

```
In [29]: #PRINT STATEMENT
print('welcome to Python Program')

print('python','Programming')

print('python','program',sep='- ')

a = "Hi"
print(f'{a}, Pranav')

print("hey there, \
How are You?")

print(int(2.5))
```

```
welcome to Python Program
python Programming
python-program
Hi, Pranav
hey there, How are You?
2
```

```
In [33]: #Variables
a=2
print(a)

a="Pranav"
print(a)

a,b=3.5,"Pranav"
print(a,b)
print(type(a))
print(type(b))

A=_b = cd = 'Python'
print(cd, _b)

a,b = b,a
print(b,a)

id(a)
```

```
2
Pranav
3.5 Pranav
<class 'float'>
<class 'str'>
Python Python
3.5 Pranav
```

Out[33]: 2189045343264

```
In [5]: #Arithmetic Operators:  +  -  *  /  %  **  //(floor division)

#Modulo(Reminder): %
#Divisor is +ve -> Regular division
#Divisor is -ve -> Floor is taken(away from 0) -> reminder is also -ve

#  $r = a - (n * \text{floor}(a/n))$ 
```

```
In [6]: 8%3
```

```
Out[6]: 2
```

```
In [7]: -8 % 3
```

```
Out[7]: 1
```

```
In [8]: 8%-3
```

```
Out[8]: -1
```

```
In [9]: -8%-3
```

```
Out[9]: -2
```

```
In [16]: #Assignment operators:  =  /=  +=  *=  //=  %=  &=  |=  ^=
```

```
In [10]: #Logical:      and - or - not    ex: not(x<5 and x<10)
```

```
In [11]: #Identity:    (from same obj or not (id))    is - is not    ex: x is not y
```

```
In [12]: #Membership:  in - not in
```

```
In [13]: #BITWISE OPS:  &(AND)  -  |(OR)  -  ^(XOR)  -  ~(NOT)  -  <<(LEFT SHIFT)  -  >>(RIGHT SHIFT)
```

```
In [14]: #Modules - file(set(functions)) -> exec'ble stmts + Fn defns -> Import keyword &
```

```
In [15]: import math
print(dir(math))
math.floor(-23.6)
```

```
['_doc_', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acos h', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cos h', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

Out[15]: -24

### Functions - subProg for a specific task -

- **function definition:** A statement that creates a new function, specifying its name, parameters, and the statements it contains.
- **header:** The first line of a function definition.
- **body:** The sequence of statements inside a function definition.
- **parameter:** A name used inside a function to refer to the value passed as an argument.
- **function call:** A statement that runs a function. It consists of the function name followed by an argument list in parentheses.
- **argument:** A value provided to a function when the function is called. This value is assigned to the corresponding parameter in the function.
- **local variable:** A variable defined inside a function. A local variable can only be used inside its function.
- **return value:** The result of a function. If a function call is used as an expression, the return value is the value of the expression.
- **fruitful function:** A function that returns a value.
- **void function:** A function that always returns None.
- **None:** A special value returned by void functions.

```
In [23]: def sum(a,b):  #->a,b : parameters
s = a+b
return(s)

x=sum(2,3)  #->2,3 : arguments
x
```

Out[23]: 5

```
In [25]: # 1. PreDefined/Lib fns    2.User-defined Fns

# def fn_name(parameters):
#     """docstring"""
#     stmts(s)
```

```
In [31]: #Conditional statements
```

1.if test expression:

```
statement(s)
```

2.if test expression:

```
Body of if
```

else:

Body of else

---

3.if test expression:

Body of if

elif test expression:

Body of elif

else:

Body of else

---

4.while test\_expression:

Body of while

```
In [2]: #Range- strt 0 (def) , step 1 (def). ->range(strt,stop,step)
print(list(range(6)))
print(list(range(3,6,2)))

[0, 1, 2, 3, 4, 5]
[3, 5]
```

```
In [3]: #For Loop
# for varname in Grp(vals):
# statements
```

```
In [4]: x=10
for i in range(x):
    print("squares: ", i*i)
```

```
squares: 0
squares: 1
squares: 4
squares: 9
squares: 16
squares: 25
squares: 36
squares: 49
squares: 64
squares: 81
```

```
In [11]: #Jumping Stmts:  
#Break - continue - pass (placeholder for future implemntn)  
for i in range(2,20):  
    if (i%5==0 or i%3==0):  
        continue  
    print(i)
```

```
2  
4  
7  
8  
11  
13  
14  
16  
17  
19
```

```

In [36]: #Strings: seq of chrctrs, ' / " " / "" "" , immutable
str='Pranav'
print('str= ', str)
print('str[0]=', str[0])
print('str[-1]=', str[-1])
print('str[1:4]=', str[1:4])
print('str[1:-2]=', str[1:-2])
print(str[::-1]) #Reverse String

a="a b"
print(len(a))
print(str.upper()) #Upper/Lowercase
print(str.replace("v","w"))

b=" Python.Program "
print(b.split("."))
print(b.strip())# blank spaces
print(b.rstrip())
print(b.center(40,'!'))
print(b.count('P')) #count of occurence
print(b.find('o')) #1st occurence index
print(b.rfind('o')) #Last occurence index
print(b.casefold()) #ignore case
print(ord('p')) #ordinal
print(chr(112)) #char
print(b.capitalize())
print(b.title())

# islower isupper isdigit isspace isalnum isalpha endswith('sub') startswi

print("""Hello, "Welcome to Pranav's Python Program!" """) #OR use '\ '

```

```

str= Pranav
str[0]= P
str[-1]= v
str[1:4]= ran
str[1:-2]= ran
vanarP
3
PRANAV
Pranaw
[' Python', 'Program ']
Python.Program
Python.Program
!!!!!!!!!!!! Python.Program !!!!!!!!!!!!!
2
6
11
python.program
112

```

```
p
python.program
Python.Program
Hello, "Welcome to Pranav's Python Program!"
```

Out[36]: True

```

In [30]: #Lists: ordered coll'n of elements | mutable| a[0] | a[-1] |
a=[1,22,333,4444,"P",55555,"P"]
b=list([1,23,45.6,"Pranav","Python","Pranav"])
print(a)
print(a[-1])
print(a[0])
print(b[:-1])
print(b[2:5])

for i in b:
    print(i)

print(len(b))           #length
print(b.count("Pranav")) #count
b.append(100.34)         #append
print(b)
b.insert(2,"haha")       #insert@ specfc postn
print(b)
b.pop(4)                 #pop @ postn
print(b)
print(b.pop())           #pop - deflt remove Last
print(b)
a.remove("P")            #remove 1st occ of value
print(a)
print(b.index("Pranav"))

print(id(a))
a.reverse()              #reverses list n stores in same mem loctn
print(a)
print(id(a))

#b.sort(reverse=True)    #Sorting List

#Aliasing: Same memory location ex; a=b , a is b ... -> changes in 1 List = change
# b = a.copy ->Diff mem loctns
#Slicing -> b = a[:] #make a clone ->Cloning List

#Clear: a.clear()
#del(a[1])

a.extend(b)
print(a)

#min, sum, max

#List Comprehension: new_List = [expression for member in iterable]
x=[1,34,455,65,34,23,66,21,19,0,99]
x1=[i for i in x if i>30]
print(x1)

x2=[i for i in range(1,101)]
print(x2)

x3=[i*i for i in range(1,11)]

```



```

print(x3)

x4=[i for i in range(50,101) if i%2==0]
print(x4)

x5=[i for i in range(1,101) if i%3==0 and i%5==0]
print(x5)

#Nested List
c=[[1,2,3],[4,5,6],[7,8,9]]
print(c)
print(c[0])
print(c[2][1])

[1, 22, 333, 4444, 'P', 55555, 'P']
P
1
['Pranav', 'Python', 'Pranav', 45.6, 23, 1]
[45.6, 'Pranav', 'Python']
1
23
45.6
Pranav
Python
Pranav
6
2
[1, 23, 45.6, 'Pranav', 'Python', 'Pranav', 100.34]
[1, 23, 'haha', 45.6, 'Pranav', 'Python', 'Pranav', 100.34]
[1, 23, 'haha', 45.6, 'Python', 'Pranav', 100.34]
100.34
[1, 23, 'haha', 45.6, 'Python', 'Pranav']
[1, 22, 333, 4444, 55555, 'P']
5
1737250646984
['P', 55555, 4444, 333, 22, 1]
1737250646984
['P', 55555, 4444, 333, 22, 1, 1, 23, 'haha', 45.6, 'Python', 'Pranav']
[34, 455, 65, 34, 66, 99]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42,
43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62,
63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82,
83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
[50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 8
8, 90, 92, 94, 96, 98, 100]
[15, 30, 45, 60, 75, 90]
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[1, 2, 3]
8

```

```

In [39]: #Tuples: seq of immutable pyth objs enclosed in paranthesis() / Faster than Lists
t=(100,9.9,"Pranav","Python",8,990)
t1="abc","def","efg"
t2=10,20,30,"P","ABC"
t3=tuple((1,2,3,4))
print(t)
print(t1)
print(t2)
print(t3)

print(t[3])
print(t[2][1:4])           #Slicing
print(t1[-1])

print(t.count(990))        #count of occ
print(t.index(8))          #index of 1st occ
print(sorted(t3,reverse=True))
t4=t1*4                    #repetition
print(t4)
t5=t2+t3
print(t5)                  #Concat
print(10 not in t5)        #Membership

del t5
print(id(t))

def square(x,y):
    return x*x,y*y
dd,ee = square(10,27)
print(dd,ee)

```

```

(100, 9.9, 'Pranav', 'Python', 8, 990)
('abc', 'def', 'efg')
(10, 20, 30, 'P', 'ABC')
(1, 2, 3, 4)
Python
ran
efg
1
4
[4, 3, 2, 1]
('abc', 'def', 'efg', 'abc', 'def', 'efg', 'abc', 'def', 'efg', 'abc', 'def',
'efg')
(10, 20, 30, 'P', 'ABC', 1, 2, 3, 4)
False
2316000318280
100 729

```

```

In [44]: #Sets: Unordered/Unindexed collection of unique elements | {} | set element is un
a={'apple','banana','cherry','orange','cherry'}
print(a)
b={'Haha','Python'}
print(b)
a.add(100) #add
print(a)
a.update(b) #update
print(a)
c=b.copy() #Copy
print(c)
print(id(a))
print(id(c))
d=a #Aliasing
print(d)
print(id(a))
print(id(d))
d.clear() #Clear - only contents; obj ref is stil
print(d)
print(id(d))

del d #Del

e={1,2,3,4}
f={3,4,5,6}
print(e)
print(f)

# | (Union) , & (Intersection) , - (Difference) , ^ (Symmetric difference)

g=e|f #Operator (only on sets)
print(g)
h=e.union(f) #Method (can be used for different s
print(h)

i=e&f
print(i)
j=e.intersection(f)
print(j)

k=e-f
print(k)
l=f-e
print(l)
m=e.difference(f)
print(m)
n=f.difference(e)
print(n)

o=e^f

```

```

print(o)
p=e.symmetric_difference(f)
print(p)

x={1,2,3,4,5,6,7,8,9,0}
print(x)
x.remove(0)           #Remove element - throws error if elem
print(x)
x.discard(9)          #Discard element - no error even if e
print(x)
print(x.pop())        #Pop element - Randomly removes elemen
print(x)

r={1,2,3,4,5,6,7,8,9,"Pranav"}
s={1,8,"Pranav"}

t=s.issubset(r)        #Is Subset
print(t)

u=r.issuperset(s)      #Is Superset
print(u)

v=a.isdisjoint(s)      # Is disjoint- null intersection
print(v)

```

```

{'banana', 'apple', 'cherry', 'orange'}
{'Haha', 'Python'}
{100, 'banana', 'apple', 'cherry', 'orange'}
{100, 'banana', 'apple', 'cherry', 'Python', 'Haha', 'orange'}
{'Haha', 'Python'}
1610422688232
1610404290120
{100, 'banana', 'apple', 'cherry', 'Python', 'Haha', 'orange'}
1610422688232
1610422688232
set()
1610422688232
{1, 2, 3, 4}
{3, 4, 5, 6}
{1, 2, 3, 4, 5, 6}
{1, 2, 3, 4, 5, 6}
{3, 4}
{3, 4}
{1, 2}
{5, 6}
{1, 2}
{5, 6}
{1, 2, 5, 6}
{1, 2, 5, 6}
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
{1, 2, 3, 4, 5, 6, 7, 8, 9}
{1, 2, 3, 4, 5, 6, 7, 8}
1
{2, 3, 4, 5, 6, 7, 8}

```

True  
True  
True

```

In [5]: # Dictionary: Collection of values which are unordered, Changeable & Indexed | {Key: Value}
# Vals=any Dtype & duplicated | Keys: Unrepeated & immutable

students={
    1:"Pranav",
    2:"Elon Musk",
    3:"Naval Ravikant",
    4:"Rick",
    5:"Morty"
}
print(students)
a=students[1] #Index
print(a)
b=students.get(100, 'Not Found') #get(key, error message)
print(b)

Dict={} #Empty dict
print(Dict)

Dict[1] = 'Haha'
Dict[3] = 'Bla Bla' #adding to empty dict
Dict[0] = 'LOL'
print(Dict)
Dict['Subjects'] = ['ML', 'Python', 'Big Data']
print(Dict['Subjects'][1]) #Multiple values for a key

students[1]="Sundar Pichai" #Update Dict value
print(students)

d = dict(name = "Google", age = 36, City = "New York") #dict function
print(d)

for i in d:
    print(i)

print(d.items()) #items(k,v)
print(len(d)) #len

Dict.update(d) #Update
print(Dict)

x= students.copy() #Copy-to diff ids
print(x)
print(id(students))
print(id(x))

print(sorted(students.items(), reverse=True)) #sorting

print(d.pop('age', "Key doesn't exist!!!"))
print(d.pop('height', "Key doesn't exist!!!")) #Pop(key, error message)
print(d)

print(d.popitem()) #Last item remove

print(d.clear()) #Clear Dict

```

```

del students[1]                                #Del item from dict
print(students)

del students                                    #Del whole dict

print(3 in Dict)                               #Membership - in / not in
print(900 not in Dict)

d={}
for i in range(1,11):
    d[i]=i**3
print(d)

#OR

d1={i:i**3 for i in range(1,11) if i%2==0}      #Dict Comprehension
print(d1)
{1: 'Pranav', 2: 'Elon Musk', 3: 'Naval Ravikant', 4: 'Rick', 5: 'Morty'}
Pranav
Not Found
{}
{1: 'Haha', 3: 'Bla Bla', 0: 'LOL'}
Python
{1: 'Sundar Pichai', 2: 'Elon Musk', 3: 'Naval Ravikant', 4: 'Rick', 5: 'Mort
y'}
{'name': 'Google', 'age': 36, 'City': 'New York'}
name
age
City
dict_items([('name', 'Google'), ('age', 36), ('City', 'New York')])
3
{1: 'Haha', 3: 'Bla Bla', 0: 'LOL', 'Subjects': ['ML', 'Python', 'Big Data'],
'name': 'Google', 'age': 36, 'City': 'New York'}
{1: 'Sundar Pichai', 2: 'Elon Musk', 3: 'Naval Ravikant', 4: 'Rick', 5: 'Mort
y'}
1907321121600
1907321125192
[(5, 'Morty'), (4, 'Rick'), (3, 'Naval Ravikant'), (2, 'Elon Musk'), (1, 'Sunda
r Pichai')]
36
Key doesn't exist!!!
{'name': 'Google', 'City': 'New York'}
('City', 'New York')
None
{2: 'Elon Musk', 3: 'Naval Ravikant', 4: 'Rick', 5: 'Morty'}
True
True
{1: 1, 2: 8, 3: 27, 4: 64, 5: 125, 6: 216, 7: 343, 8: 512, 9: 729, 10: 1000}
{2: 8, 4: 64, 6: 216, 8: 512, 10: 1000}

```

```

In [19]: #DATE TIME
import datetime
print(dir(datetime))
from datetime import date
d1=date.today()
print(d1)
print(d1.day)
print(d1.month)
print(d1.year)
print(type(d1))

from datetime import datetime
t1=datetime.now()                                #datetime
print(t1)
print(t1.microsecond)
print(t1.hour)

print('new: ',t1.replace(month=10,day=22, hour=15))    #replace new
print(t1.weekday())                                #weekday

import calendar
d1 = datetime.now()
print(calendar.calendar(2021))
print(d1.ctime()[ :10])                            #ctime - string slicing

date_string = "07 September, 2021"
date_object = datetime.strptime(date_string, "%d %B, %Y")    #strptime - create d
print(date_object)
print(type(date_object))

dtObj = datetime.now()
strg = datetime.strftime(dtObj, "%d %B, %Y")            #strftime- convert d
print(strg)
print(type(strg))

from datetime import timedelta
d = datetime.now()
d2 = d+timedelta(weeks=2)                                #timedelta - calculations
print(d2)

['MAXYEAR', 'MINYEAR', '__builtins__', '__cached__', '__doc__', '__file__',
 '__loader__', '__name__', '__package__', '__spec__', 'date', 'datetime', 'dat
etime_CAPI', 'time', 'timedelta', 'timezone', 'tzinfo']
2021-09-27
27
9
2021
<class 'datetime.date'>
2021-09-27 09:37:50.481581
481581
9
new:  2021-10-22 15:37:50.481581
0

```



2021

January

Mo	Tu	We	Th	Fr	Sa	Su
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

February

Mo	Tu	We	Th	Fr	Sa	Su
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28

March

Mo	Tu	We	Th	Fr	Sa	Su
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

April

Mo	Tu	We	Th	Fr	Sa	Su
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

May

Mo	Tu	We	Th	Fr	Sa	Su
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

June

Mo	Tu	We	Th	Fr	Sa	Su
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

July

Mo	Tu	We	Th	Fr	Sa	Su
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

August

Mo	Tu	We	Th	Fr	Sa	Su
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

September

Mo	Tu	We	Th	Fr	Sa	Su
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

October

Mo	Tu	We	Th	Fr	Sa	Su
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

November

Mo	Tu	We	Th	Fr	Sa	Su
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

December

Mo	Tu	We	Th	Fr	Sa	Su
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

```

Mon Sep 27
2021-09-07 00:00:00
<class 'datetime.datetime'>
27 September, 2021
<class 'str'>
2021-10-11 09:37:50.484572
1

```

In [23]: *#FILES*

-----

File "<ipython-input-23-b05d670f2432>", line 13

-----

^

**SyntaxError:** invalid syntax

In [24]: *#numpy*

```

import numpy as np

s=np.array([10,20,30,40,50,60])
print(s)
print(type(s))

b= np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
print(b)

print(np.zeros(5))
print(np.ones(5))
print(np.arange(4))

x=np.ones(5,dtype=np.int64)
print(x.dtype)

print(b.ndim)
print(b.size)
print(b.shape)
print(b[0:3])
print(b[2:3])

a = np.array([[1,2,3,4],[5,6,7,8]])
print(a)
print(np.sum(a,axis=1))           #axis 0 - column / axis 1 - row
print(np.mean(a,axis=1))
print(np.sort(a[:,1]))           #default- ascending order

```

```

[10 20 30 40 50 60]
<class 'numpy.ndarray'>
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
[0. 0. 0. 0. 0.]
[1. 1. 1. 1. 1.]
[0 1 2 3]
int64
2
12
(3, 4)
[1 2 3 4 5 6 7]
[[ 9 10 11 12]]
[[1 2 3 4]
 [5 6 7 8]]
[10 26]
[2.5 6.5]
[[5 6 7 8]
 [1 2 3 4]]

```

```

In [25]: class Solution(object):
          def peopleIndexes(self, favoriteCompanies):
              """
              :type favoriteCompanies: List[List[str]]
              :rtype: List[int]
              """
              dic_companies = {}
              inx = 0
              for companies in favoriteCompanies:
                  for company in companies:
                      if company not in dic_companies:
                          dic_companies[company] = inx
                          inx += 1

              favorite_comp = []
              for companies in favoriteCompanies:
                  val = 0
                  for company in companies:
                      val += 2**dic_companies[company]
                  favorite_comp.append(val)

              result = []

              for i in range(len(favorite_comp)):
                  flag = True
                  for j in range(len(favorite_comp)):
                      if i == j: continue
                      if favorite_comp[i] | favorite_comp[j] == favorite_comp[j]:
                          flag = False
                          break
                  if flag:
                      result.append(i)

              return result

```

```

In [41]: A={10,15,20,30,40}
          B={10,3,15,7,9}

          # union
          print("Union :", A | B)

          # intersection
          print("Intersection :", A & B)

          # difference
          print("Difference :", A - B)

          # symmetric difference
          print("Symmetric difference :", A ^ B)

```

```

Union : {3, 7, 40, 9, 10, 15, 20, 30}
Intersection : {10, 15}
Difference : {40, 20, 30}
Symmetric difference : {3, 20, 7, 40, 9, 30}

```

In [ ]: