FUNCTIONS(module)

a block of code to do a particular task.

```
def sum(a,b):
	c=a+b
	print(c)
sum(10,20)#calling sum() function
```

Need of the function:

Avoids repetition in code
Functions can be reused(reusability)

```
sum=0
for i in range(4,10):
	sum=sum+i
print(sum)
sum=0
for i in range(10,15):
	sum=sum+i
print(sum)
sum=0
for i in range(3,9):
	sum=sum+i
print(sum)
```

```
def sum(x,y):
	sum=0
	for i in range(x,y):
		sum=sum+i
	print(sum)
sum(4,10)
sum(10,15)
sum(3,9)
```

Note

- first define the function then call
- unless you call, you dont enter into function definition

arguments

```
def my_function(name):
    print(name)
my_function("tree")
my_function(3.4)
my_function(6)
```

formal and actual arguments

```
def sum(a,b):
    c=a+b
    print(c)
x=10
y=20
sum(x,y)
```

positional parameters

```
def mydata(age,sal,city,gen):
    print(age,sal,city,gen)
mydata(20,25000,"tokyo",'f')
```

keyword parameters

```
def mydata(age,sal,city,gen):
    print(age,sal,city,gen)
mydata(city="tokyo",gen='f',age=20,sal=25000)
```

keyword arguments must come after positional arguments

```
def mydata(x,y,z,age,sal,city,gen):
    print(age,sal,city,gen,x,y,z)
mydata(3,4,5,city="tokyo",gen='f',age=20,sal=25000)
```

flow of execution

```
print("welcome IT GUYS")
def peep():
    print("entered into  peep")
    print("milk")
    print("leaving peep")
def see():
    print("entered into  see")
```

```
        print("tea")
        peep()
        print("leaving see")
def look():
        print("entered into  look")
        print("coffee")
        see()
        print("leaving look")
look()
print("bye IT GUYS")
```

Function returning multiple values

```
a=10
b=[2,3,4,5]
c="rossel leaves"
def multi():
        print("yoghurt")
        return a,b,c
x,y,z=multi()
print(x,y,z,sep=",")
```

local and global variables

```
a=10#global variables
def eat():
        a=20#local to eat()
        print(a)
def vomit():
        print(a)
        eat()
vomit()
print(a)
```

fruitful function

functions that return a value

```
def square(val):
        return val*val
n=int(input("enter num:"))
result=square(n)
print("the suare of ",n," is:",result);
```

a function returning multiple values

```
a=10
b=[2,3,4,5]
c="rossel leaves"
def multi():
        print("yoghurt")
        return a,b,c
x,y,z=multi()
print(x,y,z,sep=",")
```

Recursive function

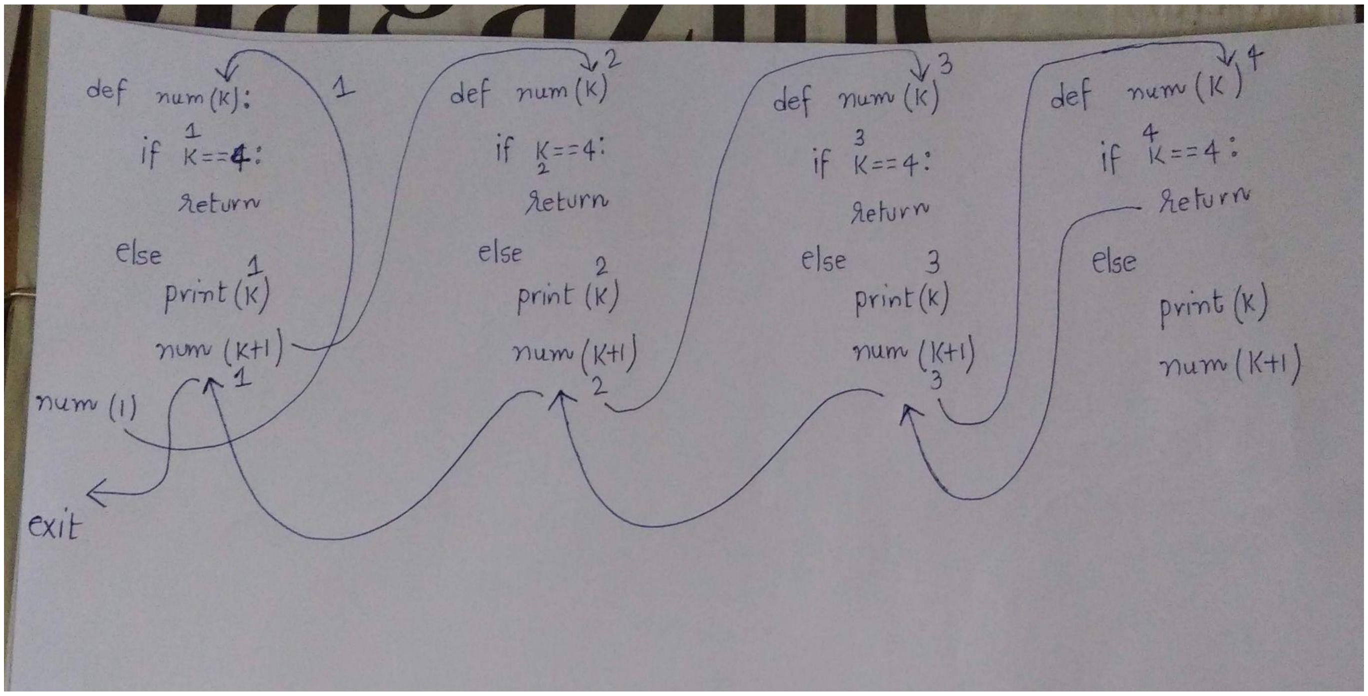
It is a process of calling a function by itself

Infinite Recursion

```
def show():
  print("i am tired")
  show()
print("welcome")
show()
print("bye")#unreachable (dead code)
```

Displaying 1-10 numbers using recursion

```
def num(k):
  if k==3:        //base condition to come out of recursion
     return
  else:
     print(k)
     num(k+1)
num(1)
```



```
def num(K):
  if K==4:
     return
  else
     print(K)
     num(K+1)
num(1)
```

```
def num(K)
  if K==4:
     return
  else
     print(K)
     num(K+1)
```

```
def num(K)
  if K==4:
     return
  else
     print(K)
     num(K+1)
```

```
def num(K)
  if K==4:
     return
  else
     print(K)
     num(K+1)
```

Note:

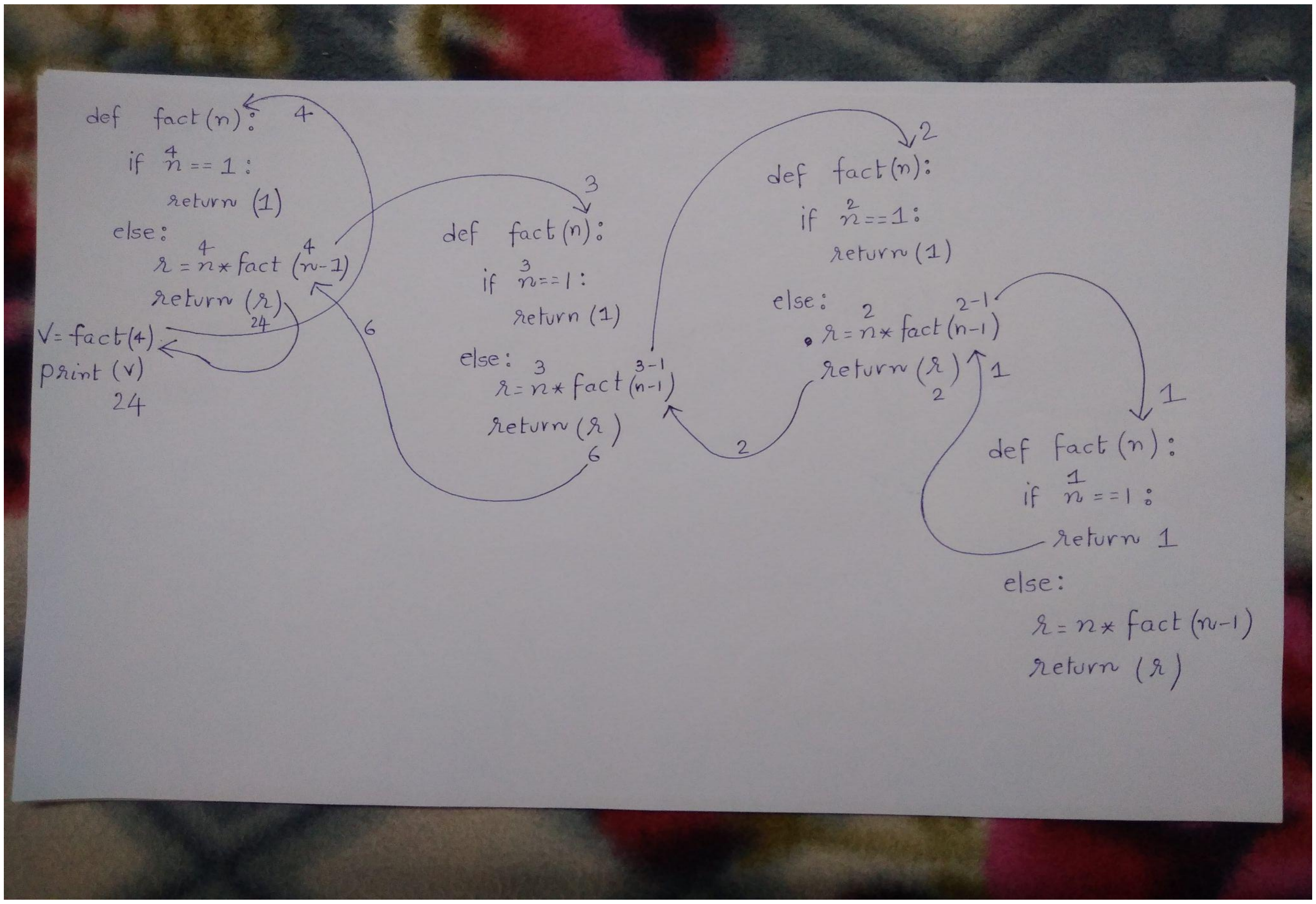
- every iterative problem can be expressed in terms of recursion(Whatever can be computed using recursion can also be computed using iteration, and vice versa.)

- Generally, a recursive function generally takes more time to execute than an equivalent iterative approach

- it is generally best to use an iterative approach .

Finding factorial

```
def fact(n):
    if n==1:
        return 1
    else:
        r=n*fact(n-1)
        return r
val=fact(1)
print(val)
```



```
def fact(n):
    if n == 1:
        return (1)
    else:
        r = n*fact(n-1)
        return (r)
V= fact(4)
print (v)
    24
```

```
def fact(n):
    if n==1:
        return (1)
    else:
        r= n*fact(n-1)
        return (r)
```

```
def fact(n):
    if n==1:
        return (1)
    else:
        r = n*fact(n-1)
        return (r)
```

```
def fact(n):
    if n ==1:
        return 1
    else:
        r = n*fact(n-1)
        return (r)
```

Finding nth fibonacci number

```
#0,1,1,2,3,5,8,13




def fib(n):

    if n==1:
        return 0
    elif n==2:
        return 1
    else:
        val=fib(n-1)+fib(n-2)
        return val
r=fib(7)
print(r)
```

Find sum of 1 to n numbers

```
def sum(n):
    if n==1:
        return 1
    else:
        r=n+sum(n-1)
        return r
val=sum(4)
print(val)
```

Displaying a string using recursion

```
name="svec"
def show(i):
    if i==-5:
       return
    else:

       print(name[i],end="")
       show(i-1)
show(-1)
print("\n")
```

```
name="i like potato in lunch"
slen=len(name)
```

```
def show(i):
    if i==slen:
       return
    else:

       print(name[i],end="")
       show(i+1)
show(0)
print("\n")
```

Exception handling

Exception:

An abnormal event or situation which stops normal flow of execution of a program or task

Few real time examples

- Pen not functioning in the middle of the exam
- Running out of petrol in the middle of a journey
- running out of gas in the middle of cooking
- Entering restricted characters in e –mail address
- The amount passed to withdraw exceeds the account's balance.
- Power cut in the middle of using word document
- You issue a command to read a file from a disk, but the file does not exist there.
- You attempt to write data to a disk, but the disk is full

All of the above are special cases because they occur rarely

Note:

- These errors are called exceptions because they are not usual   occurrences; they are “exceptional.
- Exception means special case
- When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

Handling an exception

dividing a number with zero causes an exception

```
d=int(input("enter dividend:"))
n=int(input("enter divisor:"))
r=d/n
print(r)
print("bye")#it is not reachable if n=0
```

To handle raised exceptions we use two blocks

```
try
        where ,the code which may raise an exception is placed
except
        It tells what to do after occurance of exception
```

so,from try block we go to except block

we enter into except block only when exception is raised in try   block

```
r=0
d=int(input("enter dividend:"))
n=int(input("enter numerator:"))
try:
	r=d/n
except ZeroDivisionError:
	print("division  with zero not possible")

print(r)
print("bye")
```

----

try with multiple except blocks

```
r=0
try:
	d=int(input("enter dividend:"))
	n=int(input("enter numerator:"))
	r=d/n
	print(r)
except ZeroDivisionError:
	print("division  with zero not possible")
```

```
except ValueError:
	print("please enter values in correct format")
print("bye")
```

```
r=0
try:
	d=int(input("enter dividend:"))
	n=int(input("enter divisor:"))
	r=d/n
	print(r)
except ZeroDivisionError:
	print("division  with zero not possible")
except ValueError:
	print("please enter values in correct format")
except KeyboardInterrupt:
	print("u pressed ctrl+c ....")
print("bye")
```

----

few exception types

```
a=[2,3,4,5]
try:
	print(a[3])
except IndexError:
	print("bindu,index is not valid")
print("ape")
print("weep")

------

a={'s':'sun','p':'pig','m':'mic'}
try:
	key=input("enter key:")
	print(a[key])
except KeyError:
	print("bindu,key is not valid")

print("ape")
print("weep")

------------

try:
	print(a)
except NameError:
	print("bindu,variable trying to access is not declared")

print("ape")
print("weep")

------------
a=(1,2,3,4)
try:
	a[0]=11#not possible
except TypeError:
	print("bindu,that is not possible")

print("ape")
print("weep")
```

```
r=0
try:
	d=int(input("enter dividend:"))
	n=int(input("enter divisor:"))
	r=d/n
	print(r)
except:
	print("something has gone wrong.....sorry")
print("bye")
```