VERSION 2.0

JULI , 2022

[PRAKTIKUM PEMROG. FUNGSIONAL]

MODUL 5 – *Functional programming on linear algebra case studies*.

DISUSUN OLEH :
Mutaqhin Dean
Andi Shafira Dyah K.

DIAUDIT OLEH
Fera Putri Ayu L., S.Kom., M.T.

PRESENTED BY: TIM LAB-IT
UNIVERSITAS MUHAMMADIYAH MALANG

PERSIAPAN MATERI

Praktikan diharapkan telah menguasai materi dari modul-modul sebelumnya.

TUJUAN PRAKTIKUM

1. Mahasiswa mampu mengimplementasikan pemrograman fungsional pada studi kasus logika pemrograman.
2. Mahasiswa mampu mengimplementasikan pemrograman fungsional pada matriks dan aljabar linear.

PERSIAPAN SOFTWARE/APLIKASI

- Komputer/Laptop
- Sistem operasi Windows/Linux/Mac
- Pycharm/Google Colab/ Jupyter Notebook

MATERI POKOK

Karena hampir semua materi pada praktikum pemrograman fungsional langsung diimplementasikan dalam *source code*, maka semua materi juga bisa anda akses pada *Google Collab* melalui tautan ini.

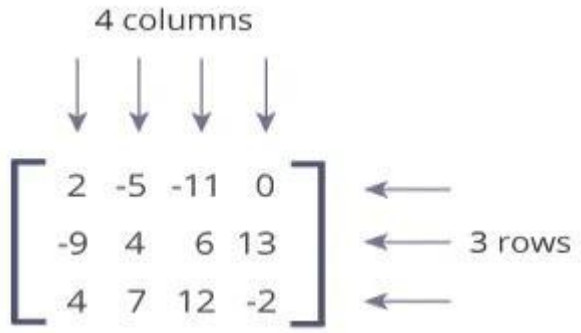
Modul 5 Pemrograman Fungsional

Functional programming on linear algebra case studies.

4 9 cells hidden

Matrix pada python

Matriks merupakan struktur data 2 dimensi yang memiliki baris dan kolom. di Python, tidak ada built-in tipe data yang support untuk matriks.



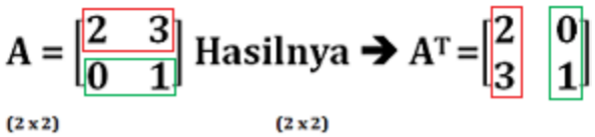
Tapi kita bisa menggunakan list untuk menyusun suatu matriks. Lebih tepatnya menggunakan *list of list* atau *nested list* (list yang berisi list). Agar mudah memahaminya, perhatikan blok kode dibawah ini.

```
list_normal = [1,1,2,3,5,8,13]
list_matrix = [[1,2,3,4],
               [5,6,2,1],
               [1,2,5,2],
               [1,6,2,9]]

print(list_normal)
print(list_matrix)

# tips !!, agar lebih rapi, gunakan perulangan untuk mem-print matrix (list 2d)
for baris_matrix in list_matrix:
  print(baris_matrix)
```

```
[1, 1, 2, 3, 5, 8, 13]
[[1, 2, 3, 4], [5, 6, 2, 1], [1, 2, 5, 2], [1, 6, 2, 9]]
[1, 2, 3, 4]
[5, 6, 2, 1]
[1, 2, 5, 2]
[1, 6, 2, 9]
```

Perobaan 1 (Mencari transpose matrix)

Bentuk matriks transpose diperoleh dengan cara menukar elemen-elemen baris suatu matriks menjadi elemen-elemen kolom dan menukar elemen-elemen kolom menjadi elemen-elemen baris. Ini berlaku untuk matrix ordo berapapun(bebas).




```
#fungsi transpose dengan inner function wrapper untuk menangkap inputan yang berupa
fun def transpose(function):
  def wrapper(*Args, **Kwargs):
    X = function(*Args, **Kwargs)
    result = [[X[j][i] for j in range(len(X))] for i in range(len(X[0]))]
    return result
  return wrapper

@transpose #Decorator transpose
def Tmatrix(matrix):
  return matrix

#fungsi untuk mencetak list matrix
def cetakMatrix(m):
  for baris_matrix in m:
    print(baris_matrix)

 mat2x2 = [[2,3],
           [0,1]]

mat3x3 = [[2,1,-1],
          [1,0,1],
          [1,-2,1]]

transposed = Tmatrix(mat2x2)
cetakMatrix(transposed)
cetakMatrix(Tmatrix(mat3x3))
```

```
[2, 0]
[3, 1]
[2, 1, 1]
[1, 0, -2]
[-1, 1, 1]
```

Perobaan 2 (Mencari determinan matrix ordo 2x2)

Misalnya A adalah matriks persegi berordo dua yang dituliskan dalam bentuk $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, maka determinan matriks A adalah:

$$\det A = |A| = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

```
#Fungsi ini menerima input sebuah nested list dan akan me-return determinant dari list
def det(matrix):
  if len(matrix) != 2 or len(matrix[0]) !=2 or len(matrix[1]) !=2:
    print("matriks harus berordo 2 x 2")
    return
  else:
    a = matrix[0][0]
    b = matrix[0][1]
    c = matrix[1][0]
    d = matrix[1][1]
    det = a*d - b*c

  return det

matriks = [[3,2],
           [5,6]]

print(det(matriks))
```

```
     8
```

bisa juga dituliskan secara langsung seperti berikut:

```
#Fungsi ini menerima input sebuah nested list dan akan me-return determinant dari list
def det(matrix):
  if len(matrix) == 2 or len(matrix[0]) ==2 or len(matrix[1]) ==2:
    return (matrix[0][0]*matrix[1][1]) - (matrix[0][1]*matrix[1][0])
  else:
    print("fungsi ini hanya untuk matriks berordo 2 x 2")
    return

print(det(matriks))
```

```
     8
```

Hasilnya sama saja. Tapii, bisakah kalian amati perbedaannya?

Percobaan 3 (Mencari invers matrix ordo 2x2)

Diketahui: $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$,

Maka, invers matriks A:

$$A^{-1} = \frac{1}{det\ A} . Adj\ A = \frac{1}{\det A} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

Matriks dari A inverse dirumuskan sebagai berikut:

inverse(A) = (1/determinan A)(adjoin A)

Dari rumus diatas, kita memerlukan fungsi untuk menghitung determinan, adjoin, dan fungsi perkalian antara nilai tunggal(1/det) dengan nilai matrix adjoin.

Well, karena fungsi untuk menghitung determinant matriks telah kita buat pada percobaan 2, maka kita cukup membuat fungsi adjoin() dan fungsi perkalian() yang akan kita panggil bersama fungsi det() di dalam fungsi inverse() nantinya. Cek kode berikut:

```
#fungsi yang akan mengembalikan hasil adjoin dari inputan matrix
def adjoin(matrix):
  if len(matrix)!=2 or len(matrix[0]) !=2 or len(matrix[1]) !=2:
    print("matrix harus berordo 2x2")
    return

  amatrix = matrix[:]
  return [[amatrix[1][1], -1*amatrix[0][1]],
          [-1*amatrix[1][0], amatrix[0][0]]]


def perkalian(x): #fungsi untuk mengalikan sebuah nilai tunggal x
  def matrix(a):  #dengan sebuah matrix a
    hasil = []
    for i in range(len(a)):
      row = []
      for j in range(len(a[i])):
        row.append(x*a[i][j])
      hasil.append(row)
                  #yang akan mengembalikan matrix hasil perkalian
    return hasil
  return matrix
```

```
#fungsi untuk mencari inverse dari matrix yang berupa nested list
def inverse(amatrix):
  if len(amatrix)!=2 or len(amatrix[0]) !=2 or len(amatrix[1]) !=2:
    print("matrix harus berordo 2x2")
    return
  return perkalian(1/det(amatrix))(adjoin(amatrix))


matriks = [[3, 2],
           [5, 6]]
hasil_invers = inverse(matriks)

#pastikan anda sudah menjalankan kode percobaan 1
#untuk dapat menggunakan fungsi cetakMatrix() berikut:
cetakMatrix(hasil_invers)
```

```
     [0.75, -0.25]
     [-0.625, 0.375]
```

Coba perhatikan, dari fungsi `det()`: `adjoin()`: hingga `inverse()`: kita mengulang kode kondisi if yang sama untuk mengecek ordo matrix. Penggunaan kode yang sama persis secara berulang seperti ini terhitung sebagai suatu pemborosan. Agar lebih e&#xfffd;sien, maka kita bisa buatkan sebuah fungsi tersendiri untuk cek ordo matrix seperti berikut:

```
def isOrdo2(matrix):
  if len(matrix) == 2 or len(matrix[0]) ==2 or len(matrix[1]) ==2:
    return True
  else:
    print("matrix harus berordo 2x2")
    return False
```

sehingga fungsi adjoin dan inverse dapat kita modif menjadi:

```
def adjoin(amatrix):
  if isOrdo2(amatrix):
    return [[amatrix[1][1], -1*amatrix[0][1]],
            [-1*amatrix[1][0], amatrix[0][0]]]

def inverse(amatrix):
  if isOrdo2(amatrix):
    return perkalian(1/det(amatrix))(adjoin(amatrix))

cetakMatrix(inverse(matriks))
```

```
     [0.75, -0.25]
     [-0.625, 0.375]
```

Naah, kode program kita sekarang jadi lebih bersih dan enak dibaca kan... ^_^

Kalian juga dapat mengubah fungsi det() dengan cara yang serupa seperti contoh diatas.

Nested loop for pada fungsi perkalian juga dapat disederhanakan menggunakan list comprehension lho(cek kembali materi modul 2), secara fungsi tersebut hanya berisi append list dan akan mengembalikan sebuah list matrix. Cek kode berikut:

```
def perkalian(x): #fungsi untuk mengalikan sebuah nilai tunggal x
  def matrix(matrixA):  #dengan sebuah matrix A
    #yang akan mengembalikan matrix hasil perkalian x*a
    return [[x*a for a in baris_matrix] for baris_matrix in matrixA]
  return matrix

inv = perkalian(1/det(matriks))(adjoin(matriks))
cetakMatrix(inv)
```

```
     [0.75, -0.25]
     [-0.625, 0.375]
```

Waaahh, isi fungsinya sekarang jadi cuma tinggal sebaris kode |* o *| berarti bisa pakai fungsi lambda juga dong, mari kita coba:

```
kalikan = lambda x, matrixA: [[x*j for j in i] for i in matrixA]

inv = kalikan(1/det(matriks),adjoin(matriks))
cetakMatrix(inv)
```

```
     [0.75, -0.25]
     [-0.625, 0.375]
```

Seperti itulah pemrograman fungsional dapat digunakan untuk menyelesaikan persoalan matematis dengan kode yang ringkas.

Exception Handling

Pada saat pembuatan program kita tentunya sering melakukan kesalahan, yang menyebabkan error ketika dijalankan. Ada dua jenis error yang ada pada python yaitu Syntax error dan Exception. Syntax error terjadi dikarenakan adanya kesalahan dalam penulisan syntax, sedangkan exception adalah error yang terjadi saat program sedang berjalan meski syntax yang kita masukkan sudah benar.

Berikut adalah contoh syntax error:

```
a =  3
if a> 3
```

```
    print("a lebih besar dari 10")

#SyntaxError: invalid syntax
```

Kenapa bisa error?? Yap, karena ada syntax yang invalid/salah/kurang dalam pemakaian if, yaitu tanda titik dua (:)

Nah kalau kode berikut ini adalah contoh dari sebuah Exception:

```
a = 5
b = 0
hasil = a/b
#ZeroDivisionError: division by zero
```

`ZeroDivisionError` merupakan salah satu jenis exception. Pada saat exception terjadi python akan menampilkan traceback, detail error terjadi, dan program akan langsung dihentikan. Tentunya kita tidak ingin program kita berhenti dikarenakan ada error yang terjadi. Terlebih lagi, pesan error yang diberikan terkadang sulit dipahami, khususnya bagi mereka yang tidak terbiasa dengan pemrograman. Nah, kita bisa mengatasi ini dengan beberapa cara berikut.

1. Raise Exception

Kita dapat menggunakan keyword raise untuk meng-handle exception. Perhatikan contoh berikut:

```
a = int(input('masukkan nilai a '))
b = int(input('masukkan nilai b '))
if b==0:
  raise Exception("nilai pembagi b tidak boleh 0")
else:
   print('hasil a/b adalah',a/b)
```

Exception handling yang pertama ini tidak jauh berbeda dengan pesan error sebelumnya, hanya saja kita dapat mengubah pesan error sesuai dengan skenario program kita. Hal ini bisa sangat membantu kita dalam melakukan debugging.

2. try, except

Pada python, exception bisa di atasi dengan try except statement. Dimana kode yang berpotensi menimbulkan exception/error kita masukkan ke dalam blok try, dan pada blok except dimasukkan kode yang akan dieksekusi saat terjadi error, berikut strukturnya:

```
try:
    kode yang berpotensi error
except:
    kode saat error terjadi
else: #(optional)
    kode yang akan dijalankan jika tidak terjadi error
```

```
try:
    a = int(input('masukkan nilai a '))
    b = int(input('masukkan nilai b '))
    print('hasil a/b adalah',a/b)
except Exception as err:
    print(err)
```

```
masukkan nilai a d
invalid literal for int() with base 10: 'd'
```

Bila anda memasukkan angka 0 untuk nilai b atau bahkan bukan angka, hal ini tidak akan menimbulkan error dikarenakan kita kita sudah mengantisipasinya dengan menggunakan try excep. Dalam kode diatas, exception yang terjadi akan disimpan kedalam variabel err dan dicetak sebagai keterangan. Nah, hasil nya jauh lebih bersih dan enak dipandang daripada sebelumnya kan. Hal ini sangatlah cocok untuk mengatasi error saat program dijalankan oleh end user.

3. try, except, finally

Berikutnya, anda dapat menambahkan blok kode finally, yang mana akan selalu dieksekusi apapun kasusnya (baik saat terjadi error maupun tidak). Mari kita lihat contoh penggunaannya berikut:

```
try:
    a = int(input('masukkan nilai a '))
    b = int(input('masukkan nilai b '))
    hasil = a/b
except Exception as err:
    print(err)
else:
    print('hasil a/b adalah',hasil)
finally:
    print('blok kode finally dijalankan')
```

```
masukkan nilai a 1
masukkan nilai b f
invalid literal for int() with base 10: 'f'
blok kode finally dijalankan
```

PRAKTIKUM MODUL

Kegiatan 1

Dengan menggunakan metode invers matriks, tentukanlah himpunan penyelesaian dari sistem persamaan linear tiga variabel berikut ini:

$$2x + y - z = 1$$

$$x + y + z = 6$$

$$x - 2y + z = 0$$

Dengan mengikuti alur penyelesaian sebagai berikut:

1. Masing-masing persamaan terlebih dahulu disusun dalam bentuk matriks berikut:

   A*X=B

   Dimana:

   - Matriks A memuat koefisien-koefisien ketiga persamaan.
   - Matriks X memuat variabel x, y, dan z.
   - Matriks B memuat konstanta-konstanta ketiga persamaan linear.

   Dengan demikian, bentuk matriks A * X = B adalah sebagai berikut:

$$\begin{vmatrix} 2 & 1 & -1 \\ 1 & 1 & 1 \\ 1 & -2 & 1 \end{vmatrix} \begin{vmatrix} x \\ y \\ z \end{vmatrix} = \begin{vmatrix} 1 \\ 6 \\ 0 \end{vmatrix}$$

2. Untuk menentukan nilai x, y, dan z maka bentuk matriks harus kita ubah menjadi bentuk invers seperti berikut:
   X = inverse(A) * B

3. Gunakan rumus inverse seperti pada percobaan
   3: inverse(A) = (1/determinan A)(adjoin A)

4. Determinan matrix 3x3 dapat menggunakan aturan sarrus seperti berikut:

$$det\ A = |A| = \begin{vmatrix} a & b & c & a & b \\ d & e & f & d & e \\ g & h & i & g & h \end{vmatrix}$$

$$det\ A = |A| = a.e.f + b.f.g + c.d.h - (c.e.g + a.f.h + b.d.i)$$

5. Untuk adjoin matriks A gunakan rumus berikut:
   adjoin(A) = transpose(matriks kofaktor A)
   Jadi sebelum dapat menentukan adjoin matriks, anda harus menentukan dahulu matriks kofaktor A yang ditranspose.

Untuk memberikan gambaraan terkait penyelesaian sistem persamaan diatas, anda bisa mengunjungi tautan berikut.

Anda dapat memanfaatkan fungsi-fungsi dari percobaan 1, 2, dan 3 dengan beberapa modifikasi agar sesuai dengan persoalan yang diberikan. Lengkapi blok kode berikut sesuai perintah dari komen yang tersedia.

```
#fungsi transpose dengan inner function wrapper untuk menangkap inputan yang berupa
fun def transpose(function):
  def wrapper(*Args, **Kwargs):
    #fungsi untuk mentranspose matrix
    return
  return

#tambahkan dekorator untuk mentranspose matrix kofaktor
def adjoin(matrix):
  #fungsi untuk menghitung matrix kofaktor
  return

def det(matrix):
  #fungsi untuk menghitung determinan matrix 3x3
  return

#fungsi untuk mencari inverse dari matrix yang berupa nested list
def inverse(amatrix):
  perkalian = #gunakan fungsi lambda disini untuk menggantikan fungsi perkalian
              #yang mengalikan sebuah nilai tunggal x dengan
  matrix return #fungsi ini mereturn hasil perkalian sesuai rumus
                #invers: (1/determinan A)(adjoin A)

#fungsi untuk menghitung nilai X yang merupakan perkalian inverse(A) dengan matrix B
def perkalianMatrix(function, matrix):
  return  #yang akan mengembalikan sebuah list untuk nilai x, y, dan z

def result(function):
  def wrapper(*Args, **Kwargs):
    #Isi fungsi ini sebagai decorator untuk mencetak hasil
    #Dapatkan nilai x, y, z dari function
    #Fungsi ini akan mereturn string x,y,z beserta masing" valuenya sebagaimana yang di
    return f"Nilai x, y, dan z masing - masing adalah {int(x)}, {int(y)} dan {int(z)}"
  return wrapper

#tambahkan dekorator untuk mencetak hasil
def get_result(matrixA, matrixB):
  return #panggil fungsi untuk mendapatkan nilai X dari matrixA dan matrixB
```

```
A = #tuliskan isi matrix A disini
B = #tuliskan isi matrix B disini
X = get_result(A,B)
print(X)
# output = Nilai x, y, dan z masing - masing adalah 1, 2 dan 3
```

Kegiatan 2

Karena kita sudah menyelesaikan hampir semua materi modul. Maka tugas modul kali ini adalah **bermain** dengan materi yang barusan kita pelajari, yaitu matriks. Jangan lupa untuk tetap memanfaatkan materi-materi sebelumnya ya (pure function, map, filter, lambda, dll), biar kodingan kalian nanti fungsional ;D

Agar kita bisa bermain, kita buat dulu yuk game nya seperti ini:

1. Program game berisi sebuah matriks 7x7 atau 10x10. Dimana matriks tersebut merupakan sebuah board game.
    - Kalian bisa banget memanfaatkan list comprehension untuk mengenerate board matrix ini. Biar ga capek ketik satu-satu sebanyak 7x7 atau 10x10 titik khan T-T
2. Terdapat sebuah bidak (disimbol A) yang dapat berjalan secara horizontal dan vertikal.
3. Jika bidak bergerak mencapai posisi tujuan (goals, disimbol O) maka permainan selesai.
4. Posisi awal bidak dan posisi goals akan digenerate secara acak saat permainan dimulai. Kalian diijinkan menggunakan library untuk mengacak posisi.

Berikut adalah contoh preview game yang bisa kalian jadikan inspirasi:

```
*** Selamat Datang di Board Game Pemrograman Fungsional ***
  Anda (A) dapat berjalan secara horizontal dan vertikal untuk menuju target (O)
  Selamat Bermain
-   -   -   -   -   -   -
-   -   -   -   A   -   -
-   -   -   -   -   -   -
-   -   -   -   -   -   0
-   -   -   -   -   -   -
-   -   -   -   -   -   -
-   -   -   -   -   -   -

==========================
  1. Cek Posisi
  2. Geser kanan
  3. Geser kiri
  4. Geser atas
  5. Geser bawah
  6. Keluar
==========================
Pilihan: 1
(1, 4)
```

Ada beberapa menu dasar yang harus ada dalam game nih:

1. Cek posisi --> untuk mengetahui dimana posisimu (A) berada saat ini
2. Geser kanan, kiri, atas, bawah --> hati-hati, jangan sampai kamu (A) hilang dari papan game yah. Tembok aja tempat mainnya.
3. Keluar --> mungkin kalian bosen muter-muter dan pengen segera cap cus nantinya.

Menu menarik lainnya boleh banget kalau kalian mau nambahin. Tapi tetap gunakan jalur fungsional yah guyys! Jangan cuma asal bisa main, eh ternyata kalian ngodingnya prosedural banget T-T kita sudah sampai modul 5 loh ini!

Rubrik Penilaian

1. Kegiatan 1:

    > - Program berjalan sesuai ketentuan. (+25)
    > - Program menerapkan `ZeroDivisionError` handling untuk nilai det 0. (+10)

2. Kegiatan 2:

    > - Program berjalan dengan baik sesuai skenario yang diberikan (+20)
    > - Program menerapkan exception handling. (+10)
    > - Program menerapkan materi dari modul modul sebelumnya (lambda, pure function, HoF, dsb). (+10)
    > - Praktikan bisa meminimalisir penggunaan kode yang sama persis secara berulang. (+10)

3. Menjelaskan dengan singkat, baik dan lancar kepada asisten dan atau dapat menjawab pertanyaan asisten dengan tepat saat demo. (+15)
4. Jika program identik dengan praktikan lain. Maka akan ada pengurangan nilai pada kedua praktikan. (-20)
5. Dilarang keras menggunakan library tambahan selain library random untuk kegiatan 2 dan library functools untuk menggunakan reduce. (-30).