Python-Tutorial 4

- function with pre-defined arguments:

```
def test(a, b=1):
    return a+b
test(5)  → 6
test(50, 10) → 60          (overwrite b)
def test(a, b=1, m, n, m):
             X (no duplicate data).
```

- argument as list:

```
def modifylist(a):
    l = []
if type(a) == list:
    for i in a:
        l.append(i+4)
else:
    print("input is not a list")
return l
modifylist([4, 5, 6, 7, 8]) → [8, 9, 10, 11, 12]
modifylist("Hello") → (input is not a list)
```

- argument as int./str.:

```
def greeting(name): → placeholder
    print('Hello %s' % name)
```

! { Writing a code is like, writing a story : just build a thought (<u>logic</u>) }

(%.d → integer
%.f → float
%.s → string)

greeting ("MADHU") ↷ Hello MADHU .

• whether a number is prime / not prime :

```
def isprime(num):
    for n in range(2, num):
        if num % n == 0 :      (% = remainder)
            print('not prime')
            break
    else ~~else~~ :
            print('prime')
```

```
isprime (17)   ⟶  prime
isprime (345)  ⟶  not prime
```

↓ alternate code (optimize above method ↓):
(just check upto sq. root of that number)

```
import math
def isprime (num):                          ← (even)
    if num % 2 == 0 and num > 2 :
        return noprime
    for i in range (3, int (math.sqrt(num)
                    +1 , 2) :   (only odd)
        ~~set~~ if num % i == 0:
            return ~~false~~ noprime
    return ~~true~~ prime
```

- Iterables, Iterators & Generators :

```
range(8) → range(0, 8)
```

(Range only generates data, it doesn't shows, so range is generator function).

```
list(range(8)) → [0, 1, 2, 3, 4, 5, 6, 7]
```

```
def gencube(n):
    for num in range(n):
    print (num**3)
gencube(5) →  0
              1
              8
              27
              64
```

```
def gencube(n):
    for num in range(n):
        yield(num**3)
gencube(5) → generateobject
```

so, yield is generator function, doesn't gives data directly, untill & unless we iterate over ~~to~~ it & extract it.

```
list(gencube(5)) →  0
                    1
                    8
                    27
```

{ yeild will try to create generator which we need to call inside some iterator (ex: for loop etc.) for data extraction.

• string:

```
{ for i in "MADHU":  →  [ M ]
      print(i)          [ A ]
                        [ D ]
                        [ H ]
                        [ U ] *
  next("MADHU")
  ↳ 'str' object is not an iterator.
```

{ **Note:-** Iterator means, we'll be able to extract data one-by-one.
Only iterable can be converted into iterator. Or if something is not iterable you can't convert it into iterator. }

String itself is not an iterator, but it is iterable, so this concept is used by for Loop (*).
By help of iter function we convert iterable function into iteration & this same concept is used by for loop internally.

ex:
```
        s = [5,6,7,8]
        next(s) → list is not iterator
      c = iter(s)  (∵ but list is iterable, so we
                     convert it by used of iter)
      next(c) → 5 \      next(c) → stopiteration
      next(c) →
```

(generator functions are iterator functions by default, so we can perform on them directly by using next(), rather than iter())

- generate fibonacci sequence :

```
def genfibon(n):
a=1
b=1
for i in range(n):
    yield a
    a, b = b, a+b
```

```
for num in genfibon(10)
    print(num) ⟶
```

```
1
1
2
3
5
8
13
21
34
56
```

(system won't be hanged ↑
will be hanged ↓)
So, check it out ☺

(*) Alternate :

```
def fibon(n)
a=1
b=1
output = [ ]
for i in range(n)
    output.append(a)
    a,b = b, a+b
return output
```

~~4~~
~~8~~
2

fibon (10) ⟶ [1, 1, 2, 3, 5, 8, 13, 21, 34, 56]

{ map : mapping of a function with large data. }
(map will return some kind of collection.)

• map() function : (for large data set & single operation)

```
def fahrenheit(T):
    return((float(9)/5)*T+32)    (external function)
temp = [0, 22.5, 40, 100]
l = [ ]
for i in temp:
    l.append(fahrenheit(i))
l → [32.0, 72.5, 104.0, 212.0]
```

(*) Alternate : (above in single line code):

```
map(function, iterable object)
F_temps = list(map(fahrenheit, temp))
F_temps → [32.0, 72.5, 104.0, 212.0]
```

• lambda : (annonymus function / inline function)

```
list(map(lambda x: x**5, l))
                arg  return data
↳ [32.0, 72.5, 104.0, 212.0]
```

(∵ no need to call external function for small tasks just create some lambda function.)

[a, b = b, a+b] ←different→ [a = b
b = a+b]

a & b are updated together

a will be updated first & then b.

```
a = [1, 2, 3, 4]
b = [5, 6, 7, 8]
c = [9, 10, 11, 12, 13, 4]
def test(x, y, 3)
        return x+y+3
list(map(test, a, b, c))
        ↳ [15, 18, 21, 24]
list(map(lambda x, y, 3 : x+y+3, a, b, c))
        ↳ [15, 18, 21, 24]
```

- Reduce : (returns single value).
  reduce(function, sequence)

```
l = [47, 11, 42, 13]        (∵ can also be str, char)
def summation(a, b):
        return a+b
reduce(summation, l) → 113
```

Alternate : (use lambda function):

```
reduce(lambda a,b: a+b, l) → 113
```