FP-2 →

⇒ Agenda
- Principle of FP
- Maps
- Filters
- Zip
- Reduce
- Arg & kwargs

⇒ Principles of FP →

1. Data should be seperated from mutations
2. Treat variables as Immutable → unchangeable
3. Treat f(x) as FCCs.

⇒ Map | Filter | Reduce → Function that also takes another function as input.

→ Map →

Transforming data → (Complex data
↓
easy to understand data)

map (function_to_perform, *iterables)

- → List
- → dict
- → set
- → tuples

↓

returns a reference of map.

if we want to check the final outcom

↓

Convert it into list.

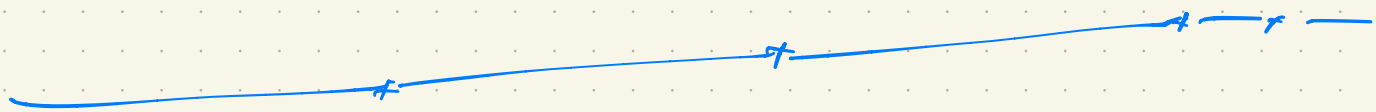
⇒ if a = [2, 4, 3, 7, 9, 10]

if item is even → select those element

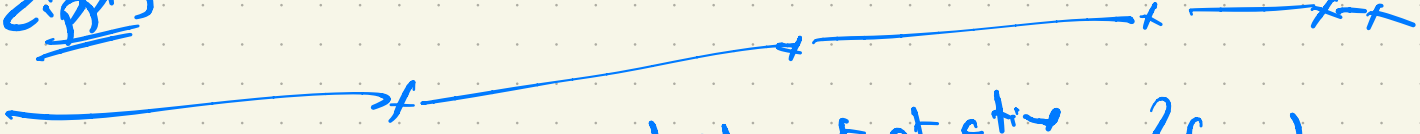
Map → Used to transform data / apply a function on each item of iterable.

Filter → filter out items from a iterable based on a condition.

Filter → takes a functional argument
↓
only returns value where the function outcome is True.



Zipping →



Reduce → map → 1 element at a time
filter → 1 element at a time
zip → 1 element at a time
} from 1 iterable

Reduce → works on 2 items from the same iterable at any time.

a = [1, 2, 3, 4, 5]

f(x,y) = x+y

func

→ Pass Step 1 a[0] = x a[1] = y

f(x,y) → 1+2 = 3

Step 2 x = 3 y = a[2]

f(x,y) → 3+3 = 6

Step 3 x = 6 y = a[3]

f(x,y) = 6+4 = 10

Step 4 x = 10 y = a[4]

f(x,y) = 10+5 = 15

↓

output

⇒ Any operation which are dependent on previous steps are done using reduce.

⇒ a = [10, 15, 30, 27, 29, 35]

```
cnt = 0
for i in a:
    if i%5==0:
        cnt+=1
```

(How many elements are divisible by 5)

reduce(func , iterable , Start value)

func → 2 variable

Start value → optional.

reduce

a[0] a[1] → f(x0, y0)

f(x0 y0) a[2]

⋮

Below with start

Start a[0]

f(s,a) a[1]

a=[2,10,22,36,45,57]

if x,s ≡ 0 then x+1 else

1st step x=2 y=10 f(x,y)= 3

2nd x=3 y=22 f(x,y)= 3

= 3

3rd x=3 y=36

4th x=3 y=45 f(x,y)=3+1=4

Args | kwargs →

⇒ def random (x, y) →

return x+y

⇒ How to handle unknown argument lengths

⇒ *args → tuple of all the undeclared inputs of the function

Order of passing arguments →

Positional → Args → keyword → kwargs .

Positional value → defined argument

Positional Argument → extra argument

keyword argument → extra argument

Doubts

→ reduce (funct() , iterable)

Fibonacci

1 1 2 3 5 8

→ keep doing cumulate sum

[4, 5, 7, 16, 13, 20]

null, 6, 13, 23, 36, 56

reduce (lambda x,y: x+y , a]

⇒ • If a custom class is there →

add is given prefer

C1 + C2

add
radd

add
radd

C1 + C2 → radd2

add1

Inbuilt class → Str/Int → explicitly returned to Caller

A × C

Col1	Col2	Col3
A	C	—
	D	—
	E	—
B	F	—
	D	—
	E	—

Col3

→ People who are living in B3

GDP of India

$A_{ow} = \frac{\text{Var b/w groups}}{\text{Var within group}}$

COllab : https://colab.research.google.com/drive/1XooKzL7mhjj_8cPgi0kvTvPxKjAK9FA4?usp=sharing