved content")
```

[12]: 78

[13]:
```
f.close()
```

1.0.2 read operation a file,

[14]:
```
data = open("day11_11_feb_files.txt", "r")
# 'r' mode is to open file in read mode
```

[15]:
```
#print(list(data))
data.read()
```

[15]: "Lorem ipsum kdlfjd jsl fjdlkfj jldf\n appended string using 'a' mode to append after the end of last saved content"

[16]:
```
data.read()
# again read() wont work, won't get any information, because while read file in␣
 ↪pervious read() CURSOR
# started form start of letter and it CURSOR has already reached end of line.
# so it won't print anything, (to print again we have to make cursor reach␣
 ↪start of line or where ever place we want it
```

[16]: ''

1.0.3 reset cursor to print again seek()

[17]:
```
data.seek(0) # reset the cursor at the starting of line
```

[17]: 0

[18]:
```
data.read()
```

[18]:
```
"Lorem ipsum kdlfjd jsl fjdlkfj jldf\n appended string using 'a' mode to append
after the end of last saved content"
```

[19]:
```
data.read() # again read won't work because cursor has again at the end of file

# so have to again use seek() to read file again
```

[19]:
```
''
```

[20]:
```
data.seek(20) # set cursor at 20th position, for reading the file in data␣
↪variable

# we can set cursor/pointer to any place from where we want to read out
```

[20]:
```
20
```

[21]:
```
data.read()
```

[21]:
```
"sl fjdlkfj jldf\n appended string using 'a' mode to append after the end of
last saved content"
```

1.0.4 without using file.read(), can iterate data through for loop

[22]:
```
data1 = open("day11_11_feb_files.txt", 'r')
```

[23]:
```
for i in data1:
    print(i)
```

```
Lorem ipsum kdlfjd jsl fjdlkfj jldf

 appended string using 'a' mode to append after the end of last saved content
```

2 get file size

[24]:
```
import os
```

[25]:
```
os.path.getsize("day11_11_feb_files.txt") # print file size in bytes
```

[25]:
```
113
```

3 copy file

[26]:
```
import shutil
```

[27]:
```
shutil.copy("day11_11_feb_files.txt", "new_day11_11_feb_files.txt")
```

[27]:
```
'new_day11_11_feb_files.txt'
```

4 delete file

[28]:
```
os.remove("new_day11_11_feb_files.txt")
```

[ ]:
```

```

5 different way to open file

[1]:
```
with open("day11_11_feb_files.txt","r") as data2: # context manager 'with'␣
 ↪keyword
    print(data2.read())
```

```
Lorem ipsum kdlfjd jsl fjdlkfj jldf
 appended string using 'a' mode to append after the end of last saved content
```

6 rename a file

[30]:
```
os.rename("day11_11_feb_files.txt", "new_day11_11_feb_files.txt")
```

[31]:
```
os.rename("new_day11_11_feb_files.txt", "day11_11_feb_files.txt") # revert it␣
 ↪to privious file name
```

7

8 2. read and write with different different file type/system

[1]:
```
data = {
    'name': 'resheph',
    "mail_id" : "res@gmail.com",
    "phone_number" : 34534656,
    "subject" :["data science", "big data", "data analytics"]
}

# dictionary object or data(key:value pair) in industry we can this type of␣
 ↪file JSON file
```

```
# JSON
```

9 JSON file format/system (.json files)

[2]:
```
import json
```

[3]:
```
with open("testttt_json_file", "r") as f:
    print(f) # file doesn't exist so in read mode it'll give error
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[3], line 1
----> 1 with open("testttt_json_file", "r") as f:
      2     print(f) # file doesn't exist so in read mode it'll give error

File /opt/conda/lib/python3.10/site-packages/IPython/core/interactiveshell.py:
 ↪282, in _modified_open(file, *args, **kwargs)
    275 if file in {0, 1, 2}:
    276     raise ValueError(
    277         f"IPython won't let you open fd={file} by default "
    278         "as it is likely to crash IPython. If you know what you are␣
 ↪doing, "
    279         "you can use builtins' open."
    280     )
--> 282 return io_open(file, *args, **kwargs)

FileNotFoundError: [Errno 2] No such file or directory: 'testttt_json_file'
```

10

11

[10]:
```
with open("test_json_file", "w") as f2: # file is not json file because .json␣
 ↪extension is missing
    json.dump(data, f2)     #dump data in given file name
```

[11]:
```
print(f2)
```

```
<_io.TextIOWrapper name='test_json_file' mode='w' encoding='UTF-8'>
```

[12]:
```
for i in f2:
    print(i)
```

```
---------------------------------------------------------------------------
```

```
ValueError                                Traceback (most recent call last)
Cell In[12], line 1
----> 1 for i in f2:
      2     print(i)

ValueError: I/O operation on closed file.
```

12

#

[14]:
```
with open("test_json_file1.json", "w") as f3:
    json.dump(data, f3)
```

[16]:
```
for i in f3:    # won't be able to read because file is close, we had opened it␣
 ↪with 'WITH' keyword
    print(i)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[16], line 1
----> 1 for i in f3:    # won't be able to read because file is close, we had␣
 ↪opened it with 'WITH' keyword
      2     print(i)

ValueError: I/O operation on closed file.
```

13

[17]:
```
with open("test_json_file", "r") as f3:
    data2 = json.load(f3) # storing opened json file in dat2 variable
```

[18]:
```
data2
```

[18]:
```
{'name': 'resheph',
 'mail_id': 'res@gmail.com',
 'phone_number': 34534656,
 'subject': ['data science', 'big data', 'data analytics']}
```

[19]:
```
data2["subject"][1]
```

[19]:
```
'big data'
```

14

[ ]:

15 comma seperated data/value (.CSV files) /or/ lists inside a list

[20]:
```
data = [["name", "mail_id", "number"],
        ["resheph", "reeh@gmail.com", 34643],
         ["mohan", "mohan@gmail.com", 487458775],
         ["sohan", "sohan@gmail.com", 894874]
        ]
```

[21]:
```
import csv
```

[22]:
```
with open("test_csv_file.csv", "w") as f4:
    w = csv.writer(f4)
    for i in data:  # writing data row wise
        w.writerow(i)
```

[23]:
```
with open("test_csv_file.csv", "r") as f4:
    read = csv.reader(f4) # in read variable assigning file f4 csv.reader() to␣
 ↪read from f4 file
    for i in read:
        print(i)
```

```
['name', 'mail_id', 'number']
['resheph', 'reeh@gmail.com', '34643']
['mohan', 'mohan@gmail.com', '487458775']
['sohan', 'sohan@gmail.com', '894874']
```

16

17 binary format data, audio and video open as text file will give binary file or (.bin) file

[25]:
```
with open("text_binary_file.bin", "wb") as f: #'wb' is for 'write binary' type␣
 ↪data
    f.write(b"\x01\x02\x03\x04") # passing binary data
```

[27]:
```
with open("text_binary_file.bin", "rb") as f:
    print(f.read())
```

```
b'\x01\x02\x03\x04'
```

18

19

20 3. Buffered Read and Write, (read data in chunks- user defined size)

[28]:
```
import io
```

[29]:
```
with open("test_buffered_file.txt", 'w') as f:
    file = io.BufferedWriter("djfdj dkfjwoefnnd kjjf sdkfiefjhdf \n ")
    file = io.BufferedWriter("this is buffered type data read and write")
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[29], line 2
      1 with open("test_buffered_file.txt", 'w') as f:
----> 2     file = io.BufferedWriter("djfdj dkfjwoefnnd kjjf sdkfiefjhdf \n ")
      3     file = io.BufferedWriter("this is buffered type data read and write")

AttributeError: 'str' object has no attribute 'writable'
```

[30]:
```
with open("test_buffered_file.txt", 'w') as f:
    file = io.BufferedWriter(f)
    file = f.write("this is buffered type data read and write")
    file = f.write("\nthis is my second line in buffered read and write file␣
 ↪example")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[30], line 4
      2 file = io.BufferedWriter(f)
      3 file = f.write("this is buffered type data read and write")
----> 4 file =␣
 ↪f.write("\nthis is my second line in buffered read and write file example")

ValueError: I/O operation on closed file.
```

[34]:
```
with open("test_buffered_file.txt", 'wb') as f: # 'wb' write buffer mode
    file = io.BufferedWriter(f)
    file = f.write("this is buffered type data read and write")
    file = f.write("\nthis is my second line in buffered read and write file␣
 ↪example")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[34], line 3
      1 with open("test_buffered_file.txt", 'wb') as f: # 'wb' write buffer mod
      2     file = io.BufferedWriter(f)
----> 3     file = f.write("this is buffered type data read and write")
      4     file = f.write("\nthis is my second line in buffered read and write
 ↪file example")

TypeError: a bytes-like object is required, not 'str'
```

[35]:
```
with open("test_buffered_file.txt", 'wb') as f: # 'wb' write buffer mode
    file = io.BufferedWriter(f)
    file = f.write(b"this is buffered type data read and write") # b"␣
 ↪(byte-like data) needed for buffered Read write file
    file = f.write(b"\nthis is my second line in buffered read and write file␣
 ↪example")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[35], line 4
      2 file = io.BufferedWriter(f)
      3 file = f.write(b"this is buffered type data read and write") # b"␣
 ↪(byte-like data) needed for buffered Read write file
----> 4 file =␣
 ↪f.write(b"\nthis is my second line in buffered read and write file example")

ValueError: write to closed file
```

[41]:
```
with open("test_buffered_file.txt", 'wb') as f: # 'wb' write buffer mode
    file = io.BufferedWriter(f)
    file = f.write(b"this is buffered type data read and write") #file = f.
 ↪write() is wrong
    # b" (byte-like data) needed for buffered Read write file
    file = f.write(b"\nthis is my second line in buffered read and write file␣
 ↪example")
    file.flush() # flush() is needed to close file
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[41], line 5
      3 file = f.write(b"this is buffered type data read and write") #file = f.
 ↪write() is wrong
      4 # b" (byte-like data) needed for buffered Read write file
```

```
----> 5 file =␣
 ↪f.write(b"\nthis is my second line in buffered read and write file example")
      6 file.flush() # flush() is needed to close file

ValueError: write to closed file
```

[42]:
```
with open("test_buffered_file.txt", 'wb') as f: # 'wb' write buffer mode
    file = io.BufferedWriter(f)
    file.write(b"this is buffered type data read and write") #file = f.write()␣
 ↪is wrong
    # b" (byte-like data) needed for buffered Read write file
    file.write(b"\nthis is my second line in buffered read and write file␣
 ↪example")
    file.flush() # flush() is needed to close file
```

[47]:
```
with open("test_buffered_file.txt", "rb") as f: # 'rb' read buffered/binary␣
 ↪(dont know)
    file = io.BufferedReader(f)
    #print(file.read())
    data = file.read(25) # it'll only read data of giving buffer byte size
    print(data)
```

```
b'this is buffered type dat'
```

21

22

23 4. logging and debugger

[48]:
```
print("this print funct/statement only give output in my console or write in my␣
 ↪console")
```

```
this print funct/statement only give output in my console or write in my console
```

23.0.1 in general if we perform some operation (read/write) with program, if there is some error or some bug

23.0.2 if I,m just to print it or get output on console is impossible

23.0.3 because print statement only print inside console

23.0.4 so we need some information to get rid of it, there comes logging concept

23.0.5 a permanent space where we store each and every log of program, to keep every details of program to check if some thing happens

[1]:
```
import logging
```

[2]:
```
logging.basicConfig(filename = "log_file_1", level = logging.info)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[2], line 1
----> 1 logging.basicConfig(filename = "log_file_1", level = logging.info)

File /opt/conda/lib/python3.10/logging/__init__.py:2059, in basicConfig(**kwarg
   2057 level = kwargs.pop("level", None)
   2058 if level is not None:
-> 2059     root.setLevel(level)
   2060 if kwargs:
   2061     keys = ', '.join(kwargs.keys())

File /opt/conda/lib/python3.10/logging/__init__.py:1452, in Logger.
 ↪setLevel(self, level)
   1448 def setLevel(self, level):
   1449     """
   1450     Set the logging level of this logger.  level must be an int or a st
   1451     """
-> 1452     self.level = _checkLevel(level)
   1453     self.manager._clear_cache()

File /opt/conda/lib/python3.10/logging/__init__.py:201, in _checkLevel(level)
    199     rv = _nameToLevel[level]
    200 else:
--> 201     raise TypeError("Level not an integer or a valid string: %r"
    202                     % (level,))
    203 return rv

TypeError: Level not an integer or a valid string: <function info at␣
 ↪0x7ff69cfff9a0>
```

[4]:
```
logging.basicConfig(filename = "log_file_1.log", level = logging.info)
# it won't work because 'info' is in lowercase, even though it'll create file␣
 ↪but won't be able
# to write anything
```

23.1 create program logging file

[2]:
```
logging.basicConfig(filename = "log_file_1.log", level = logging.INFO)
```

[5]:
```
logging.info("this is my logging info/ or line of execution")
```

[13]:
```
import os
#os.remove("log_file_1")
# to remove wrong log file, generated with wrong method
```

[10]:
```
pwd
```

[10]:
```
'/home/jovyan/work/day11_11_feb_files'
```

[13]:
```
logging.warning("this is warning log")
```

[7]:
```
logging.critical("this is critical log")
```

[12]:
```
logging.error("this is error log")
```

[11]:
```
logging.exception("this is exception log of project")
#it was tagged as error and printed extra none type
```

24 all logging type will also give label

#heirarchy of logging that logging follow >* NOSET (its not available in new python version, has been removed) >* 1. DEBUG >* 2. INFO >* 3. WARNING >* 4. ERROR >* 5. CRITICAL

so when we declare in logging.basicConfig level is DEBUG, INFO, WARNING, ERROR or CRITICAL

it,ll follow that heirarchy and we cannot print above level of mentioned level

if level = WARNING then we won't be able to print any INFO and DEBUG label/level log in log file

24.1 stop logging file to log

[16]:
```
logging.shutdown()
```

[19]:
```
logging.INFO("this is info label logging but it won't be able to print because␣
 ↪logging file is stopped"
```

```
            " logging.shutdown()")  # wont work because of capital INFO
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[19], line 1
----> 1␣
 ↪logging.INFO("this is info label logging but it won't be able to print becaus : logging fil

TypeError: 'int' object is not callable
```

[20]:
```
logging.info("this is info label logging but it won't be able to print because␣
 ↪logging file is stopped, logging.shutdown()")
```

24.2 logging log data with time and messages

[1]:
```
import logging
logging.basicConfig(filename=log_file_2, level= 'DEBUG', format = "%(asctime)s␣
 ↪%(message)s")

# logging with executed time and messages, from DEBUG level
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 2
      1 import logging
----> 2 logging.basicConfig(filename=log_file_2, level= 'DEBUG', format =␣
 ↪"%(asctime)s %(message)s")
      4 # logging with executed time and messages, from DEBUG level

NameError: name 'log_file_2' is not defined
```

[2]:
```
logging.basicConfig(filename="log_file_2.log", level= DEBUG, format =␣
 ↪"%(asctime)s %(message)s")

# logging with executed time and messages, from DEBUG level
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[2], line 1
----> 1 logging.basicConfig(filename="log_file_2.log", level= DEBUG, format =␣
 ↪"%(asctime)s %(message)s")
      3 # logging with executed time and messages, from DEBUG level

NameError: name 'DEBUG' is not defined
```

[6]:
```
logging.basicConfig(filename="log_file_2.log", level= logging.DEBUG, format =␣
 ↪"%(asctime)s %(message)s")

# logging with execution timeStamp and messages, from DEBUG level
```

[7]:
```
logging. info("this is my logging info and timestamp also")
```

[8]:
```
logging.debug("this is my debug log")
```

[9]:
```
logging.warning("this is my warning")
```

[10]:
```
logging.shutdown()
```

[4]:
```
import logging
logging.basicConfig(filename="log_file_3.log", level = logging.DEBUG, format =␣
 ↪'%(asctime)s %(name)s %(level)s %(message)s')
# will give error at printing time because %(level)s, its %(levelname)s
```

[1]:
```
import os
os.remove("log_file_3.log")
```

[2]:
```
import logging
logging.basicConfig(filename="log_file_3.log", level = logging.DEBUG, format =␣
 ↪'%(asctime)s %(name)s %(levelname)s %(message)s')
```

[4]:
```
logging.info("this logging info is with, timeStamp, name(may be system name),␣
 ↪level and message")
```

[5]:
```
# trying logging in a program
lst = [1,2,3,4,[4,5,6],"resheph","guns"]
```

[14]:
```
# seperating int and str in different list
l1_int = []
l2_str = []

for i in lst:
    logging.info("this is logging for list passed {}".format(lst))
    logging.info("this is log for i variable in for loop {}".format(i))
    if type(i)==list:
        for j in i:
            logging.info("this log for list in list where j is {} and i is {}".
 ↪format(j,i))
            if type(j)== int:
                l1_int.append(j)

            #else: string.append(j)
```

```
    elif type(i) == int:
        l1_int.append(i)

    else:
        if type(i) == str:
            l2_str.append(i)

logging.info("this is my final result, l1 is {l1} and list2 is {l2}".format(l1=␣
 ↪l1_int, l2=l2_str))
```

[13]:
```
l1_int
```

[13]:
```
[1, 2, 3, 4, 4, 5, 6]
```

[11]:
```
l2_str
```

[11]:
```
['resheph', 'guns']
```

25

26

27 5. package, module and import is on python file named module_package_import in my lab

[ ]:
```

```

day12_12_feb_exception_handling_1

1 exception handling to execute code even after there is error on program

[1]:
```
a = 10
```

[2]:
```
a/0 # will give ZeroDivisionError
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In[2], line 1
----> 1 a/0

ZeroDivisionError: division by zero
```

[3]:
```
f = open("test.txt", 'r') # will give FileNotFoundError
print("this is my print") # after error next line won't be able to execute
# so we use exception handling
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[3], line 1
----> 1 f = open("test.txt", 'r')

File /opt/conda/lib/python3.10/site-packages/IPython/core/interactiveshell.py:
 ↪282, in _modified_open(file, *args, **kwargs)
    275 if file in {0, 1, 2}:
    276     raise ValueError(
    277         f"IPython won't let you open fd={file} by default "
    278         "as it is likely to crash IPython. If you know what you are␣
 ↪doing, "
    279         "you can use builtins' open."
    280     )
--> 282 return io_open(file, *args, **kwargs)

FileNotFoundError: [Errno 2] No such file or directory: 'test.txt'
```

2

3

4 1. try and except

[6]:
```
try:
    f = open("test.txt", 'r')
except Exception as e: #printing exception if error in try block
    print("there is some issue with my code", e)
print("this is my print") # able to execute next line of code even after error␣
 ↪or exception
```

```
there is some issue with my code [Errno 2] No such file or directory: 'test.txt'
this is my print
```

4.1 can have n number of nested try and except

[7]:
```
try:
    f = open("test.txt", 'w')
    f.write("this is my message")
    #f.close() # can write here
except Exception as e: #printing exception if error in try block
    print("there is some issue with my code", e)
else:
    f.close() #this line will only execute only when try block is successfully␣
 ↪executed
    # to conclude
```

[9]:
```
try:
    f = open("test123.txt", 'r')
    f.write("this is my message")

except Exception as e:
    print("there is some issue with my code", e)
else:
    f.close() #this line won't execute because there's a error/exception
    # so else block won't be able to execute.
```

```
there is some issue with my code [Errno 2] No such file or directory:
'test123.txt'
```

[11]:
```
try:
    f = open("test123.txt", 'r')
    f.write("this is my message")
finally:
```

```
    print("this will always exexute, even though try block exception is not␣
 ↪handled")
    #finally block try to execute itself all the time doesn't matter what␣
 ↪happen to code
    # like write or keep important code in finally block, ex: f.close() to␣
 ↪close file
```

```
this will always exexute, even though try block exception is not handled
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[11], line 2
      1 try:
----> 2     f = open("test123.txt", 'r')
      3     f.write("this is my message")
      4 finally:

File /opt/conda/lib/python3.10/site-packages/IPython/core/interactiveshell.py:
 ↪282, in _modified_open(file, *args, **kwargs)
    275 if file in {0, 1, 2}:
    276     raise ValueError(
    277         f"IPython won't let you open fd={file} by default "
    278         "as it is likely to crash IPython. If you know what you are␣
 ↪doing, "
    279         "you can use builtins' open."
    280     )
--> 282 return io_open(file, *args, **kwargs)

FileNotFoundError: [Errno 2] No such file or directory: 'test123.txt'
```

[12]:
```
try:
    f = open("test123.txt",'r')
    f.write("this is my msg")
except Exception as e:
    print("there is some issue with my code", e)
else :
    f.close()
    print("this block will execute once try will execute itself without an␣
 ↪exception")
finally:
    print("i will execute always")
```

```
there is some issue with my code [Errno 2] No such file or directory:
'test123.txt'
i will execute always
```

[13]:
```
try:
    f = open("test.txt",'w')
    f.write("this is my msg to check except, else and finally")
except Exception as e:
    print("there is some issue with my code", e)
else :
    f.close()
    print("this block will execute once try will execute itself without an␣
 ↪exception")
finally:
    print("i will execute always")
```

```
this block will execute once try will execute itself without an exception
i will execute always
```

5

6

7 2. Custom Exception Handling

[14]:
```
# if there is no exception according to system but with regards to user
```

[16]:
```
age = int(input("enter your age"))
# system will take integer as input even (-) negative number
# thats an error for user but system won't understand
```

```
enter your age -3234
```

[18]:
```
# creating own exception
class validateage(Exception) : #(inheritence)inheriting Exception class which␣
 ↪is already present in python
    # inheriting Exception class in over own custom class
    def __init__(self,msg):
        self.msg = msg
```

[20]:
```
def validate_age(age):
    if age < 0:
        raise validateage("age should not be lesser than zero")# raise reserved␣
 ↪keyword invoke/call custom class
        # raise is call our custom exception class that we have created
    elif age > 200:
        raise validateage("age is too high")

    else :
        print("age is valid")
```

[24]:
```
try:
    age = int(input("enter your age"))
    validate_age(age) # when it'll raise our own exception class
except validateage as e: # accessing own custom exception class
    print(e)
```

```
enter your age 200
```

```
age is valid
```

[ ]:

day13_13_feb_exception_handling_2

1 1. List of General use Exceptions

[3]:
```
a = 10
10/0 # 10/0 this is ZeroDivisionError, because dividing by zero
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In[3], line 2
      1 a = 10
----> 2 10/0 # 10/0 this is ZeroDivisionError, because dividing by zero

ZeroDivisionError: division by zero
```

[4]:
```
try:
    a = 10
    a/0
except ZeroDivisionError as e:
    print(e)
```

```
division by zero
```

[6]:
```
try:
    int("resheph")
except (ValueError, TypeError) as e:
    print(e)
```

```
invalid literal for int() with base 10: 'resheph'
```

[7]:
```
# when we don't know specific Error then
try:
    int("resheph")
except: # when we dont know specific error is going to call because of try block
    print("this will catch an error")
    # in this situation it'll call super class and then it will be able to␣
 ↪except/catch error
    #this is not good practice, have to mention type of error in try and except␣
 ↪block
```

this will catch an error

[8]:
```
try:
    import resheph
except ImportError as e:# this will be able to handle import error
    print(e)
```

No module named 'resheph'

[9]:
```
try:
    d = {1: [3,4,5,6],"key":"resheph"}
    d["key10"] # 'key10' doesn't exist so throw error
except KeyError as e : # will able to handle key type error
    print(e)
```

'key10'

[11]:
```
try :
    "resheph".test() # there is no function we created so throw error
    # string object doesn't has test() method
except AttributeError as e:
    print(e)
```

'str' object has no attribute 'test'

[12]:
```
try :
    ls = [1,2,3,3]
    ls[10] # there's no element in 10 place so will throw error
except IndexError as e:
    print(e)
```

list index out of range

[13]:
```
try:
    123 + "resheph" # can't concatinate diffn type of object
except TypeError as e:
    print(e)
```

unsupported operand type(s) for +: 'int' and 'str'

[15]:
```
try:
    with open("test123.txt", 'r') as f: # opening a file in read mode which␣
 ↪doesn't exist
        f.read()
except FileNotFoundError as e:
    print("this is my file not found type error ",e)
```

this is my file not found type error [Errno 2] No such file or directory:
'test123.txt'

[17]:
```
try:
    with open("test123.txt", 'r') as f: # opening a file in read mode which␣
 ↪doesn't exist
        f.read()
except Exception as e: # directly call super class in first place
    print(e)
except FileNotFoundError as e: # this except won't be execute because error is␣
 ↪handled by Exception super class
    print("this is my file not found type error",e)
```

```
[Errno 2] No such file or directory: 'test123.txt'
```

1.0.1 - it's not good to write super class in a first place

1.0.2 - first except should be specific exception

1.0.3 - then next possible error

1.0.4 - and at last if not sure that all written exception won't be able to handle then write

1.0.5 - at last Exception super class

2

[19]:
```
def test(file):
    try:
        with open(file, 'r') as f: # opening a file in read mode which doesn't␣
 ↪exist
            f.read()
    except Exception as e: # directly call super class in first place
        print(e)
    except FileNotFoundError as e: # this except won't be execute because error␣
 ↪is handled by Exception super class
        print("this is my file not found type error",e)
```

2.0.1 even inside function we can use try, except block to handle Exception

3

4

5 - Best practices to do in coding, using logging and try,except Exception handling

6 - in production level project/ production level code

[21]:
```
# don't use super class 'Exception' in try and except block to handle error/
 ↪exception

try:
    10/0
except Exception as e:
    print(e)
```

```
division by zero
```

6.0.1 always use specific exception

[23]:
```
try:
    10/0
except ZeroDivisionError as e:
    print(e)
```

```
division by zero
```

6.0.2 always give meaningful message, (easily understandable at debugging)

[25]:
```
#print always a valid msg
# when ever handling error use proper understandable message
# people would understand at the time of debugging
# give meaningful message
try:
    10/0
except ZeroDivisionError as e:
    print("this is my zero division error i am handling",e)
```

```
this is my zero division error i am handling division by zero
```

7 always try to log

[28]:
```
# always try to log
import logging
logging.basicConfig(filename = "day13_13_feb_best_practice.log", level =␣
 ↪logging.ERROR)
# when ever using logging module don't use print()
#logging.basicConfig(filename = "error.log", level = logging.ERROR)
# now able to get all the Error based log in error.log file
try:
    10/0
except ZeroDivisionError as e:
    logging.error("this is my zero division error i am handling {}".format(e))
# it will log /print in permanent file (error.log file) not inside console␣
 ↪because it'll be gone
```

8 always avoid to write a multiple exception handling

[29]:
```
try:
    10/0
except FileNotFoundError as e:
    logging.error("this is my file not found {}".format(e))
except AttributeError as e:
    logging.error("this is my attribute error {}".format(e))
except ZeroDivisionError as e:
    logging.error("this is my zero division error i am handling {} ".format(e))


#multiple handling, all starting exceptions are unnecessary because they won't␣
 ↪handle
#zero division error, so try to avoid multiple exception handling
```

9

10 always prepare a proper documentation

- avoid creating anything which may give a problem to a future developer who's going to check code or who'll try to do modification
- proper documentation with the proper valid meaningful informations of/about each and every errors is very curcial and important
- proper documentations of the entire code, entire model, entire package, entire file or entire classes and objects and methods we have/used-to created
- proper documentation for each and every thing

11

12

13 cleanup all the resources

[ ]:
```
try:
    with open("test2.txt",'w') as f:
        f.write("this is msg to file")
except FileNotFoundError as e:
    logging.error("this is my file not found {}".format(e)
finally :
    f.close() # cleaning file, i.e. closing the file

# example: database connectivity sould be close, endup utalising unnecessary␣
 ↪bandwith of database connection;
# doing network io operation, suppose to clean up the entire network io,other␣
 ↪wise it will end up occuping
# the bandwith, means OVER utalization of the resources,
# so never OVER or UNDER utalize the resources.
```

day14_14_feb_multi_threading

1 multithreading

[20]:
```
import threading
```

[3]:
```
def test(id):
    print("this is my test id %d" % id)
```

[22]:
```
test(10)
```

```
this is my test id 10
```

[23]:
```
test(1) # calling function synchronously, one by one
```

```
this is my test id 1
```

[24]:
```
test(3)
```

```
this is my test id 3
```

[25]:
```
# using threading to call concurrently
thred = threading.Thread(target= test, args = [10,1,3])
```

[26]:
```
print(thred)
```

```
<Thread(Thread-10 (test), initial)>
```

[27]:
```
#for t in thred:
thred.start()
```

```
Exception in thread Thread-10 (test):
Traceback (most recent call last):
  File "/opt/conda/lib/python3.10/threading.py", line 1016, in _bootstrap_inner
    self.run()
  File "/opt/conda/lib/python3.10/threading.py", line 953, in run
    self._target(*self._args, **self._kwargs)
TypeError: test() takes 1 positional argument but 3 were given
```

[28]:
```
thred = threading.Thread(target = test, args = (10))
```

[29]:
```
thred
```

[29]:
```
<Thread(Thread-11 (test), initial)>
```

[30]:
```
thred.start()
```

```
Exception in thread Thread-11 (test):
Traceback (most recent call last):
  File "/opt/conda/lib/python3.10/threading.py", line 1016, in _bootstrap_inner
    self.run()
  File "/opt/conda/lib/python3.10/threading.py", line 953, in run
    self._target(*self._args, **self._kwargs)
TypeError: __main__.test() argument after * must be an iterable, not int
```

2

[ ]:
```
thred = [threading.Thread(target = test, args = (i,)) for i in [10,1,3]]
```

[ ]:
```
thred
```

[32]:
```
for t in thred:
    t.start()
# with threading this 3 program are executing in a single proccesor
```

```
this is my test id 10
this is my test id 1
this is my test id 3
```

[16]:
```
import threading
thred = [threading.Thread(target = test, args = (10,1,3,))]
```

[17]:
```
thred.start()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[17], line 1
----> 1 thred.start()

AttributeError: 'list' object has no attribute 'start'
```

[18]:
```
for t in thred:
    t.start()
```

```
Exception in thread Thread-9 (test):
Traceback (most recent call last):
  File "/opt/conda/lib/python3.10/threading.py", line 1016, in _bootstrap_inner
```

```
    self.run()
  File "/opt/conda/lib/python3.10/threading.py", line 953, in run
    self._target(*self._args, **self._kwargs)
TypeError: test() takes 1 positional argument but 3 were given
```

3

[39]:
```
import os
os.getcwd()
```

[39]:
```
'/home/jovyan/work/day14_14_feb_multi_threading'
```

[41]:
```
import urllib.request
```

[44]:
```
#download data form links, function
def file_download(url,file_name):
    ''' take 2 argument a url and file name'''
    urllib.request.urlretrieve(url, file_name)
```

[45]:
```
file_download('https://raw.githubusercontent.com/itsfoss/text-files/master/
 ↪agatha.txt',"test1.txt")
```

[63]:
```
url_list = ['https://raw.githubusercontent.com/itsfoss/text-files/master/agatha.
 ↪txt', 'https://raw.githubusercontent.com/itsfoss/text-files/master/sherlock.
 ↪txt' ,'https://raw.githubusercontent.com/itsfoss/text-files/master/
 ↪sample_log_file.txt']
```

[64]:
```
url_list
```

[64]:
```
['https://raw.githubusercontent.com/itsfoss/text-files/master/agatha.txt',
 'https://raw.githubusercontent.com/itsfoss/text-files/master/sherlock.txt',
 'https://raw.githubusercontent.com/itsfoss/text-
files/master/sample_log_file.txt']
```

[65]:
```
data_file_list = ["data1.txt", 'data2.txt', 'data3.txt']
```

[66]:
```
data_file_list
```

[66]:
```
['data1.txt', 'data2.txt', 'data3.txt']
```

[67]:
```
thread1 = [threading.Thread(target = file_download, args = (url_list[i],␣
 ↪data_file_list[i])) for i in range(len(url_list))]
```

[68]:
```
thread1
```

[68]:
```
[<Thread(Thread-18 (file_download), initial)>,
 <Thread(Thread-19 (file_download), initial)>,
```

```
<Thread(Thread-20 (file_download), initial)>]
```

[69]:
```
for t in thread1:
    t.start()
```

4

[70]:
```
import time
```

[71]:
```
def test2(x):
    for i in range(10):
        print("test2 print the value of x %d and the value of loop i %d" %(x,i))
```

[76]:
```
thread2 = [threading.Thread(target = test2, args = (i,)) for i in [100, 10, 20,5]]
print(thread2)

for t in thread2:
    t.start()
```

```
[<Thread(Thread-37 (test2), initial)>, <Thread(Thread-38 (test2), initial)>,
<Thread(Thread-39 (test2), initial)>, <Thread(Thread-40 (test2), initial)>]
test2 print the value of x 100 and the value of loop i 0
test2 print the value of x 100 and the value of loop i 1
test2 print the value of x 100 and the value of loop i 2
test2 print the value of x 100 and the value of loop i 3
test2 print the value of x 100 and the value of loop i 4
test2 print the value of x 100 and the value of loop i 5
test2 print the value of x 100 and the value of loop i 6
test2 print the value of x 100 and the value of loop i 7
test2 print the value of x 100 and the value of loop i 8
test2 print the value of x 100 and the value of loop i 9
test2 print the value of x 10 and the value of loop i 0
test2 print the value of x 10 and the value of loop i 1
test2 print the value of x 10 and the value of loop i 2
test2 print the value of x 10 and the value of loop i 3
test2 print the value of x 10 and the value of loop i 4
test2 print the value of x 10 and the value of loop i 5
test2 print the value of x 10 and the value of loop i 6
test2 print the value of x 10 and the value of loop i 7
test2 print the value of x 10 and the value of loop i 8
test2 print the value of x 10 and the value of loop i 9
test2 print the value of x 20 and the value of loop i 0
test2 print the value of x 20 and the value of loop i 1
test2 print the value of x 20 and the value of loop i 2
test2 print the value of x 20 and the value of loop i 3
```

```
test2 print the value of x 20 and the value of loop i 4
test2 print the value of x 20 and the value of loop i 5
test2 print the value of x 20 and the value of loop i 6
test2 print the value of x 20 and the value of loop i 7
test2 print the value of x 20 and the value of loop i 8
test2 print the value of x 20 and the value of loop i 9
test2 print the value of x 5 and the value of loop i 0
test2 print the value of x 5 and the value of loop i 1
test2 print the value of x 5 and the value of loop i 2
test2 print the value of x 5 and the value of loop i 3
test2 print the value of x 5 and the value of loop i 4
test2 print the value of x 5 and the value of loop i 5
test2 print the value of x 5 and the value of loop i 6
test2 print the value of x 5 and the value of loop i 7
test2 print the value of x 5 and the value of loop i 8
test2 print the value of x 5 and the value of loop i 9
```

[78]:
```
# it executed line by line, one element of list after another
# now giving time.sleep 1 second, to make a process sleep for 1 second after␣
 ↪single execution

def test2(x):
    for i in range(10):
        print("test2 print the value of x %d and the value of loop i %d" %(x,i))
        time.sleep(1)

thread2 = [threading.Thread(target = test2, args = (i,)) for i in [100, 10, 20␣
 ↪,5]]
print(thread2)

for t in thread2:
    t.start()

# so in a single core of processor, program is distributing resources to␣
 ↪complete each, when one execution
# goes to sleep then other element of the list executes
# so now program not complete test2() for loop of 100 completly but after '100'␣
 ↪value single loop executed then
# it goes to sleep at that time it will execute next on the queue element in␣
 ↪list '10'

# because of sleep it's giving opportunity to other program for execution, it's␣
 ↪sharing resources in single core
```

```
[<Thread(Thread-41 (test2), initial)>, <Thread(Thread-42 (test2), initial)>,
<Thread(Thread-43 (test2), initial)>, <Thread(Thread-44 (test2), initial)>]
test2 print the value of x 100 and the value of loop i 0
```

```
test2 print the value of x 10 and the value of loop i 0
test2 print the value of x 20 and the value of loop i 0
test2 print the value of x 5 and the value of loop i 0
test2 print the value of x 100 and the value of loop i 1
test2 print the value of x 10 and the value of loop i 1
test2 print the value of x 20 and the value of loop i 1
test2 print the value of x 5 and the value of loop i 1
test2 print the value of x 100 and the value of loop i 2
test2 print the value of x 10 and the value of loop i 2
test2 print the value of x 5 and the value of loop i 2
test2 print the value of x 20 and the value of loop i 2
test2 print the value of x 100 and the value of loop i 3
test2 print the value of x 10 and the value of loop i 3
test2 print the value of x 5 and the value of loop i 3
test2 print the value of x 20 and the value of loop i 3
test2 print the value of x 100 and the value of loop i 4
test2 print the value of x 10 and the value of loop i 4
test2 print the value of x 5 and the value of loop i 4
test2 print the value of x 20 and the value of loop i 4
test2 print the value of x 100 and the value of loop i 5
test2 print the value of x 10 and the value of loop i 5
test2 print the value of x 5 and the value of loop i 5
test2 print the value of x 20 and the value of loop i 5
test2 print the value of x 100 and the value of loop i 6
test2 print the value of x 10 and the value of loop i 6
test2 print the value of x 5 and the value of loop i 6
test2 print the value of x 20 and the value of loop i 6
test2 print the value of x 100 and the value of loop i 7
test2 print the value of x 10 and the value of loop i 7
test2 print the value of x 5 and the value of loop i 7
test2 print the value of x 20 and the value of loop i 7
test2 print the value of x 100 and the value of loop i 8
test2 print the value of x 10 and the value of loop i 8
test2 print the value of x 5 and the value of loop i 8
test2 print the value of x 20 and the value of loop i 8
test2 print the value of x 100 and the value of loop i 9
test2 print the value of x 10 and the value of loop i 9
test2 print the value of x 5 and the value of loop i 9
test2 print the value of x 20 and the value of loop i 9
```

5

[79]:
```
#threading.lock() lock the resource that no one will be able to use, single␣
 ↪thread will run at that time

shared_var = 0
lock_var = threading.Lock()
```

```
def test3(x):
    global shared_var
    with lock_var:
        shared_var = shared_var + 1
        print("value of x %d and value of shared_var %d" %(x,shared_var))
        time.sleep(1)

thread4 = [threading.Thread(target = test3, args = (i,)) for i in [1, 2, 3, 4,␣
 ↪4, 5]]
for t in thread4:
    t.start()
```

```
value of x 1 and value of shared_var 1
value of x 2 and value of shared_var 2
value of x 3 and value of shared_var 3
value of x 4 and value of shared_var 4
value of x 4 and value of shared_var 5
value of x 5 and value of shared_var 6
```

[ ]:

day14_14_feb_multi_threadingjkh

[1]:
```
import threading
```

[2]:
```
def test(id):
    print("this is my test id %d" % id)
```

[3]:
```
test(10)
```

```
this is my test id 10
```

[4]:
```
test(1) # calling function synchronously, one by one
```

```
this is my test id 1
```

[5]:
```
test(3)
```

```
this is my test id 3
```

[9]:
```
# using threading to call concurrently
thred = [threading.Thread(target = test, args = (i,))for i in [10 ,1, 3]]
```

[10]:
```
thred
```

[10]:
```
[<Thread(Thread-8 (test), initial)>,
 <Thread(Thread-9 (test), initial)>,
 <Thread(Thread-10 (test), initial)>]
```

[11]:
```
for t in thred:
    t.start()
# with threading this 3 program are executing in a single proccesor
```

```
this is my test id 10
this is my test id 1
this is my test id 3
```

[14]:
```
# download data from links
import urllib.request

def file_download(url, filename):
```

```
    urllib.request.urlretrieve(url, filename)
```

[15]:
```
file_download('https://raw.githubusercontent.com/itsfoss/text-files/master/
↪agatha.txt',"test1.txt")
```

[ ]:
```

```

day15_15_feb_memory_management_multi_processing

0.0.1 memory management

1 multiprocessing

[1]:
```
# can execute multliple pragram, using more than one core of processor,
# less latency
```

[2]:
```
import multiprocessing

def test():
    print("this is my multiprocessing program")
# if want to execute this function with different program
```

[3]:
```
test() # executing separatly
```

```
this is my multiprocessing program
```

2 multiprocessing.Process()

[4]:
```
# if want to execute with some other program
# so create/use/call main method _ _ main _ _ it is responsible for all the␣
 ↪executing everything
#inside python compiler.

# it invoke entire python compiler this is process in itself
```

[6]:
```
import multiprocessing

def test():
    print("this is my multiprocessing program")


if __name__ == "__main__": # calling main method, parent program
    m = multiprocessing.Process(target= test)
    # inside main program calling child program test()
    print("this is my main program")
    m.start() # start child process
```

```
    m.join()# wait untill child process terminates
```

```
this is my main program
this is my multiprocessing program
```

good practice to use main method as a parent

then execute child program inside it for multiprocessing

3 multiprocess.Pool()

[7]:
```
def square(n): #created an outside program
    return n**2

if __name__ == "__main__":#can execute without it but it's good/better for␣
 ↪multi processing,
    #to run main as parent and other program inside it as child

    #m = multiprocessing.Pool(processes=5)

    #Pool(), for providing pool of data inside this program, alocate 5␣
 ↪processor to execute and acumulate result and give output
    #giving pool of processor (5), and parallely execute the data we have␣
 ↪passed


    with multiprocessing.Pool(processes=5) as pool:
        out = pool.map(square, [3,4,5,6,6,7,87,8,8]) # if use with for it'll␣
 ↪create single program, so may be have nature for multi-thread
        print(out)

        #with pool all the data in list execute simultaneiously not one by one␣
 ↪order
        # parallely processed with 5=processes and after completing/
 ↪accumulating result, than gave a final output
```

```
[9, 16, 25, 36, 36, 49, 7569, 64, 64]
```

4 multiprocessing.Queue()

[8]:
```
# function to inserting a data
# funtion to extracting a data
```

[10]:
```
import multiprocessing

def producer(q): # insert/put data in a queue so it'll create a queue
    for i in ["resheph", "mohan", "pwskills", "sohan", "rohan"]:
        q.put(i) # it'll one by one put data in to a queue

def consume(q): # extract/fetch/eliminate data from queue
    while True:
        item = q.get() # it'll give item one by one
        if item is None:
            break
        print(item)

if __name__ == '__main__': # main method
    queue = multiprocessing.Queue() # creating a queue (queue object/queue␣
 ↪datastructure to put() and get() data)
    m1 = multiprocessing.Process(target = producer, args = (queue,)) #en-queue /
 ↪ producer
    #allocted m1 process in a processor
    m2 = multiprocessing.Process(target = consume, args = (queue,)) #de-queue /␣
 ↪consumer
    #allocted m2 process into another processor
    m1.start() #starting process m1
    m2.start() # starting process m2
    queue.put("xyz") # putting/inserting extra/additional element in queue, at␣
 ↪runtime
    m1.join() #terminating m1 process, releasing all the given resources
    m2.join() # terminating entire queue

#Call process.join() to tell the program that it should wait for this process␣
 ↪to complete before it continues with the rest of the code.
```

```
resheph
mohan
pwskills
sohan
rohan
xyz
Process Process-10:
Traceback (most recent call last):
  File "/opt/conda/lib/python3.10/multiprocessing/process.py", line 314, in
_bootstrap
    self.run()
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
```

```
Cell In[10], line 24
     22 queue.put("xyz") # putting/inserting extra/additional element in queue,
 ↪at runtime
     23 m1.join() #terminating m1 process, releasing all the given resources
---> 24 m2.join() # terminating entire queue

File /opt/conda/lib/python3.10/multiprocessing/process.py:149, in BaseProcess.
 ↪join(self, timeout)
    147 assert self._parent_pid == os.getpid(), 'can only join a child process'
    148 assert self._popen is not None, 'can only join a started process'
--> 149 res = self._popen.wait(timeout)
    150 if res is not None:
    151     _children.discard(self)

File /opt/conda/lib/python3.10/multiprocessing/popen_fork.py:43, in Popen.
 ↪wait(self, timeout)
     41             return None
     42     # This shouldn't block if wait() returned successfully.
---> 43     return self.poll(os.WNOHANG if timeout == 0.0 else 0)
     44 return self.returncode

File /opt/conda/lib/python3.10/multiprocessing/popen_fork.py:27, in Popen.
 ↪poll(self, flag)
     25 if self.returncode is None:
     26     try:
---> 27         pid, sts = os.waitpid(self.pid, flag)
     28     except OSError:
     29         # Child process not yet created. See #1731717
     30         # e.errno == errno.ECHILD == 10
     31         return None

KeyboardInterrupt:
```

```
  File "/opt/conda/lib/python3.10/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
  File "/tmp/ipykernel_1719/3584824666.py", line 9, in consume
    item = q.get() # it'll give item one by one
  File "/opt/conda/lib/python3.10/multiprocessing/queues.py", line 103, in get
    res = self._recv_bytes()
  File "/opt/conda/lib/python3.10/multiprocessing/connection.py", line 221, in
recv_bytes
    buf = self._recv_bytes(maxlength)
  File "/opt/conda/lib/python3.10/multiprocessing/connection.py", line 419, in
_recv_bytes
    buf = self._recv(4)
  File "/opt/conda/lib/python3.10/multiprocessing/connection.py", line 384, in
_recv
```

```
    chunk = read(handle, remaining)
KeyboardInterrupt
```

[ ]:
```
# another example of multi- processing
```

4.0.1 multiprocessing.Array()

[4]:
```
import multiprocessing

def square(index, value):
    value[index] = value[index] ** 2

if __name__ == "__main__":

    arr = multiprocessing.Array('i', [2,3,6,7,8,8,9,3,3,3])
    #GOOGLE COPIED : The 'd' and 'i' arguments used when creating num and arr␣
 ↪are typecodes of the kind used by the array module:
    # 'd' indicates a double precision float and 'i' indicates a signed integer.

    processes = [] # for join() to know and terminate after executing loop␣
 ↪start() program
    print([ar for ar in arr])
    for i in range(10):
        m = multiprocessing.Process(target = square, args = (i, arr))
        processes.append(m)
        m.start()
    for process in processes:
        process.join() # terminate processes which we started

    print(list(arr))
```

```
[2, 3, 6, 7, 8, 8, 9, 3, 3, 3]
[4, 9, 36, 49, 64, 64, 81, 9, 9, 9]
```

[ ]:
```

```

4.1 multiprocessing.Pipe()

default duplex = True, means one can do both sending and receiving message - two way communication

if duplex = False, means it's in simplex mode where only one send message - one way communication

[22]:
```
#producing a message and receiving a message
#like whatsapp, sending message it will get process then someone will receive it
#pipe will let 2-way communication/ and there is 1-way communication also
# piping will let process 2-way and 1-way communication between systems
```

[26]:
```
import multiprocessing

def sender(conn, msg): #conn is connectivity/connection of internet, to connect␣
 ↪with server
    # pipe will create/stablish whole connection between sender and receiver
    for i in msg:
        conn.send(i)
    conn.close()

def receive(conn):
    while True:
        try:
            msg = conn.recv()
        except Exception as e: #if receiver won't get msg or won't be able to␣
 ↪connect then break
            print(e)
            break  # break
        print(msg)

if __name__ == "__main__":
    msg = ["my name is resheph", "this is my msg to my employee", "I'm giving␣
 ↪bonus", "try to practice all the code"]
    parent_conn, child_conn = multiprocessing.Pipe()
    # pipe duplex=True so getting two connection, named : '1) parent connection␣
 ↪and 2) child connection for sender and receiver'
    m1 = multiprocessing.Process(target=sender, args=(child_conn, msg))
    m2 = multiprocessing.Process(target=receive, args = (parent_conn,))
    m1.start()
    m2.start()
    m1.join()
    child_conn.close()
    m2.join()
    parent_conn.close()
```

```
my name is resheph
this is my msg to my employee
I'm giving bonus
try to practice all the code
```

```
Process Process-124:
Traceback (most recent call last):
  File "/opt/conda/lib/python3.10/multiprocessing/process.py", line 314, in
_bootstrap
    self.run()
  File "/opt/conda/lib/python3.10/multiprocessing/process.py", line 108, in run
    self._target(*self._args, **self._kwargs)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[26], line 28
     26 m1.join()
     27 child_conn.close()
---> 28 m2.join()
     29 parent_conn.close()

File /opt/conda/lib/python3.10/multiprocessing/process.py:149, in BaseProcess.
 ↪join(self, timeout)
    147 assert self._parent_pid == os.getpid(), 'can only join a child process'
    148 assert self._popen is not None, 'can only join a started process'
--> 149 res = self._popen.wait(timeout)
    150 if res is not None:
    151     _children.discard(self)

File /opt/conda/lib/python3.10/multiprocessing/popen_fork.py:43, in Popen.
 ↪wait(self, timeout)
     41             return None
     42     # This shouldn't block if wait() returned successfully.
---> 43     return self.poll(os.WNOHANG if timeout == 0.0 else 0)
     44 return self.returncode

File /opt/conda/lib/python3.10/multiprocessing/popen_fork.py:27, in Popen.
 ↪poll(self, flag)
     25 if self.returncode is None:
     26     try:
---> 27         pid, sts = os.waitpid(self.pid, flag)
     28     except OSError:
     29         # Child process not yet created. See #1731717
     30         # e.errno == errno.ECHILD == 10
     31         return None

KeyboardInterrupt:
```

```
  File "/tmp/ipykernel_1719/2227431326.py", line 12, in receive
    msg = conn.recv()
  File "/opt/conda/lib/python3.10/multiprocessing/connection.py", line 255, in
recv
    buf = self._recv_bytes()
  File "/opt/conda/lib/python3.10/multiprocessing/connection.py", line 419, in
_recv_bytes
    buf = self._recv(4)
  File "/opt/conda/lib/python3.10/multiprocessing/connection.py", line 384, in
_recv
    chunk = read(handle, remaining)
```

```
KeyboardInterrupt
```

[ ]: