Functional programming — 2

⇒ Principle of FP :

① Data should be seperate from mutation

② Treat Variables as immutabl

③ Treat f(x) as FCC.

⇒ F.C.C ⇒ First Clay Citizen

↳ [which can be stored in a Variable and Passed as well as retuned from a F(x

Ex , int, List

⇒ Store f(x) into Var ✓

⇒ Function as Argument ✓

⇒ return Function ✓

① lambda

⇒ f = $\boxed{\text{lambda } x : x**2}$

⇒ a=3 , b=f(a)

⇒ a = lambda x : x+10

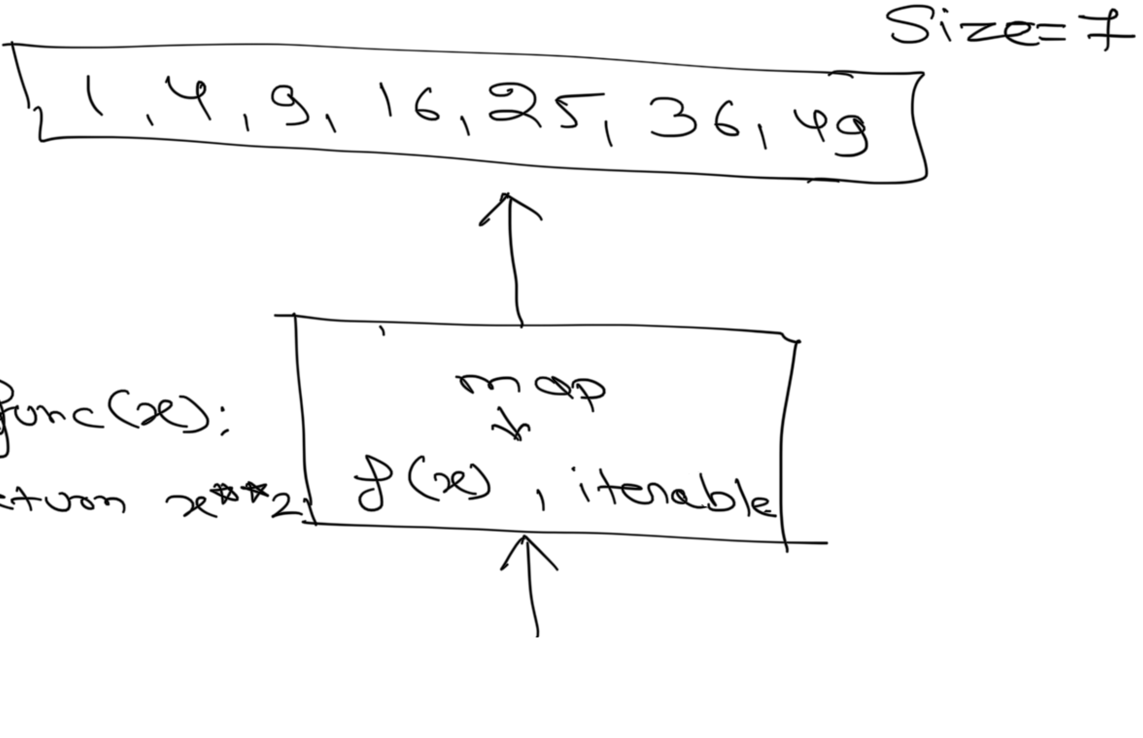
Functions to acheive F.P.

① map

② Filter

③ Reduce

④ Zip



```
def func(x):
    return x**2
```

=> 1, 2, 3, 4, 5, 6, 7

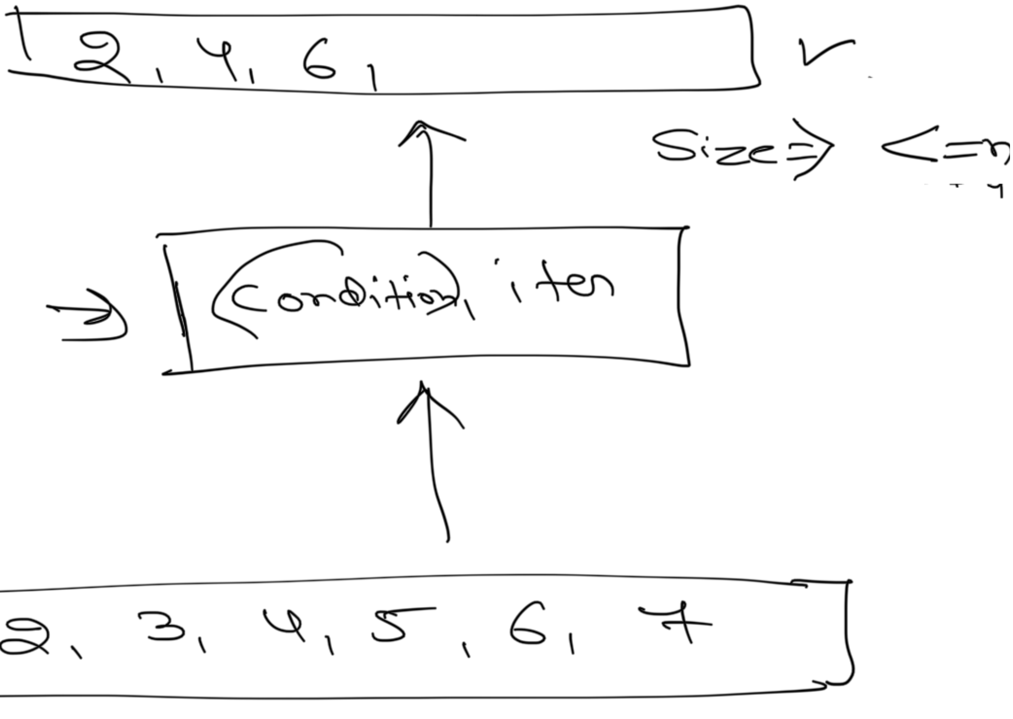
⇓ Li

map(fun, L1)

=> f(it[0]), f(it[1]), f(it[2]) - - -

=>

Filter =>


2, 4, 6,

✓

Size => <=n

True or false => (condition), iter

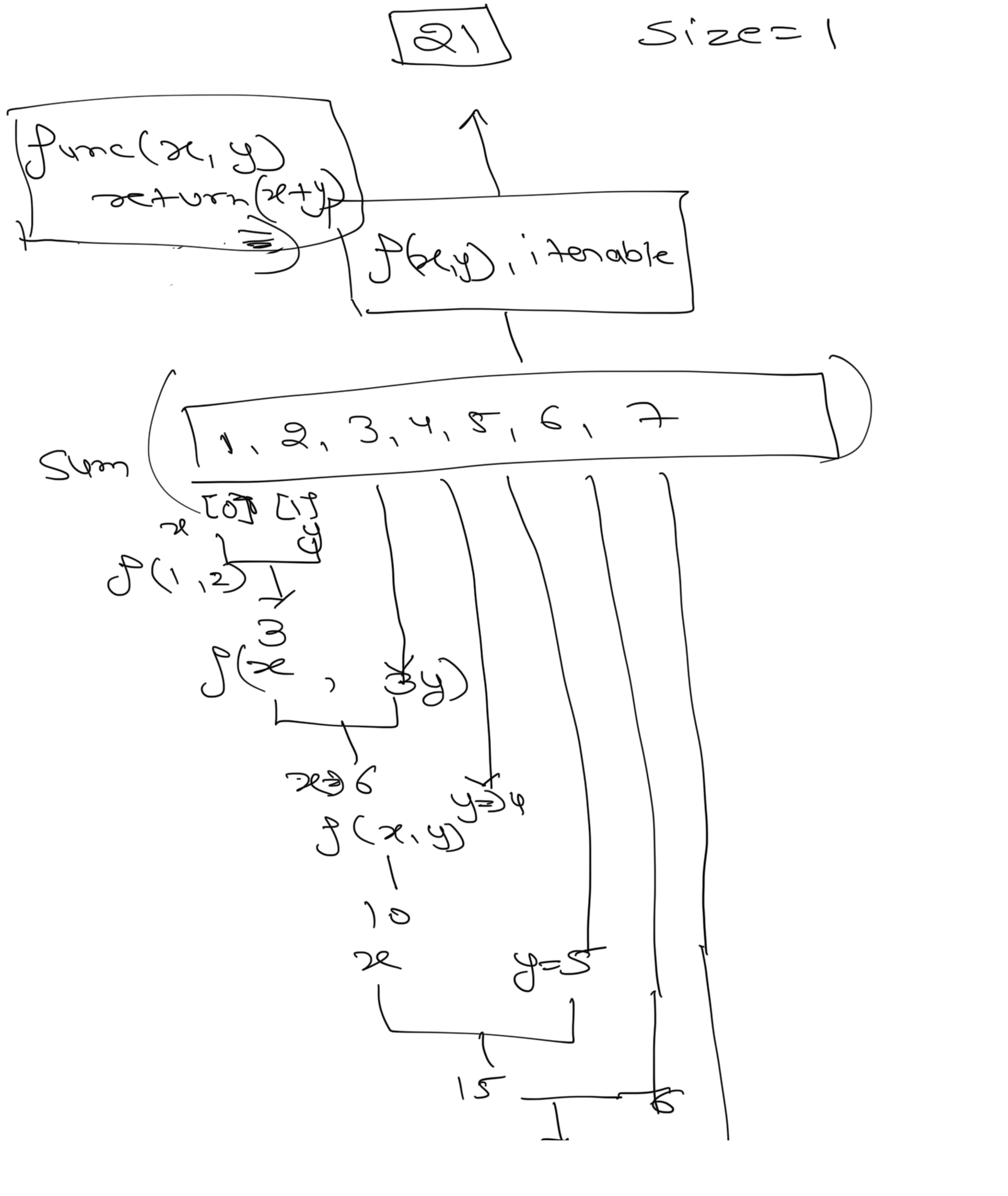
1, 2, 3, 4, 5, 6, 7

funct

```
if x % 2 == 0:
    return True
else
  ret False
```

=> even

⇒ Reduce


21
size = 1
func(x, y)
return(x+y)
f(x,y), iterable
Sum
1, 2, 3, 4, 5, 6, 7
[0] [1]
x
y
f(1, 2)
3
f(x , 3y)
x⇒6
y⇒4
f(x, y)
10
x
y=5
15
6

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

f(x, y) —— ↓ reduce(f, L)

(x+y)

⇒ 'abcd'

	map	reduce	Fiter
input			
Output			
Syntax			
Example			