Fabio Hedfam Gagano Siregar

120450100

RB

Buatlah sebuah fungsi bernama ulangi_NIM, ulangi memiliki input sebuah bilangan skalar a, dan mengeluarkan vektor 1xn dengan seluruh elemen nya adalah a !

In [1]:
```
n = 8
def ulangi_100(n):
    a = list(map(lambda n: 1*n, range(1,n+1))) #scalar number code
    return a
print('bilangan skalar :', n)
ulangi_100(n)
```

```
bilangan skalar : 8
[1, 2, 3, 4, 5, 6, 7, 8]
```
Out[1]:

Buatlah deret bilangan sebagai berikut dengan input n sebagai panjang deret:

In [2]:
```
#row length
n = 5
deret = list(map(lambda x: ((-1)*(x+1)) * (1/(2*x)), range(1,n+1)))
print(deret)
```

```
[-1.0, -0.75, -0.6666666666666666, -0.625, -0.6000000000000001]
```

In [3]:
```
b = range(1,n+1)

def pola_deret(x):
    return ((-1)*(x)) * (1/(2*(x+1)))
print(list(map(pola_deret, b)))
```

```
[-0.25, -0.3333333333333333, -0.375, -0.4, -0.41666666666666663]
```

Jumlahkan deret bilangan tersebut!

- first, import library "functools" to retrieve reduce . function
- second, write the function with reduce that we have

In [4]:
```
from functools import reduce
print(reduce(lambda x,y: x+y, deret))
```

```
-3.6416666666666666
```

Sebuah DNA dimodelkan dalam sebuah string menjadi sequence TCGA dan disimpan ke dalam data :

the first dat is used to read "txt", then the second dat is used to remove "/n" in the data when it is read

In [5]:
```
dat = open("dna.txt", 'r').read()
dat=dat[:-1]
seq='ACT'
```

The append function is used to add n characters to i ; remap function is useful to remap all seq functions

In [6]:
```
seq="ACT"
def append_n(dat, i, n):
    return reduce(lambda x, y: x + y, [dat[i:i+n]])
append_n(dat, 0, 3)
```

Out[6]:
```
'TGT'
```

In [ ]:
```
def remap(dat, seq):
    return map(lambda x:append_n(dat, x,len(seq)), range(0, len(dat)-len(seq)+1))
list(remap(dat,"ACT"))
```

In [8]:
```
def count_mer (dat,seq):
    return reduce(lambda x,y:x +(1 if y==seq else 0), remap(dat,seq), 0)
count_mer(dat,"ACT")
```

Out[8]:
```
106
```

In [9]:
```
list(remap(dat,'ACT'))[-1]
len(dat)
```

Out[9]:
```
6930
```

- *sequence is used for dictonary to store the result of count_mer*
- *count_all is used to count all seq*
- *res is used to call all number of seq we are looking for*

In [10]:
```
sequences=["A","AT","GGT", "AAGC", "AGCTA"]

def count_all(dat, sequences):
    return map(lambda x: count_mer(dat,x), sequences)

res=count_all(dat,sequences)
print(*res)
```

```
2112 557 77 22 5
```

- *complement is used as a library for complement*
- *reverse complement is used to convert complement to reverse*

In [11]:
```
def komplemen(x):
    return{'A':'T','T':'A','C':'G', 'G' :'C'}.get(x)

def reverse_komplemen(f):
    return map(lambda x:komplemen(x),f)
```

In [ ]:
```
res = reverse_komplemen(dat) #to call all komplemen
```

```
print(*res)
```

Number 6

- *first we need to call the math library*
- *def activasi is used to calculate activation function*
- *def WTi is used to transpose matrix*
- *def WT is used to accommodate calculations*
- *def XW is used for calculations for one input*
- *def input_to_hidden is used to run the activation function*
- *W shape must be the same as M to match the pattern that has been made*

In [13]:
```
import math

def aktivasi(x):
    return 1/(1+math.exp(-x))
def WTi(W,i):
    return list(map(lambda w:w[i],W))
def WT(W):
    return list(map(lambda i:WTi(W,i),range(len(W[0]))))
def XW(X,W):
    return map(lambda w: reduce(lambda a,b:a+b, map(lambda xx,ww:xx*ww,X,w),0),WT(W))
def input_to_hidden(X,W):
    return list(map(lambda x:aktivasi(x),XW(X,W)))
def feed_forward(X,W,M):
    return input_to_hidden(input_to_hidden(X,W),M)
```

In [14]:
```
X=[9,10,-4]
W=[[0.5,0.4],[0.3,0.7],[0.25,0.9]]
M=[[0.34],[0.45]]

feed_forward(X,W,M)
```

Out[14]:
```
[0.6876336740661236]
```