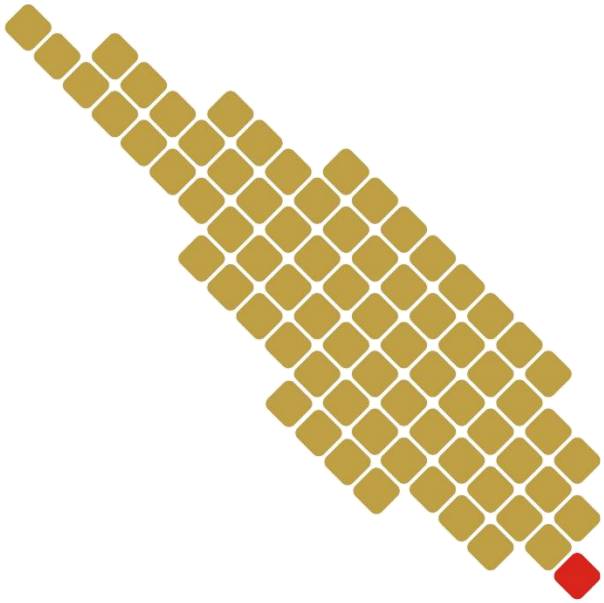
Laporan Akhir Praktikum

Pemograman Berbasis Fungsi



ITERA

SHAHNAZ SALSABILA ISHAK

120450065

RA

Institut Teknologi Sumatera

Lampung Selatan

2022

LATIHAN SOAL

Chapter 9

May 16, 2022

0.1 1. Buat program untuk menghitung deret bilangan prima dari 2 hingga N menggunakan HOF filter dan map

[10]:
```
def factor(N):
    return list(filter(lambda x: N%x == 0, range(1,N+1)))
def primes(N):
    return list(filter(lambda x: len(factor(x))  == 2 , range(1,N+1)))

print(primes(100))
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73,
79, 83, 89, 97]
```

0.2 2. Terdapat dictionary employee berisi nama dan umur pegawai, lakukan filter untuk mengetahui pegawai yang berumur > 25 tahun !

[11]:
```
employee = {
            'Nagao':35,
            'Ishii':30,
            'Kazutomo':20,
            'Saito':25,
            'Hidemi':29}
```

[18]:
```
def age_morethan(dictionary, age):
    return list(filter(lambda x: x[1] > age, dictionary.items()))
def emp_res(dictionary, age):
    return [i[0] for i in age_morethan(dictionary, age)]

print(* emp_res(employee,25))
```

```
Nagao Ishii Hidemi
```

Chapter 10

May 16, 2022

0.1 1. Buat fungsi mencari jumlah bilangan genap dari list L!

[1]:
```
L = [2,1,9,10,3,90,15]
```

[4]:
```
from functools import reduce as r

def total_genap(l):
    return r(lambda a,b: a+1 if b%2 == 0 else a, l,0)

total_genap(L)
```

[4]: 3

0.2 2. Buat fungsi untuk menghitung n! Menggunakan reduce!

[11]:
```
def faktorial(n):
    return r(lambda a,b: a*b, range(1,n+1))

faktorial(6)
```

[11]: 720

0.3 3. Hitung euclidian distance dari dua vektor berikut menggunakan higher order function!

[16]:
```
x = [2,5,6,7,10]
y = [-2, 9,2,-1,10]

def euclid_distance(x,y):
  res = list(map (lambda x,y: (x-y)**2 , x,y  ) )
  return (r (lambda a,b : a + b, res,0) ) **(1/2)

print(euclid_distance(x,y))
```

```
10.583005244258363
```

0.4 4. Terdapat dictionary employee berisi nama dan umur pegawai, lakukan reduce untuk mengetahui berapa jumlah pegawai yang berumur > 25 tahun !

[18]:
```
employee = {
    'Nagao' : 35,
    'Ishii' : 30,
    'Kazutomo' : 20,
    'Saito' : 25,
    'Hidemi' : 29
    }

def emp_agemorethan(dictionary, age):
    return r(lambda a,b: a + 1 if b[1] > age else a, dictionary.items(), 0 )

emp_agemorethan(employee,  25)
```

[18]:
```
3
```

0.5 4. Buatlah deret fibonacci menggunakan higher order function!

[19]:
```
fibo_seq = lambda n: r(lambda x, _: x + [x[-1] + x[-2]], range(n-2), [0,1])
print(fibo_seq(10))
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Chapter 11

May 16, 2022

0.1 Buat sebuah program untuk membuat deret fibonacci dari 0 hingga N dengan menggunakan fungsi non-rekursif dan rekursif!

Bandingkan keduanya jika nilai N = 500, Manakah yang lebih baik? Jelaskan!

Rekursif

[4]:
```
def fibo_rec(N):
    if N <= 1:
        return N
    else:
        return(fibo_rec(N-1) + fibo_rec(N-2))
def fibonacci_seq(N):
    return [fibo_rec(i) for i in range(1,N+1)]
```

[14]:
```
fibonacci_seq(10)
```

[14]:
```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

0.1.1 fibonacci_seq(500) Loadingnya terlalu lama, sampai saya kira laptop saya rusak

Non-Rekursif

[3]:
```
from functools import reduce as r

fibo_seq = lambda n: r(lambda x, _: x + [x[-1] + x[-2]], range(n-2), [0,1])
print(fibo_seq(500)) #Waktu yang dibutuhkan sangat singkat (tidak sampai 1␣
 ↪detik)
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584,
4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229,
832040, 1346269, 2178309, 3524578, 5702887, 9227465, 14930352, 24157817,
39088169, 63245986, 102334155, 165580141, 267914296, 433494437, 701408733,
1134903170, 1836311903, 2971215073, 4807526976, 7778742049, 12586269025,
20365011074, 32951280099, 53316291173, 86267571272, 139583862445, 225851433717,
365435296162, 591286729879, 956722026041, 1548008755920, 2504730781961,
4052739537881, 6557470319842, 10610209857723, 17167680177565, 27777890035288,
44945570212853, 72723460248141, 117669030460994, 190392490709135,
308061521170129, 498454011879264, 806515533049393, 1304969544928657,
2111485077978050, 3416454622906707, 5527939700884757, 8944394323791464,
```

14472334024676221, 23416728348467685, 37889062373143906, 61305790721611591,
99194853094755497, 160500643816367088, 259695496911122585, 420196140727489673,
679891637638612258, 1100087778366101931, 1779979416004714189,
2880067194370816120, 4660046610375530309, 7540113804746346429,
12200160415121876738, 19740274219868223167, 31940434634990099905,
51680708854858323072, 83621143489848422977, 135301852344706746049,
218922995834555169026, 354224848179261915075, 573147844013817084101,
927372692193078999176, 1500520536206896083277, 2427893228399975082453,
3928413764606871165730, 6356306993006846248183, 10284720757613717413913,
16641027750620563662096, 26925748508234281076009, 43566776258854844738105,
70492524767089125814114, 114059301025943970552219, 184551825793033096366333,
298611126818977066918552, 483162952612010163284885, 781774079430987230203437,
1264937032042997393488322, 2046711111473984623691759, 331164814351698201718008l,
53583592549909666408718 40, 86700073985079486580519 21,
14028366653498915298923761, 22698374052006863956975682,
36726740705505779255899443, 59425114757512643212875125,
96151855463018422468774568, 15557697022053106568164969 3,
25172882568354948815042426 1, 40730579590408055383207395 4,
65903462158763004198249821 5, 10663404174917105958145721 69,
17253750390793406377970703 84, 27917154565710512336116425 53,
45170904956503918714087129 37, 73088059522214431050203554 90,
11825896447871834976429068427, 19134702400093278081449423917,
30960598847965113057878492344, 50095301248058391139327916261,
81055900096023504197206408605, 13115120134408189533653432 4866,
21220710144010539953374073 3471, 34335830278418729487027505 8337,
55556540422429269440401579 1808, 89892370700847998927429085 0145,
14544891112327726836783066419 53, 23534128182412526729525974 92098,
38079019294740253563090413 4051, 61613147477152780295835016 26149,
99692166771893033862144057 60200, 16130531424904581415797907 386349,
26099748102093884802012313 146549, 42230279526998466217810220 532898,
68330027629092351019822533 679447, 11056030715609081723763275 4212345,
17889033478518316825745528 7891792, 28945064194127398549508804 2104137,
46834097672645715375254333 29995929, 75779161866773113924763137 2100066,
12261325953941882930001747 02095995, 19839242140619194322478060 74196061,
32100568094561077252479807 76292056, 51939810235180271574957868 50488117,
84040378329741348824743767 626780173, 13598018856492162040239554 477268290,
22002056689466296922983322 104048463, 35600075545958458963222876 581316753,
57602132235424755886206198 685365216, 93202207781383214849429075 266681969,
15080434001680797073563527 3952047185, 24400654779819118558506434 9218729154,
39481088781499915632069962 3170776339, 63881743561319034190576397 2389505493,
10336283234281894982264635 95560281832, 16724457590413798401322275 67949787325,
27060740824695693383586911 63510069157, 43785198415109491784909187 31459856482,
70845939239805185168496098 94969925639, 11463113765491467695340528 626429782121,
18547707689471986212190138 521399707760, 30010821454963453907530667 147829489881,
48558529144435440119720805 669229197641, 78569350599398894027251472 817058687522,
12712787974383433414697227 8486287885163,
20569723034323322817422375 1303346572685,
33282511008706756232119602 9789634457848,

538522340430300790495419781092981030533,
871347450517368352816615810882615488381,
140986979094766914331203559197559651891 4,
228121724146503749612865140285821200729 5,
369108703241270663944068699483380852620 9,
597230427387774413556933839769202053350 4,
966339130629045077501002539252582905971 3,
156356955801681949105793637902178495932 17,
252990868864586456855893891827436786529 30,
409347824666268405961687529729615282461 47,
662338693530854862817581421557052068990 77,
107168651819712326877926895128666735145 224,
173402521172797813159685037284371942044 301,
280571172992510140037611932413038677189 525,
453973694165307953197296969697410619233 826,
734544867157818093234908902110449296423 351,
118851856132312604643220587180785991565 7177,
192306342848094413966711477391830921208 0528,
311158198980407018609932064572616912773 7705,
503464541828501432576643541964447833981 8233,
814622740808908451186575606537064746755 5938,
131808728263740988376321914850151258073 74171,
213271002344631833494979475503857732749 30109,
345079730608372821871301390354008990823 04280,
558350732953004655366280865857866723572 34389,
903430463561377477237582256211875714395 38669,
146178119651438213260386312206974243796 773058,
236521166007575960984144537828161815236 311727,
382699285659014174244530850035136059033 084785,
619220451666590135228675387863297874269 396512,
100191973732560430947320623789843393330 2481297,
162114018899219444470188162576173180757 1877809,
262305992631779875417508786366016574087 4359106,
424420011530999319887696948942189754844 6236915,
686726004162779195305205735308206328932 0596021,
111114601569377851519290268425039608377 66832936,
179787201985655771049810841955860241270 87428957,
290901803555033622569101110380899849648 54261893,
470689005540689393618911952336760090919 41690850,
761590809095723016188013062717659940567 95952743,
123227981463641240980692501505442003148 737643593,
199387062373213542599493807777207997205 533596336,
322615043836854783580186309282650000354 271239929,
522002106210068326179680117059857997559 804836265,
844617150046923109759866426342507997914 076076194,
136661925625699143593954654340236599547 3880912459,
221123640630391454569941296974487399338 7956988653,
357785566256090598163895951314723998886 1837901112,

57890920688648205273383724828921139822497948897 65,
93669477314257265089773319960393539711116327908 77,
15156039800290547036315704478931467953361427680642,
24522987531716273545293036474970821924473060471519,
39679027332006820581608740953902289877834488152161,
64202014863723094126901777428873111802307548623680,
10388104219572991470851051838277540168014203677584 1,
16808305705945300883541229581164851348244958539952 1,
27196409925518292354392281419442391516259162217536 2,
44004715631463593237933511000607242864504120757488 3,
71201125556981885592325792420049634380763282975024 5,
11520584118844547883025930342065687724526740373251 28,
18640696674542736442258509584070651162603068670753 73,
30161280793387284325284439926136338887129809044005 01,
48801977467930020767542949510206990049732877714758 74,
78963258261317305092827389436343328936862686758763 75,
12776523572924732586037033894655031898659556447352 249,
20672849399056463095319772838289364792345825123228 624,
33449372971981195681356806732944396691005381570580 873,
54122222371037658776676579571233761483351206693809 497,
87571595343018854458033386304178158174356588264390 370,
14169381771405651323470996587541191965770779495819 9867,
22926541305707536769274335217959007783206438322259 0237,
37095923077113188092745331805500199748977217818079 0104,
60022464382820724862019667023459207532183656140338 0341,
97118387459933912954764998828959407281160873958417 0445,
15714085184275463781678466585241861481334453009875 50786,
25425923930268855077154966468137802209450540405717 21231,
41140009114544318858833433053379663690784993415592 72017,
66565933044813173935988399521517465900235533821309 93248,
10770594215935749279482183257489712959102052723690 265265,
17427187520417066673081023209641459549125606105821 258513,
28197781736352815952563206467131172508227658829511 523778,
45624969256769882625644229676772632057353264935332 782291,
73822750993122698578207436143903804565580923764844 306069,
11944772024989258120385166582067643662293418870017 7088360,
19327047124301527978205910196458024118851511246502 1394429,
31271819149290786098591076778525667781144930116519 8482789,
50598866273592314076796986974983691899996441363021 9877218,
81870685422883100175388063753509359681141371479541 8360007,
13246955169647541425218505072849305158113781284256 38237225,
21434023711935851442757311448200241126227918432210 56597232,
34680978881583392867975816521049546284341699716466 94834457,
56115002593519244310733127969249787410569618148677 51431689,
90795981475102637178708944490299333694911317865144 46266146,
14691098406862188148944207245954912110548093601382 197697835,
23770696554372451866815101694984845480039225387896 643963981,
38461794961234640015759308940939757590587318989278 841661816,

622324915156070918825744106359246030706265443771754856257 97,
100694286476841731898333719576864360661213863366454327287613,
162926777992448823780908130212788963731840407743629812913410,
263621064469290555679241849789653324393054271110084140201023,
426547842461739379460149980002442288124894678853713953114433,
690168906931029935139391829792095612517948949963798093315456,
111671674939276931459954180979453790064284362881751204642988 9,
180688565632379924973893363958663351316079257878131013974534 5,
292360240571656856433847544938117141380363620759882218617523 4,
473048806204036781407740908896780492696442878638013232592057 9,
765409046775693637841588453834897634076806499397895451209581 3,
123845785297973041924932936273167812677324937803590868380163 92,
200386689975542405709091781656657576085005587743380413501122 05,
324232475273515447634024717929825388762330525546971281881285 97,
524619165249057853343116499586482964847336113290351695382408 02,
848851640522573300977141217516308353609666638837322977263693 99,
137347080577163115432025771710279131845700275212767467264610 201,
222232244629420445529739893461909967206666939096499764990979 600,
359579325206583560961765665172189099052367214309267232255589 801,
581811569836004006491505558634099066259034153405766997246569 401,
941390895042587567453271223806288165311401367715034229502159 202,
152320246487859157394477678244038723157043552112080122674872 8603,
246459335992117914139804800624667539688183688883583545625088 7805,
398779582479977071534282478868706262845227240995663668299961 6408,
645238918472094985674087279493373802533410929879247213925050 4213,
104401850095207205720836975836208006537863817087491088222501 20621,
168925741942416704288245703785545386791204910075415809615006 24834,
273327592037623910009082679621753393329068727162906897837507 45455,
442253333980040614297328383407298780120273637238322707452513 70289,
715580926017664524306411063029052173449342364401229605290021 15744,
115783425999770513860373944643635095356961600163955231274253 486033,
187341518601536966291015050946540312701895836604078191803255 601777,
303124944601307480151388995590175408058857436768033423077509 087810,
490466463202844446442404046536715720760753273372111614880764 689587,
793591407804151926593793042126891128819610710140145037958273 777397,
128405787100699637303619708866360684958036398351225665283903 8466984,
207764927881114829962999013079049797839997469365240169079731 2244381,
336170714981814467266618721945410482798033867716465834363635 0711365,
543935642862929297229617735024460280638031337081706003443366 2955746,
880106357844743764496236456969870763436065204798171837807001 3667111,
142404200070767306172585419199433104407409654187987784125036 76622857,
230414835855241682622209064896420180751016174667804967905736 90289968,
372819035926008988794794484095853285158425828855792752030773 66912825,
603233871781250671417003548992273465909442003523597719936510 57202793,
976052907707259660211798033088126751067867832379390471967284 24115618,
157928677948851033162880158208040021697730983590298819190379 481318411,
255533968719576999184059961516852696804517766828237866387107 905434029,
413462646668428032346940119724892718502248750418536685577487 386752440,

668996615388005031531000081241745415306766517246774551964595292186469,
108245926205643306387794020096663813380901526766531123754208267893890 9,
175145587744443809540894028220838354911578178491208578950667797112537 8,
283391513950087115928688048317502168292479705257739702704876065006428 7,
458537101694530925469582076538340523204057883748948281655543862118966 5,
741928615644618041398270124855842691496537589006687984360419927125395 2,
120046571733914896686785220139418321470059547275563626601596378924436 17,
194239433298376700826612232625002590619713306176232425037638371636975 69,
314286005032291597513397452764420912089772853451796051639234750561411 86,
508525438330668298340009685389423502709486159628028476676873122198387 55,
822811443362959895853407138153844414799259013079824528316107872759799 41,
133133688169362819419341682354326791750874517270785300499298099495818 696,
215414832505658809004682396169711233230800418578767753330908886771798 637,
348548520675021628424024078524038024981674935849553053830206986267617 333,
563963353180680437428706474693749258212475354428320807161115873039415 970,
912511873855702065852730553217787283194150290277873860991322859307033 303,
147647522703638250328143702791153654140662564470619466815243873234644 9273,
238898710089208456913416758112932382460077593498406852914376159165348 2576,
386546232792846707241560460904086036600740157969026319729620032399993 1849,
625444942882055164154977219017018419060817751467433172643996191565341 4425,
101199117567490187139653767992110445566155790943645949237361622396533 46274,
163743611855695703555151489893812287472237566090389266501761241553067 60699,
264942729423185890694805257885922733038393357034035215739122863949601 06973,
428686341278881594249956747779735020510630923124424482240884105502668 67672,
693629070702067484944762005665657753549024280158459697980006969452269 74645,
112231541198094907919471875344539277405965520328288418022089107495493 842317,
181594448268301656413948075911105052760867948344134387820089804440720 816962,
293825989466396564333419951255644330166833468672422805842178911936214 659279,
475420437734698220747368027166749382927701417016557193662268716376935 476241,
769246427201094785080787978422393713094534885688979999504447628313150 135520,
124466686493579300582815600558914309602223630270553719316671634469008 5611761,
201391329213688779090894398401153680911677118839451719267116397300323 5747281,
325858015707268079673709998960067990513900749110005438583788031769332 1359042,
527249344920956858764604397361221671425577867949457157850904429069655 7106323,
853107360628224938438314396321289661939478617059462596434692460838987 8465365,
138035670554918179720291879368251133336505648500891975428559688990864 35571688,
223346406617740673564123319000380099530453510206838235072028935074763 14037053,
361382077172658853284415198368631232866959158707730210500588624065627 49608741,
584728483790399526848538517369011332397412668914568445572617559140390 63645794,
946110560963058380132953715737642565264371827622298656073206183206018 13254535,
153083904475345790698149223310665389766178449653686710164582374234640 876900329,
247694960571651628711444594884429646292615632415916575771902992555242 690154864,
400778865046997419409593818195095036058794082069603285936485366789883 567055193,
648473825618649048121038413079524682351409714485519861708388359345126 257210057,
104925269066564646753063223127461971841020379655512314764487372613500 9824265250,
169772651628429551565167064435414440076161351104064300935326208548013 6081475307,
274697920694994198318230287562876411917181730759576615699813581161514 5905740557,
444470572323423749883397351998290851993343081863640916635139789709528 1987215864,

719168493018417948201627639561167263910524812623217532334953370871042789295642 1,
116363906534184169808502499155945811590386789448685844897009316058057098801722 85
, 188280755836025964628665263112062537981439270711007598130504653145161377731287
06, 304644662370210134437167762268008349571826060159693443027513969203218476533 00991, 492925418206236099065833025380070887553265330870701041158018622348379854 26429697, 797570080576446233503000787648079237125091391030394484185532591551598330
79730688, 129049549878268233256883381302815012467835672190109552534355121389997 818506160385, 208806557935912856607183460067622936180344811293149000952908380545 157651585891073, 337856107814181089864066841370437948648180483483258553487263501 935155470092051458, 546662665750093946471250301438060884828525294776407554440171 82480313121677942531, 884518773564275036335317142808498833476705778259666107927 4353844154685917699939 89, 143118143931436898280656744424655971830523107303607366 2367607266895781713447936520, 231570021287864401914188458705505855178193685129573 977029504265131125030521793050 9, 374688165219301300194845203130161827008716792 43181343266264991820703201866586 7029, 606258186507165702109033661835667682186910 4775627553202957692569518282323883797538, 980946351726467002303878864965829509195 627269995936663562034248772531434254966456 7, 158720453823363270441291252680149 71913825377475586919838578035057243596666433462105, 256815088996009970671679139176 732670057816501755462864741983775449689110089831266 72, 415535542819373241112970 3918568823891960702765113320631277641260221250767541658877 7, 67235063181538321178 464953103361505925388677826679492786974790147181418684399715449, 108788617463475 645289761992289049744844995705477812699099751202749393926359816304226, 176023680 645013966468226945392411250770384383304492191886725992896575345044216019675, 284 812298108489611757988937681460995615380088782304890986477195645969271404032323901, 460835978753503578226215883073872246385764472086797082873203188542544616448248343576, 745648276861993189984204820755333242001144560869101973859680384188513887852280667477, 120648425561549676821042070382920548838690903295589905673288357273105850430052901105 3, 195213253247748995819462552458453873038805359382500103059256395691957239215280967 8530, 315861678809298672640504622841374421877496262678090008732544752965063089645333868958 3, 511074932057047668459967175299828294916301622060590111791801148657020328860614836811 3, 826936610866346341100471798141202716793797884738680120524345901622083418505948705769 6, 133801154292339400956043897344103101171009950679927023231614705027910374736656354258 09, 216494815378974035066091077158223372850389739153795035284049295190118716587251224835 05, 35029596967131343602213497450232647402139968983372205851566400021802909132390757909314, 566790785050287471088226051660549846871789428987517093799713295408147807911158803928 19, 917086754721600907110361026162876320893189118821239152315377295626176899235066383021 33, 148387753977188837819858707782342616776497854780875624611509059103432470714622518694952, 240096429449348928530894810398630248865816766662999539843046788660501606381291569970 85, 388484183426537766350753518180972865642314621443875164454555847769482631352751675692037, 628580612875886694881648328579603114508131388106874704297602636435532791990880832689122, 101706479630242446123240184676057598015044600955074986875215848420501542334363250838115 9, 164564540917831115611405017534017909465857739765762457304976112064054821533451334107028 1, 26627102054807356173464520221007550748090234072083744418019196048455636386781458494514 40, 430835561465904677346050219744093416946760080486599901485168072548611185401265919052 1721, 697106582013978239080695421954168924427662421207437345665360033033167549269080503997316 1, 112794214347988291642674564169826234137442250169403724715052810 5

581778734670346423049488 2, 18250487254938611555074410636524312658020849229014745
928158881386149462839394269270468043, 29529908689737440719341867053506936071765 0
742459551183996641624419672501860973350096292 5, 47780395944676052274416277690 03
124872978592347496986432782304382811671302549200277143096 8, 77310304634413492993
758144743538184801550997720924982727487206270083963211589736272393893, 125090700
579089545268174422433569433531336921195894847055310250098200676237081739043824861, 20240100521350303826193256717710761833288791891681982978279745636828463944867
1475316218754, 32749170579259258353010698961067705186422484011271467683810770646
648531568575321436004361 5, 52989271100609562179203955678778467019711275902953450
662090516283476995513442468967626236 9, 85738441679868820532214654639846172206133
759914224918345901286930125527082017790403630598 4, 13872771278047838271141861031
862463922584503581717836900799180321360252259546025937125683 53, 2244661544603472
032436332649584708114319787957314032873538930901437280496774780497748874337, 363
193867240825585955051875277095450657823831548581656361884893357330572272938309 14
61442690, 58766002170117278919868514023556626208980262727998494371577798350105 86
219504163589210317027, 95085388894199837515373701551266171274762645882856660007 7
662876858389194223354668067175971 7, 15385139106431711643524221557482279748374290
861085515437934408603594478161737710269882076744, 24893677995851695395061591712 6
088968758505554493711814387110373721783701039712569505538364 61, 4027881710228340
703858581327009117662422484631045669687664544597577284826570896722043591320 5, 65
172495098135102433647404982700073500075401759827878315356483347951218369680224 17
0989749666, 10545131220041850947223321825279125012430024807028457519200192932372
406663538919139142566287 1, 17062380729855361190588062323549132362437564983011245
350735841267167528500506941556241541253 7, 27607511949897212137811384148828257374
867589790039702869936034199539935164045860695384107540 8, 44669892679752573328399
446472377389737305154773050948220671875466707463664552802251625648794 5, 72277404
629649785466210830621205647112172744563090651090607909666247398828598662947009 75
63353, 11694729730940235879461027709358303684947789933614159931127978513295486 24
931514651986354051298, 18922470193905214426082110771478868396165064389923225040 1
887694799202261321750128145645161465 1, 30617199924845450305543138480837172081112
854323537384971316747993215712381490159334428056659 49, 49539670118750664731625 24
925231604047727791871346061001150551747313593851366517214899257280600, 801568700
435961150371683877331532125583907730369979949828222654663516508951553314834206 29
46549, 12969654016234677976879363698546925303566869175045860499432778293948758 94
088205036324132022714 9, 20985341020594289480596202471862246559405946478745659997
715004840583924030397583511583383173698, 33954995036828967457475566170409171862 9
728156537915204971477831345326829712796338748247034008 47, 54940336057423256938 07
176864227141842237876213253718049486278797511660700167721738640808657454 5, 88895
331094252224395547334812680590285351577786328700992010571109649289972956851261 23
2789975392, 14383566715167548133361910345495200870773033991886588148687335908476
589697463406864764087654993 7, 23273099824592770572916643826763259899308191770519
458247888393019441518694759091990887366652532 9, 37656666539760318706278554172258
460770081225762406046396575728927918108392222498855651454307526 6, 60929766364353
089279195197999021720669389417532925504644464121947359627086981590846538820960 05
95, 98586432904113407985473752171280181439470643295331551041039850875277735479 20
408970219027526758 61, 15951619926846649726466895017030190210886006082825705568 55
039728226373625661856805487290962276456, 25810263217257990525014270234158208354 8
330704123588606726543823697915098045389770250919371495231 7, 41761883144104640251

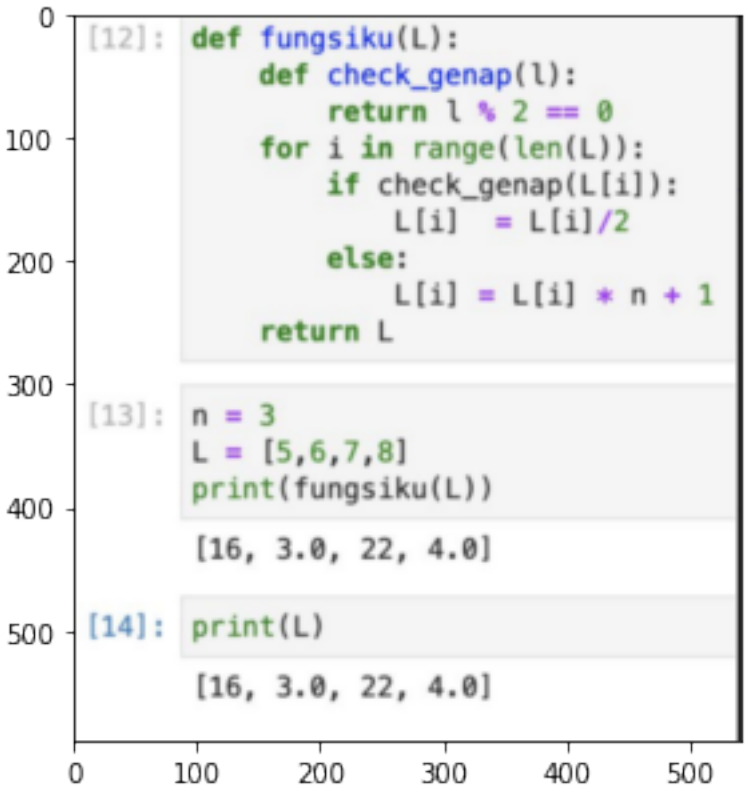
481165251188398565719076495184566241204779652055246061157545079964846772287 73, 6757214636136263077649543548534660692055214690754342691385916202184675586569652210505678392181090, 109334029505467271027976600736535005486271223402727993155063941673902001926854067185021630694098631, 176906175866829901804472036221881612406823370310271420068923103695748757792550589290078414615909532, 28624020537229717283244863695841661789309459371299941322398704536965075971940465647510004531000816, 463146381239127074636920673180298230299917964023270833292910149065399517511955245765178459925917693, 74938658661142424746936931013871484819301255773627024651689719443505027723135990224027850523592585, 12125329678505513221062899833190130784929305217595410798098073435004497947433151480054569651618435 4, 196191955446197556957565929345772792668594307949581132632670453793550007197467505024573547039776939, 31744525223125268916819492767767410051788736012553524061365118814359498667179901982511924355596129 3, 513637207677450246125760857023446893186481668075116373246321641937144993869266524849692790595738232, 83108245990870293529395578470112099370436902820065161385997283008073998054106554467481203415169952 5, 13447196675861531814197166417245678868908506962757679871062944720178849744103320695245048247474377 57, 217580212749485611671367242642568888059521972447641960096626730209862495495139761419931685889913728 2, 352052179508100929813338906815025676748607042075218758807256177411650992936172968372382168364657503 9, 569632392257586541484706149457594564808129014522860718903882907621513488431312729792313854254571232 1, 921684571765687471298045056272620241556736056598079477711139085033164481367485698164696022619228736 0, 149131696402327401278275120573021480636486507112094019661502199265467796979879842795700987687379996 81, 241300153578896148408079626200283504792160112771901967432616107768784245116628412612170589949302870 41, 390431849981223549686354746773304985428646619883995987094118307034252042096508255407871577636682867 22, 631732003560119698094434372973588490220806732655897954526734414803036287213136668020042167585985737 63, 102216385354134324778078911974689347564945335253989394162085272183728832930964492342791374522266860485, 165389585710146294587522349272048196587026008519579189614758713664032461652278159144795591280865434248, 267605971064280619365601261246737544151971343773568583776843985847761294583242651487586965803132294733, 432995556774426913953123610518785740738997352293147773391602699511793756235520810632382557083997728981, 700601527838707533318724871765523284890968696066716357168446685359555050818763462119969522887130023714, 113359708461313444727184848228430902562996604835986413056004938487134880705428427275235207997112775 2695, 183419861245184198059057335404983231052093474442658048772849607023090385787304773487232160285825777 6409, 296779569706497642786242183633414133615090079278644461828854545510225266492733200762467368282938 5529104, 48019943095168184084529951903839736466718355372130251060170415253331565228003797424969952856876433055 13, 776979000658179483631541702671811498282273632999946972430558698043540918772771175012166896851702883 4617, 125717843160986132447684122171020886294945718672124948303226285057685657105280914926186642542046721 40130, 203415743226804080810838292438202036123173081972119645546282154862039748982558032427403332227217009747 47, 32913358638779021325852241460922292241811880064424459384950843991972540608783894735358997476926373 114877, 532549329614594294069360707047424958541291882616364239395790594781765155070396979780993306996480 74089624, 861682916002384507327883121656647880959410683260608833245299034701490561158235927134583281765744 47204501]

Chapter 12

May 16, 2022

0.1 1. Ubah fungsiku menjadi pure function!

[9]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('12_no 1.png')

fig, ax = plt.subplots(figsize=(10, 5))
plt.imshow(img)
plt.show()
```



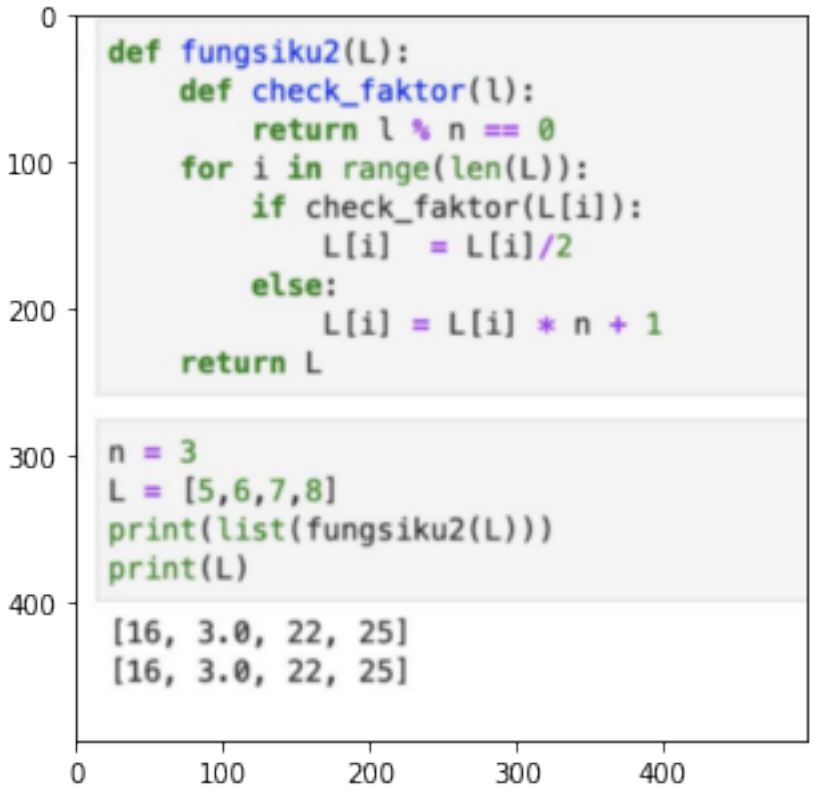
[12]:
```
def fungsiku(L):
    def check_genap(l):
```

```
        return l % 2 == 0
    return list(map(lambda l: l/2 if check_genap(l) else l * n + 1, L))

n = 3
X = [5,6,7,8]
print(fungsiku(X))
print(X)
```

```
[16, 3.0, 22, 4.0]
[5, 6, 7, 8]
```

0.2 2. Ubah fungsiku2 menjadi pure function!

[10]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('12_no 2.png')

fig, ax = plt.subplots(figsize=(10, 5))
plt.imshow(img)
plt.show()
```



[11]:
```
def fungsiku2(L):
    def check_faktor(l):
        return l % n == 0
    return list(map(lambda l: l/2 if check_faktor(l) else l * n + 1, L))

n = 3
L = [5,6,7,8]
print(fungsiku(L))
print(L)
```

```
[16, 3.0, 22, 4.0]
[5, 6, 7, 8]
```

0.3 3. Apakah isi dalam tupel tup ada yang dapat diubah?

[14]:
```
tup = ([3, 4, 5], 'myname')
tup
```

[14]:
```
([3, 4, 5], 'myname')
```

Ada, yaitu elemen list dalam tuple

[17]:
```
tup[0][2] = 8
tup
```

[17]:
```
([3, 4, 8], 'myname')
```

Chapter 13

May 16, 2022

0.1 1. Latihan 1

[1]:
```
Addku = lambda x:x+10
Powku = lambda x:x**2
Kurku = lambda x:x-2*x
```

a. Buatlah fungsi komposisi menggunakan 3 fungsi diatas yang melakukan hal sebagai berikut secara berurut:

1. Menjumlahkan input dengan nilai 10
2. Mengurangi input dengan 2 kali input nya
3. Mengeluarkan nilai kuadrat dari input nya

[2]:
```
#compost_1 =compost_func(Powku,Addku)
#compost_2 =compost_func(Kurku, compost_1)
compost_func =lambda f,g: lambda x: f(g(x))
comp_func_f = compost_func(Kurku, compost_func(Powku, Addku))


print(comp_func_f(10))
```

```
-400
```

B. Buatlah fungsi invers nya!

[3]:
```
inv_add = lambda x: x-10
inv_pow = lambda x: x**(0.5)
inv_kur = lambda x: -1 *  x
```

[4]:
```
inv_compost_func =lambda f,g: lambda x: f(g(x))
inv_comp_func_f = inv_compost_func(inv_add, compost_func(inv_pow, inv_kur))

print(inv_comp_func_f(-400))
```

```
10.0
```

0.2 2. Latihan Penentuan UKT Mahasiswa

[23]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('chp13=pbf.png')

fig, ax = plt.subplots(figsize=(15, 10))
plt.imshow(img)
plt.show()
```

[5]:
```
from functools import reduce as r
```

[6]:
```
mhs = [3, [120000, 75000, 50000], 5.5*10**6, False]
```

Tanpa Fungsi Komposisi

[7]:
```
def skor_1(X):
    return 1 if X[0] >= 5 else 5-X[0]
def skor_2(X):
    def rata2(X):
        return sum(X[1])/len(X[1])
    def skor(X):
        return 3 if rata2(X) >= 100000 else 2 if rata2(X) >= 50000 else 1
    return skor(X)
def skor_3(X):
    return 7 if X[2] > 10*10**6 else 6 if X[2] > 8*10**6 else 5 if X[2] >␣
 ↪6*10**6 else 4 if X[2] > 4*10**6 else 3 if X[2] > 3*10**6 else 2
def skor_4(X= True):
    return 1 if X[3] == True else 5
def skor(X):
    return 0.2 * skor_1(X) + 0.3 * skor_2(X) + 0.2 * skor_3(X) + 0.3 * skor_4(X)
def total_UKT(X):
    return 75*10**4 + skor(X) * 5*10**5
```

[8]:
```
total_UKT(mhs)
```

[8]:
```
2400000.0
```

Dengan fungsi Komposisi

[9]:
```
compose_func = lambda *funcs: r(lambda f,g : lambda x:f(g(x)), reversed(funcs),␣
 ↪lambda x:x)
```

[10]:
```
#Jumlah tanggungan
def skor1(jtg):
    return 1 if jtg >= 5 else 5-jtg
#Token Listrik
def skor2(X):
    def rata2(X):
        return sum(X)/len(X)
    def l_cond_1(X):
        return [X, [X>100000]]
    def l_cond_2(X):
        return [X[0], X[1]+[X[0] >= 50000]]
    def to_score2(X):
        return r( lambda a,b: a+(1 if b==True else 0), X[1], 1)
    compose_skor2 = compose_func(rata2, l_cond_1, l_cond_2, to_score2)
    return compose_skor2(X)
#Ketentuan Gaji
def skor3(gaji):
    def con_1(X):
        return [X[0], 1, X[2], [ X[0] > X[2][X[1] ] ] ]
```

```
    def con_2_to_n(X):
        return [X[0], X[1] + 1, X[2], X[3] + [X[0] > X[2][X[1] ] ] ]
    def to_score(X):
        return r(lambda a,b: a + (1 if b==True else 0), X[-1], 2)
    def prep(gaji):
        return [gaji, 0 , list(map(lambda x:x*10**6, list(range(10,3,-2)) +␣
 ↪[3]) )]
    comp_skor3 = compose_func(prep, con_1, *(con_2_to_n for i in range(4)),␣
 ↪to_score)
    return comp_skor3(gaji)
```

[11]:
```
#Ket KIP-K
def skor4(X=True):
    return 1 if X else 5
```

[12]:
```
def combineskor(X):
    return X + [map( lambda f,x: f(x), X[1], X[0])]
def bobot(X):
    return r(lambda a,b: a+b, map( lambda x,y:x*y, X[-1], [0.2,0.3,0.2,0.3]))
def to_UKT(X):
    return 750000+X*500000
```

[13]:
```
datas = [mhs, [skor1, skor2, skor3, skor4]]
compose_final = compose_func(combineskor, bobot, to_UKT)
compose_final(datas)
```

[13]:
```
2400000.0
```

0.3 3. Turunan Polinom

[14]:
```
def splt(dat):
    return dat.replace(' ','').replace('-', '+-').split('+')
def chdepan(dat):
    return dat[1:] if dat[0] == '' else dat
def eqkan(dat):
    return map(lambda x: x if '^' in x else x + '^1' if 'x' in x else␣
 ↪x+'x^0',dat)
def toarr2d(dat):
    return r(lambda a,b: a + [ [ float(hurf) for hurf in b.split('x^')]], dat,␣
 ↪[])
def sortdesc(dat):
    return sorted(dat, key=lambda x:x[1], reverse=True)
def calctur(dat):
    return map(lambda x: [0,0] if x[1] == 0 else [ x[1] * x[0], x[1]-1], dat)
def tostr(dat):
```

```
    return map(lambda x: '0' if x[0] == 0 else str(x[0]) if x[1] ==0 else␣
 ↪str(x[0]) + 'x' if x[1] == 1 else str(x[0]) + 'x^' + str(x[1]), dat)
def prettykan(dat):
    return r(lambda a,b : a+ '+' + b if b!='0' else a, dat, '')
def prettysign(dat):
    return dat.replace('+-', ' -').replace('+', ' + ')
```

[15]:
```
dat = '-3x^5 + 2x^2 - 4x + 5'
fss = (splt, chdepan, eqkan, toarr2d, sortdesc, calctur, tostr, prettykan,␣
 ↪prettysign)
turunan = compose_func(*fss)
```

[21]:
```
print(turunan(dat))
```

```
-15.0x^4.0 + 4.0x -4.0
```

0.4 4. Buatlah fungsi untuk menghitung biaya yang harus dibayar customer pada suatu e-commerce menggunakan higher order function.

Buatlah decorator untuk mengeluarkan harga sebelum pajak dan sesudah pajak (pajak = 11%) ! Gunakan decorator untuk menambahkan perhitungan waktu eksekusi!

[17]:
```
keranjang = [
    {'jumlah_barang' : 5, 'harga' : 10},
    {'jumlah_barang' : 7, 'harga' : 20},
    {'jumlah_barang' : 20, 'harga' : 4.5}
]
```

[18]:
```
def pajak(func):
    def inner(*args, **kwargs):
        res = func(*args, **kwargs)
        print('subtotal : ', res)
        print('pajak:', res* 0.11)
        print('total:', res + res*0.11)
        return res
    return inner

def calc_time(fn):
    from time import perf_counter
    def inner(*args, **kwargs):
        start_time = perf_counter()
        to_execute = fn(*args, **kwargs)
        end_time = perf_counter()
        execution_time = end_time - start_time
        print('Execution Time : {0:.8f}s'.format(execution_time))
        return to_execute
    return inner
```

[19]:
```
@calc_time
@pajak
def total_pembayaran(keranjang):
    return r( lambda a,b: a+ (b['jumlah_barang'] * b['harga']), keranjang, 0) *␣
↪1000
```

[20]:
```
total_pembayaran(keranjang)
```

```
subtotal :  280000.0
pajak: 30800.0
total: 310800.0
Execution Time : 0.00015180s
```

[20]:
```
280000.0
```

Chapter 14

May 16, 2022

0.1 1. Buatlah fungsi untuk menampilkan bilangan fibonacci ke-101 dengan menggunakan konsep lazy evaluation!

[10]:
```
def fibo_gen():
    yield 0
    a, b = 0, 1
    while True:
        yield b
        a,b = b, a+b
```

[18]:
```
from itertools import islice
list(islice(fibo_gen(), 101,102))
```

[18]:
```
[573147844013817084101]
```

JURNAL MODUL

Jurnal Modul 1

May 16, 2022

Seorang mahasiswa sains data ingin menyewa buku dari sebuah startup yang menyediakan layanan sewa buku. Startup tersebut memiliki ketentuan sewa dengan aturan sebagai berikut: - Harga sewa buku berbeda-beda sesuai dengan kategorinya - Harga sewa buku dihitung berdasarkan jumlah halaman nya - Harga sewa buku dihitung per hari nya - Maksimal durasi sewa adalah 26 hari

Startup tersebut masih dalam tahap awal pengembangan, sehingga ingin melakukan uji coba penyewaan 5 kategori buku. Berikut rincian kategori nya:

- Kategori 1 : 100 rupiah per lembar per hari
- Kategori 2 : 200 rupiah per lembar per hari
- Kategori 3 : 250 rupiah per lembar per hari
- Kategori 4 : 300 rupiah per lembar per hari
- Kategori 5 : 500 rupiah per lembar per hari

Startup tersebut memerlukan sebuah program untuk: - menghitung total biaya dari customer - mencatat tanggal awal sewa, dan durasi hari - menampilkan informasi kapan tanggal pengembalian buku dari customer

Format input tanggal adalah yyyy-mm-dd Bantulah startup tersebut membuat program tersebut dengan menggunakan konsep modularisasi!

[1]:
```
def tgl_list(tgl): #memisahkan tanggal. bulan, dan tahun
    return [int(i) for i in tgl_pinjam.split('-')]

def is_kabisat(tgl): #mengecek apakah tahun tersebut kabisat
    return ( tgl_list(tgl)[0] % 100 == 0 and tgl_list(tgl)[0] % 400 == 0 ) or␣
 ↪(tgl_list(tgl)[0] % 100 != 0 and tgl_list(tgl)[0] % 4 == 0)

def monthh(tgl): #menentukan jumlah hari dalam suatu bulan
    return 29 if tgl_list(tgl)[1] == 2 and is_kabisat(tgl) else 28 if␣
 ↪tgl_list(tgl)[1] == 2 and not is_kabisat(tgl) else 30 if tgl_list(tgl)[1] %␣
 ↪2 == 0 and tgl_list(tgl)[1] <= 7 else 31 if tgl_list(tgl)[1] % 2 == 1 and␣
 ↪tgl_list(tgl)[1] <= 7 else 30 if tgl_list(tgl)[1] % 2 == 1 and␣
 ↪tgl_list(tgl)[1] > 7 else 31

def is_newmonth(tgl, durasi): #menentukan apakah terjadi pergantian bulan
    return tgl_list(tgl)[2] + durasi > monthh(tgl)

def is_newyear(tgl, durasi): #menentukan apakah terjadi pergantian tahun
    return tgl_list(tgl)[1] == 12 and is_newmonth(tgl, durasi)
```

```
def tgl_kembali(tgl, durasi): #menentukan tanggal buku harus dikembalikan
    return [tgl_list(tgl)[0] + 1, tgl_list(tgl)[1]+1, (tgl_list(tgl)[2] +␣
 ↪durasi) - monthh(tgl) ] if is_newyear( tgl, durasi) and is_newmonth(tgl,␣
 ↪durasi) else [tgl_list(tgl)[0], tgl_list(tgl)[1]+1, (tgl_list(tgl)[2] +␣
 ↪durasi) - monthh(tgl) ] if is_newmonth(tgl, durasi) else [tgl_list(tgl)[0],␣
 ↪tgl_list(tgl)[1], (tgl_list(tgl)[2] + durasi) ]

def tgl_pengembalian(tgl, durasi): #menggabungkan tanggal, bulan, dan tahun␣
 ↪tanggal pengembalian
    return '-'.join([str(i) for i in tgl_kembali(tgl, durasi)])

def biaya_per_kategori(rincian_buku, rincian_kategori): #menghitung biaya buku␣
 ↪per kategori
    return list(map(lambda x, y: x[1] * rincian_kategori.get(x[0]),␣
 ↪rincian_buku, rincian_kategori))

def biaya_total(rincian_buku, rincian_kategori, durasi): #menghitung biaya total
    return sum(biaya_per_kategori(rincian_buku, rincian_kategori)) * durasi
```

[2]:
```
rincian_kategori = {
    1 : 100,
    2 : 200,
    3 : 250,
    4 : 300,
    5 : 500
    }
```

[8]:
```
tgl_pinjam = input('Masukkan tanggal peminjaman (YYYY-MM-DD): ')
durasi = int(input('Masukkan durasi peminjaman (hari) : '))
```

```
Masukkan tanggal peminjaman (YYYY-MM-DD): 2022-06-15
Masukkan durasi peminjaman (hari) : 25
```

[6]:
```
#[kategori, halaman]
rincian_buku = [[3, 6], [2, 5], [4, 3], [1, 5], [5,10]]
```

[10]:
```
def book_fee(tgl_pinjam, durasi, rincian_buku, rincian_kategori =␣
 ↪rincian_kategori):
    print(f'Biaya yang harus dibayar adalah Rp.{biaya_total(rincian_buku,␣
 ↪rincian_kategori, durasi)} dengan tanggal pengembalian␣
 ↪{tgl_pengembalian(tgl_pinjam, durasi)}  ')
```

[11]:
```
book_fee(tgl_pinjam, durasi, rincian_buku)
```

```
Biaya yang harus dibayar adalah Rp.222500 dengan tanggal pengembalian 2022-7-10
```

Jurnal Modul 2

May 16, 2022

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

[2]:
```
def ulangi_065(a): #Mapping mereturn vektor 1xa dengan seluruh elemennya adalah␣
↪a
    return list(map(lambda x: a, range(a)))
```

[3]:
```
ulangi_065(5)
```

[3]:
```
[5, 5, 5, 5, 5]
```

0.0.2 2. Buatlah deret bilangan sebagai berikut dengan input n sebagai panjang deret:

[4]:
```
def deret(n): #Menghitung deret sebanyak n deret
    return list(map(lambda n: (-1)**n*(1/2**(n+1)), range(n+1)))
```

[5]:
```
deret(5)
```

[5]:
```
[0.5, -0.25, 0.125, -0.0625, 0.03125, -0.015625]
```

0.0.3 3. Jumlahkan deret bilangan tersebut!

[6]:
```
from functools import reduce as r
```

[7]:
```
def add(a,b): #fungsi untuk menambahkan setiap elemen list dengan reduce
    return a+b
r(add, deret(5))
```

[7]:
```
0.328125
```

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

[8]:
```
dna_data = open('DNA.txt', 'r')
dna = dna_data.read() #membaca seluruh isi file .txt
```

[9]:
```
def split_txt(txt, dna): #membagi kumpulan kode DNA sesuai jumlah karakter pola␣
 ↪yang dicari
    return [txt[i:i+len(dna)] for i in range(len(txt))]
def pattern(txt, dna): #memfilter kode DNA yang telah dibagi yang sesuai dengan␣
 ↪pola yang dicari
    return list(filter(lambda x: x == dna, split_txt(txt, dna)))
def counting_pattern(txt, dna): #menghitung jumlah DNA yang sesuai dengan pola␣
 ↪DNA
    return len(pattern(txt, dna))
def count_all_pattern(txt, sequences): #Menghitung jumlah DNA jika pola yang␣
 ↪dicari lebih dari 1
    return map(lambda x: counting_pattern(txt,x), sequences)
```

hitunglah jumlah kemunculan pola berikut pada data tersebut: 1. A 2. AT 3. GGT 4. AAGC 5. AGCTA

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

[11]:
```
print(* count_all_pattern(dna, sequences))
```

```
2112 557 77 22 5
```

==========

[12]:
```
def append_n(dat, i, n): #Menentukan pembagian kode dna
    return dat[i:i+n]
def remap(dat, seq):#membagi kumpulan kode DNA sesuai jumlah karakter pola yang␣
 ↪dicari
    return map(lambda x: append_n(dat, x, len(seq)), range(len(dat) - len(seq)␣
 ↪+ 1))
def count_mer(dat, seq): #menghitung jumlah DNA yang sesuai dengan pola DNA
    return r(lambda a,b : a + (1 if b == seq else 0), remap(dat, seq), 0)
```

[13]:
```
def count_all(dat, sequences): #Menghitung jumlah DNA jika pola yang dicari␣
 ↪lebih dari 1
    return map(lambda x: count_mer(dat,x), sequences)
print(* count_all(dna, sequences))
```

```
2112 557 77 22 5
```

5. Reverse complement dari suatu sequence string DNA memiliki aturan sebagai berikut:

- A adalah komplemen dari T
- C adalah komplemen dari G

[14]:
```
def komplemen(dna): #menukar value sesuai komplemennya
    return list(map(lambda x: 'A' if x == 'T' else('T' if x == 'A' else ('C' if␣
 ↪x == 'G' else ('G' if x == 'C' else x))), list(dna)))
def hasilkomplemen(dna):
    return ''.join(komplemen(dna)) #menggabungkan list menjadi string
```

[15]:
```
dna[:10]
```

[15]:
```
'TGTCTTCCGG'
```

[16]:
```
hasilkomplemen(dna)[:10]
```

[16]:
```
'ACAGAAGGCC'
```

[17]:
```
def komplemen2(x): #mendapatkan value dari key 'x'
    return {'A' : 'T', 'T' : 'A', 'C' : 'G', 'G' : 'C'}.get(x)
def reverse_komplemen(dna): #menukar nilai key dengan value jika memenuhi dict␣
 ↪komplemen2
    return map(lambda x: komplemen2(x), dna)
```

[18]:
```
print(* reverse_komplemen(dna))
```

```
A C A G A A G G C C G A C T C G C C A A G G A T T G G T C G T C T G A C T A T G
A C C A G C T T A T A G C T G C C C G T T C T C G G G A C C C T A A C T A C G C
A A A G T G G T A C G C G C A G A G T C A C G T C C G T C C T T A C G T C T C G
A A T G A A G T T T G A T C A A T G A C C G T T T T T T A T G T T T A A A A A A
G C T A G C T G G A A C T C A A A T A A G T A A T G G C G T G T C A G A A A A T
G G C G T G G A C A A T G G C G T G T A G G C A T T C A A A T G G C G T G C A A
T G G C G T G A T G G A G A G A T A T A A T G G C G T G A A G C A A A T G G C G
T G C G A C T C C T T G C C A A T G G C G T G A A T G G C G T G G T G T T C C A
C G C A C G A G A C A A T A A T G G C G T G G T G G T A A T G G C G T G C G T G
A A A A T A A T G G C G T G G T C C C G T G T C G G T G C A T C C C A T C G C A
G C A A G A G T G A C A T A A C G C C G C T G C C A G C A T T A A A T G G C G T
A A T G G C G T G G T G A G C A A T C G A A T G G C G T G G A T C C C A A C A A
T G G C G T G C T G A A T G G C G T G T C G G C A A T G G C G T G C A C A A T G
A A C T G C G A G A T T G A G G G T G A G T A T A G T C A G A A T A A T G G C G
T G T G A C C C G A A T G G C G T G G G C G T G G A A T T C A T C C G T C A A T
G G C G T G C A T A A T G G C G T G C A T T A A T G G C G T G T G G A C A T T T
C C G T C C C A T T T C A T G T C T G A A T G G C G A A T G G C G T G C C A A C
G T G G T G C T G T T T A G A T T G C A A T C C A T G C A A T G G C G T G C C C
T T T A A T G G C G T G A G G T C C C A A A A T G G C G T G T C T A T A G G T A
A G C C C T T A C A C T G G G G A C C T C A C C T C A A C A C G C T T T C T A T
G C C T C A A A A G T T C C C G T G T G G G T C G A T A C A A T A A T T C G C A
A T G T C A C C G G C G A C G T A G T A C A G T T A C A A G T C C A G T A A G A
G A T A G A A C G A T A C A T G C T T G G G A G C A A T T C T C C C T C A T T C
```

G C T A G A A A A C T G T T T T A G C A T A C G T A C A T C C G C T C C G T T A
C G G C T A A T G T A A C T T G C C G C C C T G A A A A A G C A T A C T C T G T G
G C G C C A A C T T T A T A A A A A A A A T A C G T T C T C G C C C T A A C C C G
C C T T C C T C T G A A T T G C G T C A C G G A T C G T G A C A A T T G A C G C
C G T A C C G G C C T A C C T G A T G G A T A A A A C G T C G A G G T C G C A A
A C T C A A G G T G C A T G A C T G C C T T G T C A G G G C T C T A T C C G G T
A C A C C A G C T A G G G T C A C T C T T T A C T C T G A G C T C T A C G G C C
A T G G C C A T C G T A G T G G T G T A A C G A G G T C A T A C T A T A G T C A
G A A G T G A C A G T C G T T A A T T A C G T C G C T A G A A C T T C T C T C A
A T A A G T A G A G A A T A G T G G A C T G T T A T T T A G T T A A A T G G T C
A G T T T A A G A G A A A T T G T A G C A C G G C T T G A C G C T A C G C A G C
A T C A G A T C T A A T C C T A T A T A A A A G A A T C G A C C G A A G C T A C
T A A C C G A C A T G C G A T T C C A C T A A C T T A A A G C T A G A C G T A A
C C T C G A C A T G G G G T G G A A C G T A C C G T A A C T G T C G G A T T T C
G C A C T T C T T A C G T T A T G T C G A C T G T C T T T T T A T T G C C C G A
G C T A T T G C A A G G T T C T A A G A C T G A A T T G C T G C C G A T C G C T
C G C T C A G T A T T T A G G G C A G G T G T G G C C C G T T A G C C C A G C C
T C A C C T T T C C C C G C C C T A A A A T A A T A A T G C A C T G C G T C T A G
A G G C A C A G T G A T A T G A G T G T A G G A G A G A C A T C T A T T T C A A
T A T G G T T G G A G G T A T A A G A A G A A T G C G A T T C A A G C C C G A T
A G G C T C A G A G C C G G G T A T C G T C C T C G T G A A A T T C C C T T C A
G G A T A A C G G C T T A T G T C A T G C A A G G G G C G T T A T A C A A T A T
G A G T G G G T T T A T A C A A T T A T A C A A T A T A C A A T T T T G C G T C
A C A C C C T T A T A T A C A A T T A T A C A A T A C A C T T A T A C A A T T T T A
T A C A A T T T T A T A C A A T T G C T A C A A T C G G C A C T A T T T A T A C
A A T A A T T G C C G C A C G C A A T T A T A C A A T C G C T G C T G A C C C C
C A G T T A T A C A A T C G G T T G A A G G A G T T A T A C A A T T G G C C A A
T T A T A C A A T C A A T T C T A G T T A T T T A T A C A A T C G A T G C A T C
T G T T A T T T T C G T A T T C G T T A T A C A A T A T T A T A C A A T C T G T
C A A G A G A T T G G C T A T T A T A C A A T T C C G T A T G A A T T G G T C G
C T T A C T G T C T T T A T A C A A T T A T A C A A T T T A T T T A T A C A A T
C T A T T A T A C A A T G C T A T A A T G G G C G T G T A A C G A G G C T T A T
A C T T A T A C A A T T C C A C C A A G A G G C A T A A A T T A T A A C A C T C
T C T A T C G A A C A C T C T C T A C A A C A A C A C T C T C T C G A C A C T C
T C G A A A C G G C T C G G A A A T T T A T A A C A C T C T C A A T A C A T C A G
C C G G A C A C T C T C T A C A C A C T C T C T C A A T T T A T A C A A T C A A
T C G A C T C T C G A T G C G A C A C T C T C C G C T T A A C T G C A T C A C G
A A A A A C A A G A C A C T C T C T A C A C A C T C T C A C A C T C T C G C G C
A C A C T C A C A C T C T C A C T A A C A C G T A C C A G G T C A T T C T A C A
C T C T C A C A C T C T C A C T A G A T T G C G A T A C A C A C T C T C T C C C
A C A C T C T C C G A C G C A T A C T T C G T G T T T A C A C T C T C A A C A C
T C T C T A T G C A A T T C T C G G G C C T T C G A G C C G T A G T A T T C G A
C T C G T C T A A G T T A C A C T C T C C C G C T C G G C T G C C A T C C G A C
A C T C T C A G T A A T A A C A C T C T C A G C G C A C C A C A C T C T C A G G
G T A A A A T A C A C T C T A C A C T C T C G A G A C C C C G A C A C T A C A C
T C T C T C A T G C G G C T T C G C A C A C T C T C A G G A C A C T C T C T C T A A
G C C T C C A G A C C T A C T G T A A C A C T C T C G G A C G A A T G C G C T G
C A C T A C T T G C G C T G G C T G A T C G C T G G C G G G T G A T G A T G A G

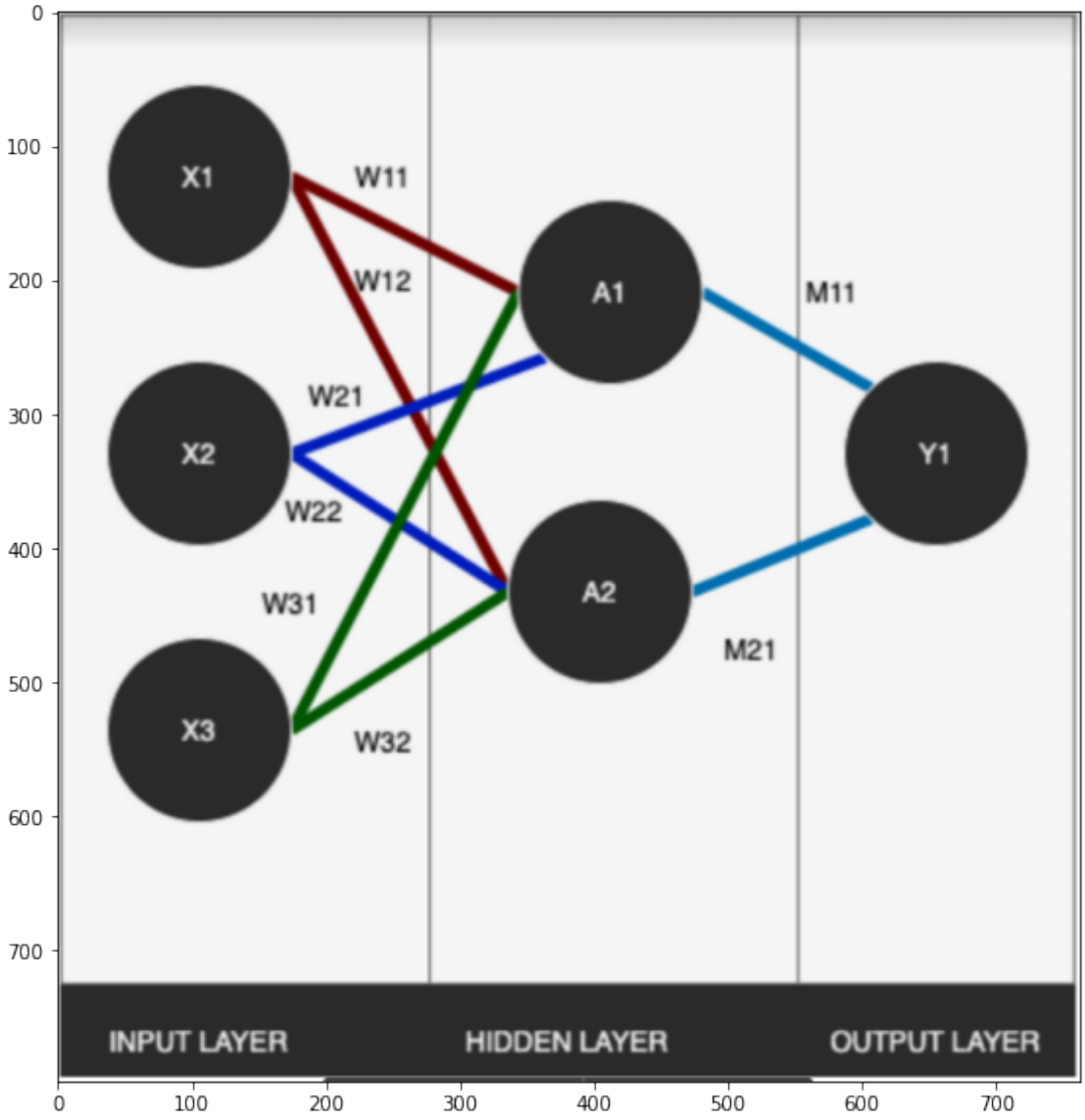
C G T C A A C C A G A T C T C C G T A A C G A A A T G A C T T T A T G C G T C C
T A C G A A T A C T G C G A G C G C G G T T A T G T A G C G C G A G C G T G A C
A T A C A G C G A A G T G G A A T T A G G A T T T C G A G T T T A T A T T G C C
T T T T T C T C T T T A A T C C T G C T G G C T C C C A G C A G G A G G C C A C
C A A A A G T G C T G A A G C G G T T A C C G C A C G A C G C A G C T T T A C A
C G A G T T T C G G G G C A T T T C G A G T C T G T G G T A C G T C C T T A C C
C T T A C A C A T G G G T C T C T A G G G A T C A T T C T C T C T A G G T T C T
G A A T T T C G G C A A G G C T C T C T C T A G A T T A G T G A T C T C T A G A
A T T G T G G T T A T C T C T A G G A G A T T C T C T A G T C A T C T C T A G C
G A A A A G T C T C T A T C T C T A G T G A G T G G C T C T C T A G A A T G T C
A A A C T A T A C A G T C A A G C C A A T T T T C G T C T C T A G C A G A C G T
C T C T A G C C A T C G C A T C T C T A G G G C A C A G C A T G T T T T T G A A
T C T C T A G T C T A G C G C G G A G C T T G A C A T G A A T C T C T A G A T G
T A A T A G A T T C T C T C T C T C T A G T C T C T A G T G T T C C G G T G T G T G
C T G T T T C A A T C T C T A G A T G T G T G C T A T C C A C C A C G G C T T G
G A C T C T C T A G G C C C A A A A C T C T C T C T A G T T C T C T A T C T C T
A G C A A T C T C T T C T C T A G A T C T C T A G C G T G C C C A A A A C C T C
T C T A G C A A G C C C C A A A A C A G C C C T T C T C T C T A A T T A C G C C A C T
C A A T T A C G C C C A T A T T A C G C C G T C T A T T A C A T T A C G C C A G
A T T A C A T T A C G C C C T C T A T T A C G C C A C T A C T T T G A A T T A C
G C C G A T T A C G C C A A T T A C G C C A G C T T G C G A T T A C G C C T C G
A T T A C G C C C C G C A T T G T A T T A C A T T A C G C C A A C A G T T A T A A
C A A A A A G T T A T A A T T A T A A G T T A A T G T T A T A A G T T T G C G T T
A T A A T T T G C C G G C C A T T A C G C C C C A G T T A G T T A T A A A A G C
A T T A C G C C C C C A A T T A C G C C A A A A A G T T A T A A T A G C C A T T A C
G C C C T C G A C C G T T A T A A C C A A A A A C C A T T A C G C C A A A T T A T
T A C G C C C C C G C T G T T A T A A C C C C A T T A C G C C T A T A A A T A G T T
A T A A C A C A A A A G T T A T A A A A T T G T G T T A T A A A C G G G C A T C C A T
A C T G G A T T A A T T A C G C C T A T A A T C C C C C G G T T A A T T A C G C C
T A G C A T T A C G C C C C A G C C C C G A A T A T T G T T A A T T A C G C C A G T
T A T A A T G A T T A C G A T T A C G C C C G C C T G A T G T T A T A A A T G T T
T T C T G A T G G T T A T T A C G C C T A T T A C G C C A G T T A T T A C G C C
T T C T A T T G C G C C C G T T A T A A C G G G C T G T T A T A A A C T G A T G T
G T T C T G A T G T G T T A T A A G G C A A T A A G A C A C G G T T G C G G T C
C A G T T A C G C A G C T T G G T T A T A A G A A C T A A C A C T A C G T C T G
A T G T G C T G A T G T T A T A A A T G G G G G C C C T G A T G T A T A G G T G
C T G A T G T C C C C G C T C T G A T G T A T C C C T G A T G T C T G A T G T T G
T T A A T A C C A G T G T A A T T G A G A C G G G C C G C C C G A G A A G G G A T
T T A G A G T G C A C T A C C T G A T C T G A T G T G G C T G A T G T T G T A T
G A A A C G T T G C T G A T G T C A T G C A A T T C T G A T G T C C T A A G T C
T G A T G T G A A C T A A A G G A C T G A T G T G A A G A C T G T T G G G C G T
G T A A C G G G G C G A T T G A G A C T A C C G G G G G T C T C T G A T G T A T G
G T A G C T C G C G C T G A T G T C C T G A T G T C G G C A T C T G G G A A A T
C T G A T G T G C G G T C C C G G T T G A C T G T G C C T A T T C C A G A A A C
G G G G C G T T C A C G A G C G G C T T A C A C T A A T T A G A G T T G T A A G
G C T G G A C G T T C T C G T G T G C G T A A A C T A T A C C C A T A T T C C T
T C T A G A G C A G G T C G A T A T T A C A T G T T G T A A A G G G G C A G T A
C T G A A C G A T G T A T T T G T C T T A T T C T G C A C T G C A G C G G T T A

T A T T C T G C A T G A G C T A A C T G G C A T T T T A A A A A G A T T C T T G
G T T T T A T T C T G C A T T C T G C A A G T G A A T T T A T T C T G C A T C C
C G C A A T G G C T A T T C T G C A A T T C T G C A C C T A G C G G T A G C G G
G C A C T C A G C G A G A G G G C G T A T T C T G C A A T T C T G C A G G G T T
A T C A C G A G G G A T G T G A A A T G G C C A C C A T C T A T T C T G C A T C
T G C A A T A T T C T G C A G C C C A T T T A T A T T C T G C A A T A A G G G T
T A T T T A T T C T G C A T T A G G G A C A T T G T G A C C T T C A C T A T T C
T G C A A A C A A G A T T G T A T T C T G C A A C A T T G A C G G G A T T G G G
A C T A T T C T G C A A A A T T T T T C A T G A T A T T C T G C A A G C T C C T
T A C T C T G G T A T T C T G C A G C A G G G A G G G A G T C G T G A C T T A A
A A A A G C T T C T A T T C T G C A T T C T G C A C A A C C A A A T A G C A A T
C T T T A T T C T G C A T G C A A A T A T T C T G C A T T A C C A G T A T T C T
G C A T G C A A T T C T G C A T T A T T C T G C A A T A G G T A G G G T T T T A
A T G T G C A G T C T T T A G T A C C G T T G G C G G C A C T A C C T T C T C T
C A T C G T T G G C T G A T G T A T G T C A T A T G A C A C C C G T C T G A G C
A A A C A T G T G G T T G T G A A G G C G G C G G T A A T A A T T T A T G C T A
A C C A C G A A A T G C G T A G A A C T A C T G G T A C C A A T G A G T G G A G
C C C A C G A C T G G G C G G A C A G A G G A T A C T G C A G C C C C G A G G T G
A T G C C G G G G C A A A G C T G T C T A T C C C C C C T C A A C T G G A G C T
T A C G C C C A A T G A A G C G G A C G G A A A G C T G C T T A G C C A T A C C
G A T C G A A C C T G T T C A T A T C C T A A C C A G A A A G T T C G A C G T G
A C A A A A C G T C G A A G A T C G C T C T A T T C C G A C T T C G G A G G T C
G C T A T A A C A G G T C A A C C T T T T T T C A A C C T T T T T A C C C C C A
A A C C T T T T T T C T T T T G C G G G C C C A A T G T G G C C C C T G T A T T
A A C C T T T T T T T G G T C A A C C T T T T T C G A A T C T T T C G A A C C T T
T T T C A G A A C C C T T G T T A A T A A C C T T T T T T G C T A C C C G C T G A
C T C T C A A C C T T T T T T T T T T A A C C T T T T T T A A C C T T T T T T G C A C
C G A A A C C T T T T T T A C C T T T T T T C T A A C C T T T T T T A A C C T T T T T T
G A A A A C C T T A A C C T T T T T T T T T T C G G T G A C G C C C A C G A A A C C
T T T T T T A T A A A C C T T T T T T T G A T C G T T T C G C C G T A A G A C T C
T C T A A C C T T T T T T G C A C G A T T C G A A G A A A C C T T T T T T C T T A A
C C T T T T T T T T T T T C G C G T G G T G A G T C C T T C T G T A C A G A C C G
T G A A A T C G C A A T T T C A A A C C T T T T T T T G A G G A G G G T G T A A A
C C T T T T T A C C T T T T T C T T A G C C A A T C T C G C C G T G C A C A G T
A T A A C C T T T T T T A T G A G T C G C G C A A T C G T C A A C C T T T T T T T T
A C T A C T G A T A C A A A C C T T C T G T T C C T C T T T C A G A G G C T T G
T T G T A G G T A C T G T T C C T C C T C C G A C C T G T T C C T A A G T C C G
A C A A G T C T G T T C C T C C C T G C T G T T C C T T C C T G A C A A G T C C
G A C C T G T T C T G T T C C T G A C A A C T G T T C C T G T C C T G T T C C T
G C T T T C C G A C A A G T C C C T G T T C C T T C C T G T C C G A C A A G T C
T C C T G C T C C T G C T G T T C C T T C C G A C A A G T C C G A C A A G T C C
T C C T G C T C C T T C C T A C A A G C T G T T C T C C T G C T G T C C G A T C
C T G C T G C T T C T C C T G C T G A C A A G T C C G A C A T C C T G C T C C T
G C T T T C C T A C A A C T G T T C C T C C T C T C C T C C T G C T C C T T C C
T G C T T T C C T C T G T T C C T C T G T T C C T C T G T T C C T T C C T G C T
T C C T G C T T C C T G C T T C C T G C T T C C T G C T T C C T G C T T C C T G
T C T C C T G T C C T G C T G C T T C C T G C T T C C T G C T T C C T G T C C T
G C T T C C T G C T T C C T G C T T C C T G C T G T T A G T A G T T A G T A G T

```
T A G T A G T T A G T A G T T A G T A G T T A G T A G T T A G T A G T T A G T A
G T T A G T A G T T A G T A G T T A G T A G T T A G T A G T T A G T A G T T A G
T A G T T A G T A G T T A G T A G T T A G T A G T T A G T A G T T A G T A G T T
A G T A G T T A G T A G T T A G T A G T T A G T A G T T A G T A G T T A G T A G
T T A G T A G T T A G T A G T T A G T A G T T A G T A G T T A G T A G T T A G T
A G T T A G T A G T T A G T A G T T A G T A G T T A G T A G T T A G T A G T T A
G T A G T T A G T A None
```

6. Neural Network

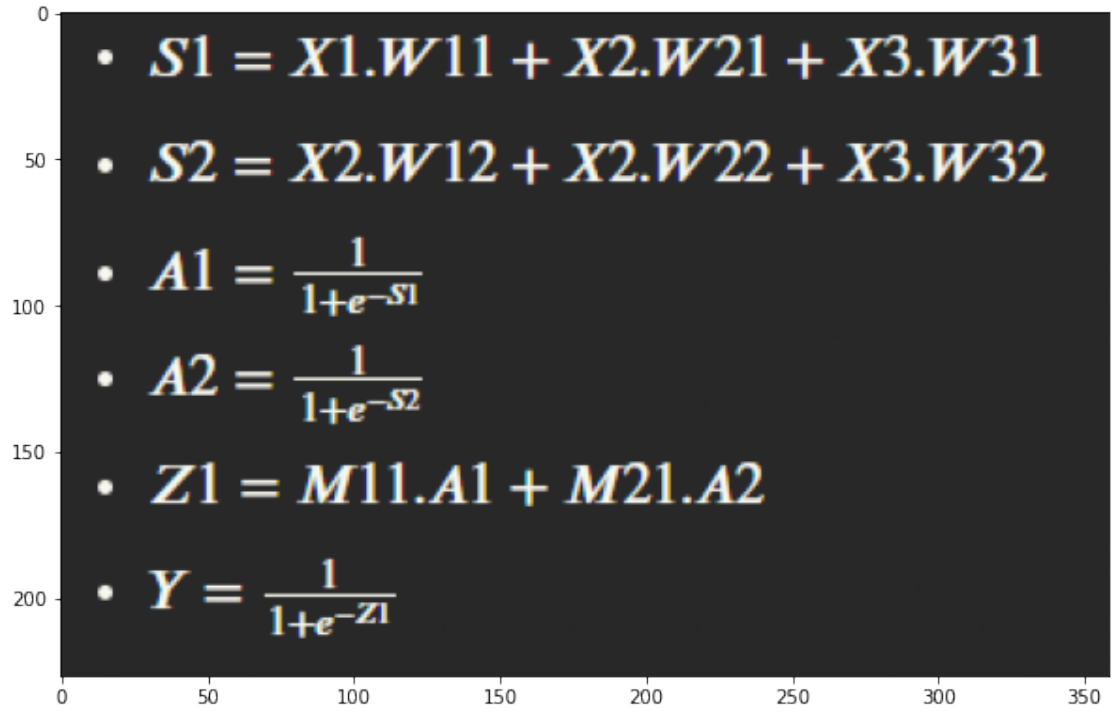
[19]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../jm2.png')

fig, ax = plt.subplots(figsize=(15, 10))
plt.imshow(img)
plt.show()
```



- Terdapat proses yang dinamakan feed-forward. Input dalam sebuah neural network diproses ke hidden layer hingga ke output layer.
- Setiap Node, menunjukan neuron dan setiap garis menunjukan weight.
- Proses Feed-Forward berjalanan dari input layer menuju output layer.
- Nilai yang masuk ke neuron di hidden layer adalah penjumlahan antara perkalian weight dengan
- nilai yang masuk pada input neuron setelah itu diaktifkan dengan fungsi aktivasi. Atau dapat dimodelkan sebagai berikut:

[26]:
```
img = mpimg.imread('4.png')

fig, ax = plt.subplots(figsize=(10, 8))
```

```
plt.imshow(img)
plt.show()
```



[20]:
```
W = [[0.5, 0.4],[0.3, 0.7],[0.25 , 0.9]]
M = [[0.34] , [0.45]]
X = [9, 10, -4]
```

[21]:
```
import math

def WTi(W, i): #Untuk mengambil setiap elemen pada index i dalam list
    return list(map(lambda w: w[i], W))

def WT(W): #Mengaplikasikan func WTi berdasarkan jumlah elemen list dalam list,␣
 ↪sehingga W akan berubah bentuk ke bentuk transposenya
    return list(map(lambda i : WTi(W, i), range(len(W[0]))) )

def XW(X,W):#Mengalikan Wji * Xi dan menjumlahkan semua hasil perkaliannya
    return map( lambda w: r(add, map( lambda xx,ww : xx * ww, X, w), 0), WT(W) )

def A(x): #fungsi aktivasi
    return 1/(1+math.exp(-x))

def hidden_layer(X,W): #Mapping dari input layer ke hidden layer atau dari␣
 ↪hidden layer menuju output layer
```

```
    return list(map(lambda x: A(x), XW(X,W) ))

def feed_forward(X, W, M): #menghitung input layer dan hidden layer dengan␣
 ↪fungsi untuk menghasilkan output layer
    return hidden_layer(hidden_layer(X,W), M)
```

[22]:
```
hidden_layer(X,W)
```

[22]:
```
[0.998498817743263, 0.9990889488055994]
```

[23]:
```
feed_forward(X,W,M)
```

[23]:
```
[0.6876336740661236]
```