Bitwise operators

```
And
Or
Not
Nor
```

function

```
reusability peace of code
advantages:modularity,resuability,
```

In [5]:

```python
def sumofnumbers(a,b):
    return a+b
sumofnumbers(5,8)
```

Out[5]:

```
13
```

In [6]:

```python
#functional
def sumofnumbers(a,b):
    return a+b
print(sumofnumbers(5,6))

```

```
11
```

types of arguments

```
1.positional-
2.keyword-
3.default-
4.variable length-
they 2 function userdefine and built in
```

In [9]:

```python
#positional
def sumofnumbers(a,b):
    print(a)
    print(b)
    return a+b
c=sumofnumbers(5,5)
print(c)
```

```
5
5
10
```

In [17]:

```
#default argument
def myself(name,age):
    print(name)
    print(age)
    return name
print(myself("priya",age=20))
```

```
priya
20
priya
```

In [19]:

```
#keyword based argument
def goodmorning(name,age=20):
    print(name)
    print(age)
    print("morning",name)
    return None
goodmorning("sir")
goodmorning("sir",age=19)
goodmorning("hello",age=21)
```

```
sir
20
morning sir
sir
19
morning sir
hello
21
morning hello
```

In [64]:

```
def sum():
    sum=0
    for i in range(1,11):

        sum+=i
    print(sum)
sum()

```

```
55
```

list

```
list is mutable
list is ordered
```

In [74]:

```
my_list=[1,2,3,4,4,4]
print("list",my_list)
#methods in list:
#1.append
my_list.append(5)
print("append",my_list)
#2.extend
my_list.extend([6,7,8])
print("extend",my_list)
#3.insert
my_list.insert(7,11)
print("insert",my_list)
#4.remove
my_list.remove(5)
print("remove",my_list)
my_list.pop(1)
print("pop",my_list)
#count
a=my_list.count(4)
print("count",a)
# or
print("count",my_list.count(1))

```

```
list [1, 2, 3, 4, 4, 4]
append [1, 2, 3, 4, 4, 4, 5]
extend [1, 2, 3, 4, 4, 4, 5, 6, 7, 8]
insert [1, 2, 3, 4, 4, 4, 5, 11, 6, 7, 8]
remove [1, 2, 3, 4, 4, 4, 11, 6, 7, 8]
pop [1, 3, 4, 4, 4, 11, 6, 7, 8]
count 3
count 1
```

In [2]:

```
#list input
user_input=input()
number=list(map(int,user_input.split()))
print(number)
```

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

In [3]:

```
#for removing [] for above output
user_input=input()
number=list(map(int,user_input.split()))
print(*number)
```

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

aggreate function

```
    min()
       ``
```

Tuple

In [77]:

```
my_tuple=(1,2,3,4,5)

#index
print("index",my_tuple.index(3))


```

```
index 2
```

Set

```
add
update
remove
discribe
pop
clear
copy
union
intersection
difference
symmetric difference
```

In [22]:

```
my_set1={1,2,3,4,5}
my_set2={6,7,8,9,10}
my_set1.add(6)
print("after add",my_set1)

my_set1.update({7,8})
print("update",my_set1)

my_set1.remove(8)
print("remove",my_set1)

union_set=my_set1.union(my_set2)
print("union",union_set)

intersection=my_set1.intersection(my_set2)
print("intersection",intersection)

diff=my_set1.difference(my_set2)
print("difference",diff)

sym_diff=my_set1.symmetric_difference(my_set2)
print("sym_difference",sym_diff)


```

```
after add {1, 2, 3, 4, 5, 6}
update {1, 2, 3, 4, 5, 6, 7, 8}
remove {1, 2, 3, 4, 5, 6, 7}
union {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
intersection {6, 7}
difference {1, 2, 3, 4, 5}
sym_difference {1, 2, 3, 4, 5, 8, 9, 10}
```

Dictionaries

In [26]:

```python
my_dict=dict()
my_dict=dict(one=1,two=2,three=3)
print(my_dict)

my_dict_new={'one:1','two:2','three:3'}
print(my_dict_new)
my_dict_new.clear()
#2.copy
#3.from keys
#we will print elements using keys and list..
co=my_dict.copy()
print(co)

keys=['one','two','three']
values=0
my_dict_new_2=dict.fromkeys(keys,values)
print(my_dict_new_2)
my_dict_new_3={'one':1,'two':2,'three':3}
print(my_dict_new_3.get('one'))




```

```
{'one': 1, 'two': 2, 'three': 3}
{'two:2', 'three:3', 'one:1'}
{'one': 1, 'two': 2, 'three': 3}
{'one': 0, 'two': 0, 'three': 0}
1
```

OOPS-object oriented programming language--it is combinnation of class and object

```
class-blue print of object,once class is created memory is allocated
object-real world entity
constructor-allocates memory
destructor-deletes the memory
default constructor is __init__ constructor.
destructor is __del__
access modifiers:accessibility

they are two variables
class variable
instance variable
---------------------------------
```