VERSION 2.1

AGUSTUS , 2023

[PRAKTIKUM PEMROG. FUNGSIONAL]

MODUL 3 – Higher Order Functions for Processing Structured Data.

DISUSUN OLEH :
Fildzah Lathifah
Hania Pratiwi Ningrum

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

PRESENTED BY: TIM LAB-IT

UNIVERSITAS MUHAMMADIYAH MALANG

PERSIAPAN MATERI

Praktikan diharapkan telah memahami fungsi lambda dan iterator/iterable object/sequence pada modul 2 sebelumnya karena akan sering digunakan pada modul ini.

TUJUAN PRAKTIKUM

1. Praktikan diharap dapat memahami konsep Higher Order Function dan Build-in Higher Order Function pada python.
2. Praktikan diharap dapat memahami dan menggunakan teknik Currying dalam pemrograman Fungsional
3. Praktikan diharap dapat mengelola data berdasarkan proses yang tepat sebagai bentuk komputasi data

TARGET MODUL

Penguasaan materi:

1. Higher Order Function
2. Map
3. Filter
4. Reduce

PERSIAPAN SOFTWARE/APLIKASI

- Komputer/Laptop
- Sistem operasi Windows/Linux/Mac OS/Android
- Pycharm/Google Collab/ Jupyter Notebook

MATERI POKOK

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

1. Higher Order Function

Pada bahasa pemrograman “fungsional” atau bahasa yang multiparadigm (misalnya python), fungsi juga merupakan sebuah value. Seperti value pada umumnya, fungsi juga dapat di-*assign* ke variabel, diterima sebagai parameter fungsi, dan dikembalikan sebagai return value layaknya sebuah data string, integer, boolean, dan sebagainya. Dengan penjelasan tersebut, kita sampai pada definisi higher-order function:

Higher-order function merupakan fungsi yang menerima sebuah fungsi sebagai parameter input dan atau fungsi yang mengembalikan sebuah fungsi sebagai return value. Perhatikan contoh berikut:

1.1. Menerima sebuah fungsi sebagai parameter input

```
[ ]  #fungsi yang me-retrun hasil perkalian
     def pangkat(a,b):
       return a ** b

     #fungsi yang me-retrun hasil perkalian
     def kali(a,b):
       return a * b

     #fungsi hitung yang merupakan HoF
     def hitung(a, operasi, b):
       return  operasi(a,b)

     #fungsi 'pangkat' menjadi salah satu parameter input (argumen) dari fungsi 'hitung'
     #kemudian fungsi 'hitung' ditampung  pada variabel tiga_pangkat2
     tiga_pangat2 = hitung(3, pangkat, 2)

     #fungsi 'kali' menjadi salah satu parameter input (argumen) dari fungsi 'hitung'
     #kemudian fungsi 'hitung' ditampung  pada variabel tiga_kali4
     tiga_kali4 = hitung(3, kali, 4)

     print(f"3^2 = {tiga_pangat2} \n3*4 = {tiga_kali4}")
```

```
3^2 = 9
3*4 = 12
```

```
[ ]  # kita juga bisa mengirimkan fungsi lain seperti print ke dalam fungsi hitung
     hitung(tiga_pangat2, print, tiga_kali4)
```

```
9 12
```

1.2. Mengembalikan sebuah fungsi sebagai return value

```
[ ]  def menu(pilihan):
         def pizza():
             return "Anda memesan Pizza"
         def bakso():
             return "Anda memesan Bakso"

         if pilihan==1:
            return pizza
         else:
            return bakso

     pizza = menu(1)
     bakso = menu(10)
     print(pizza)  # referensi ke object fungsi pizza
     print(bakso)  # referensi ke object fungsi bakso
```

```
<function menu.<locals>.pizza at 0x7f492be617a0>
<function menu.<locals>.bakso at 0x7f492be614d0>
```

Kenapa outputnya bukan tulisan "Anda memesan Pizza/Bakso"? Bukankah sudah di print juga?

1. Karena yang di return oleh fungsi menu() adalah sebuah fungsi
2. Maka hasil yang disimpan pada variabel pizza maupun bakso adalah juga fungsi
3. Sehingga saat kita print pizza dan bakso, hasilnya adalah sebuah referensi ke object fungsi pizza/bakso

Untuk menampilkan tulisan "Anda memesan Pizza/Bakso", kita perlu memperlakukan variabel pizza dan bakso sebagai fungsi dan memanggilnya seperti kita memanggil fungsi pada umumnya, seperti ini:

```
[ ]  pizza()
```

```
'You ordered Pizza'
```

```
[ ]  bakso()
```

```
'You ordered Bakso'
```

Kita juga dapat memodifikasi fungsi HoF yang pertama menjadi HoF tipe 2 seperti berikut:

```
[ ]  def operasi(op):
       def pangkat (a,b):
         return a ** b
```

```
    def kali(a,b):
      return a * b

    if op == '^':
        return pangkat
    elif op == '*':
        return kali

pangkat = operasi('^')
kali = operasi('*')
print(f"3^2 = {pangkat(3,2)} \n3*4 = {kali(3,2)}")
```

```
3^2 = 9
3*4 = 6
```

Untuk fungsi Hof tipe 2 ini, kita perlu mendeklarasikan fungsi yang akan di return di dalam fungsi utama. Hal ini agar fungsi HoF tetap bisa mempertahankan karakteristik fungsional, yaitu sebagai fungsi murni (pure). Dimana sebuah fungsi tidak boleh bergantung pada hal lain diluar fungsi. Deklarasi seperti ini dikenal juga dengan istilah inner function.

Selanjutnya, dipakai untuk apa sih fungsi orde tinggi (HoF) itu? Coba cek beberapa percobaan berikut:

Percobaan 1

Higher-order function sangat **bermanfaat untuk menyederhanakan dan meringkas kode** program dimana terdapat beberapa fungsi yang memiliki statement atau kode program yang sama. Coba perhatikan contoh kasus berikut:

Kita ketahui NIM mahasiswa UMM memiliki struktur sbb:

FT	T. Mesin	201210120311xxx
	T. Sipil	201210340311xxx
	T. Elektro	201210130311xxx
	T. Industri	201210140311xxx

buatlah fungsi untuk mencetak nim mahasiswa per jurusan dengan ketentuan diatas!

```
[ ]  def nimMesin(x):
       kodeMesin = '012'
       return "2021 1 " + kodeMesin + " 0311 " + x

     def nimSipil(x):
       kodeSipil = '034'
       return "2021 1 " + kodeSipil + " 0311 " + x

     def nimElektro(x):
       kodeElektro = '013'
       return "2021 1 " + kodeElektro + " 0311 " + x
```

```
def nimIndustri(x):
  kodeIndustri = '014'
  return "2021 1 " + kodeIndustri + " 0311 " + x

print(nimMesin('123'))
```

```
2021 1 012 0311 123
```

itu baru dari jurusan teknik. bagaimana dengan jurusan lain? haruskah dibuatkan fungsinya satu-satu? sepertinya melelahkan :(

Cukup merepotkan bukan harus menulis banyak fungsi yang terpisah, padahal isi parameter dan kode programnya hampir sama semua, cuma beda di variabel kodenya saja.

Mungkin teman-teman ada yang terpikirkan untuk bisa lebih menyederhanakan kode jika dibuat menjadi satu fungsi dengan menambahkan seleksi kondisi seperti ini:

```
[ ] def cetakNIM(jurusan,nim):
      if jurusan == "mesin":
        kode='012'
      elif jurusan == "sipil":
        kode='034'
      elif jurusan == "elektro":
        kode='013'
      elif jurusan == "industri":
        kode='014'
      return "2021 1 " + kode + " 0311 " + nim

    print(cetakNIM('mesin','123'))
```

```
2021 1 012 0311 123
```

tapi itu prosedural banget, gak fungsional. apa masih bisa lebih efisien lagi? Mari kita manfaatkan HoF agar lebih fungsional dengan memanfaatkan struktur data dict untuk menyimpan kode-kode jurusan:

```
[ ] kodeJur = {'mesin':'012',
               'sipil':'034',
               'elektro':'013',
               'industri':'014'}

    def cetakNIM(kodeJurusan):
      def cetak(angkatan, nim):
        return str(angkatan) + " 1 " + kodeJurusan + " 0311 " + nim
      return cetak

    nimMesin = cetakNIM(kodeJur['mesin'])
    print(nimMesin(2022,'123'))
```

```
2022 1 012 0311 123
```

Lebih sederhana bukan... Kode jadi lebih efektif dengan kita memanfaatkan higher order function. Kita dapat menghindari perulangan kode fungsi maupun nested if untuk mendata kode jurusan. Kita cukup menyimpan kode jurusan yang diperlukan dalam sebuah variabel yang lebih mudah diingat (`nimMesin, nimIndustri, dsb`) untuk kemudian dicetak sesuai nim mahasiswa. Hal ini juga akan menghindarkan kita dari kesalahan(human error) jika harus menuliskan kode jurusan sebagai parameter fungsi.

Kok bisa sesimple itu? Bagaimana cara membuatnya?? Pertama-tama, temukan dulu persamaan dan perbedaan dari fungsi-fungsi yang mirip uang bisa digabungkan.

Bisa kita lihat bahwa dari fungsi-fungsi nim diatas yang membedakan adalah variabel kode jurusan. Selebihnya semua sama, kecuali nama fungsi yang pastinya tidak boleh sama.

untuk membuat higher order function, statement/kode fungsi yang sama persis cukup kita tulis satu kali. Dan statement/kode fungsi yang berbeda bisa kita gabungkan jadi satu dalam sebuah inner function (konsep inner function akan dipelajari pada modul selanjutnya). Atau bisa juga dijadikan parameter, jika statement yang berbeda adalah sebuah fungsi seperti contoh berikut:

Percobaan 2

```
[ ]  #fungsi untuk menampilkan output (print)
     def write_repeat_print(message, n):
         for i in range(n):
             print(message)

     write_repeat_print('Hello', 3)
```

```
Hello
Hello
Hello
```

```
[ ]  #fungsi untuk menampilkan logging
     import logging
     def write_repeat_log(message, n):
         for i in range(n):
             logging.error(message)

     write_repeat_log('Hello', 3)
```

```
ERROR:root:Hello
ERROR:root:Hello
ERROR:root:Hello
```

Kita bisa memanfaatkan ciri higher order function yang pertama, yaitu menggunakan fungsi sebagai parameter atau inputan. Bisa kita lihat bahwa dari kedua fungsi diatas **yang membedakan adalah** di baris setelah for loop yaitu di baris `logging.error(message)` dan `print(message)` . Kita bisa membuat dua statement ini jadi satu, yaitu dengan menambahkan

parameter di fungsi.

Sehingga fungsi write_repeat_print dan write_repeat_log yang awalnya masing masing hanya memiliki 2 parameter, kita gabungkan jadi satu yaitu dengan fungsi bernama hof_write_repeat. Fungsi hof_write_repeat kita tetapkan parameternya yaitu sebanyak 3. Berikut adalah implementasinya:

```
[ ]  # Import logging library
     import logging
     def hof_write_repeat(message, n, action):
         for i in range(n):
             action(message)

     #gunakan fungsi hof_write_repeat untuk mencetak dengan print()
     hof_write_repeat('Hello', 3, print)
     #gunakan fungsi hof_write_repeat untuk mencetak log dengan logging.error()
     hof_write_repeat('Hello', 3, logging.error)
```

```
ERROR:root:Hello
ERROR:root:Hello
ERROR:root:Hello
Hello
Hello
Hello
```

Percobaan 3

Kita juga bisa memanfaatkan fungsi lambda sebagai return value

```
[ ]  def hof_product(multiplier):
         return lambda x: x * multiplier

     mult5 = hof_product(5) # 5 sebagai input parameter multiplier
     op5x7 = mult5(7) # 7 adalah input parameter x pada fungsi lambda
     print(op5x7) # 5x7 = 35
```

```
35
```

Sebagai latihan, coba kalian ubah fungsi cetakNim sebelumnya dengan memanfaatkan fungsi lambda sebagai return value nya! Lengkapi kode berikut:

```
[ ]  def cetakNIM(kodeJurusan):
       return lambda #lengkapi fungsi lambda agar sesuai dengan fungsi sebelumnya

     nimMesin = cetakNIM(kodeJur['mesin'])
     print(nimMesin(2022,'123'))
```

2. Built-in Higher Order Functions pada python

Tentu saja terdapat build-in Higher Order Funtion pada bahasa pemrograman python. Beberapa diantaranya mungkin tidak asing atau bahkan pernah anda gunakan. Terdapat banyak build-in HoF pada python, beberapa diantaranya yang menarik untuk dipelajari adalah map(), filter(), dan reduce(). Fungsi-fungsi ini mengambil fungsi dan iterator sebagai parameter input.

Sebagaimana yang telah kita pelajari sebelumnya tentang iterator, kita dapat menggunakan iterable object seperti: range, list, tuple, dictionary, generator, dll sebagai parameter input.

Saat kita bekerja dengan koleksi data(seperti range, list, tuple, dictionary, dl), dua pola pemrograman yang sangat umum muncul:

1. Melakukan iterasi koleksi **untuk membangun koleksi lain**. Pada setiap iterasi, terapkan beberapa transformasi atau beberapa tes ke item saat ini dan tambahkan hasilnya ke koleksi baru. Ini adalah konsep dari map() dan filter()
2. Melakukan iterasi dan proses akumulasi hasil **untuk membangun nilai tunggal**. len(), min(), max(), sum(), dan reduce() adalah contoh dari konsep ini.

2.1. Map

Fungsi map() digunakan untuk membuat koleksi baru dengan cara mengaplikasikan sebuah fungsi pada setiap elemen yang ada pada iterable object (range, list, tuple, dictionary, generator, dll).

Seperti arti namanya, map adalah peta. Digunakan untuk memetakan suatu data. Sehingga hasil dari map pastilah sama dalam hal jumlah/panjang datanya dengan iterable objek yang diberikan. Karena fungsinya untuk memetakan, maka kita **hanya boleh memasukkan fungsi yang akan merubah data (bukan fungsi logika).**

Sebagai contoh, kita memiliki sebuah list nama-nama dan kita akan memberikan setiap nama dalam list tersebut sebuah ucapan 'Hai'. Kita bisa menggunakan fungsi map() mendapatkan hasil tersebut :

```
[ ]  nama = ['Andi', 'Budi', 'Cici', 'Dilan'] #iterable object list
     nama_hai = map(lambda x: 'Hai ' + x, nama)
     print(type(nama_hai))
     #akan menghasilkan sebuah map object (sebuah iterator)
     print(nama_hai)
```

```
<class 'map'>
<map object at 0x7af155daebc0>
```

Sama seperti generator, fungsi ini juga mengimplementasikan konsep Lazy Evaluation. Masih ingat?? coba cek modul sebelumnya jika tidak ingat. Oleh karena itu, hasil dari map tidak bisa di print langsung. Harus melalui sebuah iterasi/loop for:

```
[ ]  for nama in nama_hai:
         print(nama)
```

```
Hai Andi
Hai Budi
Hai Cici
Hai Dilan
```

Atau bisa juga dengan mem-parsing data nya ke dalam sebuah list seperti ini:

```
[ ]  print(list(nama_hai))
```

```
['Hai Andi', 'Hai Budi', 'Hai Cici', 'Hai Dilan']
```

Atau bahkan sebuah tuple:

```
#tips !!!
#untuk mempermudah pemanggilan, object map bisa diparsing menjadi tipe data sequence
nilai_tambah7 = tuple(map(lambda x: 7 * x, range(10)))

print(nilai_tambah7)
```

```
(0, 7, 14, 21, 28, 35, 42, 49, 56, 63)
```

Tapi perlu diingat lagi ya, bahwa konsep lazy evaluation merupakan salah satu keunggulan paradigma fungsional. Dan bahkan data sekuen pun masing-masing memiliki kelebihan masing-masing. SO, be wise in using them all... Alias, disesuaikan kebutuhan ya..

2.2. Filter

Seperti namanya, fungsi filter() digunakan untuk menyaring data. Umumnya hasil dari filter akan lebih sedikit dari data semula. Kecuali seluruh isi data yang menjadi masukan filter memenuhi syarat dari fungsi penyaring.

Dengan menggunakan fungsi filter (), kita dapat mengaplikasikan sebuah fungsi penyaring pada setiap elemen yang ada pada iterable object (range, list, tuple, dictionary, generator, dll) untuk melakukan pengecekan yang menghasilkan sebuah return **True atau False**. Fungsi filter () hanya akan menyimpan element yang memiliki return True saja. Oleh karena itu, pastikan bahwa fungsi yang dipakai dalam filter adalah mengandung **conditional value** atau berupa **fungsi logika** (wajib ya..).

Sebagai contoh, kita memiliki sebuah list angka dan kita ingin menyortir angka dari list tersebut yang hanya bisa dibagi oleh 5. Kita bisa melakukan seperti dibawah ini :

```
[ ]  numbers = [13, 4, 18, 35] #iterable object

     div_by_5 = filter(lambda num: num % 5 == 0, numbers)
     print(type(div_by_5))
     #akan menghasilkan sebuah filter object
     print(div_by_5)
```

```
<class 'filter'>
<filter object at 0x7f85ddd59f10>
```

Sama seperti map, filter juga menghasilkan sebuah iterator baru dan mengimplementasikan konsep Lazy Evaluation. Sehingga hasil dari filter tidak bisa di print langsung. Harus melalui sebuah iterasi/looping, atau bisa juga langsung di parsing menjadi tipe data sequence lainnya seperti ini:

```
[ ]  print(tuple(div_by_5))
```

```
(35,)
```

2.3. Reduce

Fungsi reduce seperti fungsi map(), menerima dua argumen yakni sebuah fungsi dan iterable. Namun tidak seperti fungsi map, fungsi reduce harus di import terlebih dahulu dari modul functools.

```
from functools import reduce
```

Jika fungsi map menghasilkan iterable baru, fungsi reduce menghasilkan suatu nilai kumulatif dari operasi fungsi masukan terhadap nilai pada iterable masukan

```
[ ]  from functools import reduce
     ini_list = [1, 1, 2, 3, 5, 8]

     sum = reduce(lambda x, y : x + y, ini_list)
     print(f"jumlah semua element dalam list  = {sum}")
```

```
jumlah semua element dalam list  = 20
```

```
[ ]  from functools import reduce

     def faktorial(n):
       return reduce(lambda x, y : x * y, range(1,n+1))

     print(faktorial(5))
```

```
120
```

3. Currying

Dalam pemecahan masalah dan pemrograman fungsional, currying adalah praktik penyederhanaan eksekusi fungsi yang membutuhkan banyak argumen untuk mengeksekusi fungsi dengan argumen tunggal secara berurutan. Dengan kata lain, Currying adalah teknik mengubah fungsi dengan multiple parameter/argumen menjadi pecahan banyak fungsi, setiap fungsi harus mengambil setiap parameter yang ada.

Coba perhatikan ilustrasi matematika berikut:
Diketahui sebuah formula matematika sbb: `f(x, y) = (x*x*x) + (y*y*y)`

```
[ ]  def f(x,y) :
       return (x*x*x) + (y*y*y)

     print(f(2,3))
```

```
35
```

Secara matematis, setiap operasi yang ada akan dituliskan sebagai formula dengan lambang baru yang berbeda. Mari kita telusuri setiap operasi matematis yang ada pada formula diatas:

1. Operasi perkalian `(x*x*x)` dan `(y*y*y)` akan menjadi seperti ini:

   `h(x) = (x*x*x)`

   `h(y) = (y*y*y)`

```
[ ]  # maka kita juga bisa buatkan definisi fungsi h
     def h(n):
       return n*n*n

     # dan fungsi f dapat kita modifikasi menjadi
     def f(x,y):
       return h(x) + h(y)

     print(f(2,3))
```

```
35
```

Kode diatas adalah contoh fungsi umum biasa. Belum berupa currying. Mari kita lanjutkan langkah berikutnya:

2. Operasi **penjumlahan** antara h(x) dengan h(y), karena memiliki lambang fungsi yang sama {h} maka akan dituliskan sbb:

   `h(x)+h(y) = h(x)(y)` sehingga
   `f(x, y) = h(x)(y)` dan
   `Curry f = h(x)(y)`

```
[ ]  # berikut adalah contoh currying yang sebenarnya
     def h(x):
         def h(y):
           return y*y*y + x*x*x
         return h

     f=h(2)(3)
     print(f)
```

```
35
```

Apa bedanya dengan fungsi f(x,y) yang pertama???
Yang pertama merupakan fungsi yang biasa kita gunakan di python. Dan yang kedua merupakan implementasi dari metode currying, yang merupakan fundamental dari paradigma pemrograman fungsional.

Mari kita belajar dari contoh yang lebih sederhana berikut:

Percobaan 4

Maksud dari penjelasan currying tadi jika kita tulis di dalam code adalah seperti ini:
Fungsi sum kita tulis tanpa teknik currying

```
[ ]  # Fungsi sum jika ditulis tanpa teknik currying
     def sum(a,b,c,d):
       return a + b + c + d

     print(sum(1,1,2,3))
```

```
     7
```

Fungsi sum tadi jika diubah kedalam bentuk currying:

```
[ ]  def sums(a):
         def x(b):

             def y(c):
                 def z(d):
                   return a + b + c + d
                 return z
             return y
         return x

     sums(1)(1)(2)(3)
```

```
     7
```

Percobaan 5

Mari kita implementasikan pada kasus untuk menghitung jumlah detik pada satuan jam berikut:

```
[ ]  def konversi(j=0):
         def menit(m=0):
           def detik(d=0):
             return ((j*60)+m)*60+d
           return detik
         return menit

     data = "05:33:05"

     # kita split data untuk mendapatkan nilai jam, menit, dan detik
     data_split = data.split(':')
     print("data = ",data)
     print("data split = ",data_split)
```

```
    # kita simpan masing-masing valuenya dalam variabel terpisah
    jam = int(data_split[0])
    menit = int(data_split[1])
    detik = int(data_split[2])
    print("jam = ", jam)
    print("menit = ", menit)
    print("detik = ", detik)

    konvert = konversi(jam)(menit)(detik)
    print("hasil konversi = ", konvert) #19985
```

```
data =  05:33:05
data split =  ['05', '33', '05']
jam =  5
menit =  33
detik =  5
hasil konversi =  19985
```

LATIHAN PRAKTIKUM

KEGIATAN 1

Modifikasi percobaan 5 (currying) untuk mengkonversi sejumlah minggu dan hari lengkap dengan jamnya menjadi menit sehingga menghasilkan input dan output seperti ini :

```
data = ["3 minggu 3 hari 7 jam 21 menit",
        "5 minggu 5 hari 8 jam 11 menit",
        "7 minggu 1 hari 5 jam 33 menit"]
```

```
OutputData =  [35001, 58091, 72333]
```

KEGIATAN 2

Dari data kegiatan 1, gunakan fungsi filter untuk mengambil hanya nilai integer saja dari data yang ada! Sehingga bisa didapatkan hasil seperti berikut:

```
['3', '3', '7', '21']
['5', '5', '8', '11']
['7', '1', '5', '33']
```

KEGIATAN 3

Buatlah program dengan mengimplementasikan map() dan filter() berdasarkan Data list pada kegiatan 1 modul 1, sehingga menghasilkan output seperti berikut :

```
[ ]  random_list = #seperti pada kegiatan 1 modul 1

    # Filter untuk memisahkan nilai float, int, dan string

    # Map untuk memetakan nilai int menjadi satuan, puluhan, dan ratusan
```

```
# Output
print("Data Float :")
print("Data Int :")
print("Data String :")
```

```
Data Float : (3.1, 2.7, 5.5)
Data Int :
{'ratusan': 1, 'puluhan': 0, 'satuan': 5}
{'ratusan': 7, 'puluhan': 3, 'satuan': 7}
{'ratusan': 4, 'puluhan': 1, 'satuan': 2}
Data String : ['Hello', 'Python', 'world', 'AI']
```

TUGAS PRAKTIKUM

Pilihlah salah satu diantara dua jenis soal berikut lalu buatlah program sekreatif kalian dengan syarat harus sesuai dengan ketentuan soal dan gunakan semaksimal mungkin materi yang sudah kalian pelajari di modul 3 ini.

Guna mengantisipasi duplikasi code, praktikan diijinkan untuk improvisasi kode maupun skenario selama tidak menyalahi paradigma fungsional dan tetap memenuhi program requirement yang diberikan.

SOAL A (INTERMEDIATE)

Diberikan studi kasus berupa pengolahan data mengenai daftar transaksi pembelian produk di sebuah toko.

Misalkan kita memiliki daftar transaksi pembelian produk seperti berikut:

```
[ ]  transactions = [
        {"product": "Buku", "price": 10000, "quantity": 2},
        {"product": "Pensil", "price": 2000, "quantity": 5},
        {"product": "Pensil", "price": 2000, "quantity": 3},
        {"product": "Pulpen", "price": 5000, "quantity": 2},
        {"product": "Buku", "price": 12000, "quantity": 1},
        {"product": "Pulpen", "price": 6000, "quantity": 4}
    ]
```

Studi kasus ini akan mencakup empat fungsi fungsional:

1. Menghitung total harga untuk setiap transaksi (product * price * quantity).
2. Memilih transaksi hanya untuk produk tertentu.
3. Menghitung total harga dari semua transaksi yang sudah dipilih.
4. Menghitung jumlah item yang terjual dari produk tertentu.

Lengkapi kode berikut, dan buatlah kode fungsi – fungsi yang diperlukan menggunakan Higher Order Function, Filter, Map, Reduce dan juga lambda sehingga menghasilkan output seperti dibawah ini* :

```python
from functools import reduce

# Fungsi untuk menghitung total harga transaksi menggunakan lambda

# Fungsi untuk menyaring transaksi hanya untuk produk tertentu dengan HoF

# Input nama produk yang ingin disaring
product_filter_input = input("Masukkan nama produk yang ingin disaring: ")

# Menggunakan filter() untuk menyaring transaksi sesuai input produk

# Menggunakan map() untuk menghitung total harga untuk setiap transaksi yang tersaring

# Menggunakan reduce() untuk menghitung total Pendapatan dari semua transaksi yang tersaring

# Menampilkan Output
```

```
Masukkan nama produk yang ingin disaring: Buku

Transaksi Pembelian Produk Buku:
{'product': 'Buku', 'price': 10000, 'quantity': 2}
{'product': 'Buku', 'price': 12000, 'quantity': 1}

Total Harga untuk Setiap Transaksi Produk Buku:
20000
12000

Total Pendapatan dari Transaksi Produk Buku: 32000

Total Jumlah Item Terjual dari Produk Buku: 3
```

Soal B (advance)

Diberikan studi kasus berupa pengolahan data mengenai daftar film dan menghitung beberapa statistik yang ditampilkan dengan ketentuan :

1. Terdapat dictionary data film seperti berikut :

```python
movies = [
    {"title": "Avengers: Endgame", "year": 2019, "rating": 8.4, "genre": "Action"},
    {"title": "Parasite", "year": 2019, "rating": 8.6, "genre": "Drama"},
    {"title": "Nomadland", "year": 2020, "rating": 7.3, "genre": "Drama"},
    {"title": "Dune", "year": 2021, "rating": 7.9, "genre": "Sci-Fi"},
    {"title": "Spider-Man: No Way Home", "year": 2021, "rating": 7.6, "genre": "Action"},
    {"title": "The French Dispatch", "year": 2021, "rating": 7.0, "genre": "Comedy"},
    {"title": "A Quiet Place Part II", "year": 2020, "rating": 7.4, "genre": "Horror"},
    {"title": "No Time to Die", "year": 2021, "rating": 6.8, "genre": "Action"},
    {"title": "The Power of the Dog", "year": 2021, "rating": 7.3, "genre": "Drama"},
    {"title": "Eternals", "year": 2021, "rating": 6.4, "genre": "Action"},
    {"title": "The Last Duel", "year": 2021, "rating": 7.0, "genre": "Drama"},
]
```

2. Terdapat pilihan beberapa fungsi untuk inputan yang mencakup :

a. Menampilkan data dari film yang dipilih (data berisi title, year, rating, dan genre)
b. Jumlah film berdasarkan genre
c. Rata-rata rating film berdasarkan tahun rilis
d. Film dengan rating tertinggi

Implementasikan fungsi – fungsi diatas menggunakan Higher Order Function, Filter, Map, Reduce dan juga lambda.

Output yang dihasilkan :

```
Pilih tugas yang ingin dilakukan:
1. Menghitung jumlah film berdasarkan genre
2. Menghitung rata-rata rating film berdasarkan tahun rilis
3. Menemukan film dengan rating tertinggi
4. Cari judul film untuk menampilkan informasi rating, tahun rilis, dan genre
5. Selesai
Masukkan nomor tugas (1/2/3/4/5): 1
Jumlah Film Berdasarkan Genre:
{'Action': 4, 'Drama': 4, 'Horror': 1, 'Comedy': 1, 'Sci-Fi': 1}

Pilih tugas yang ingin dilakukan:
1. Menghitung jumlah film berdasarkan genre
2. Menghitung rata-rata rating film berdasarkan tahun rilis
3. Menemukan film dengan rating tertinggi
4. Cari judul film untuk menampilkan informasi rating, tahun rilis, dan genre
5. Selesai
Masukkan nomor tugas (1/2/3/4/5): 2
Rata-rata Rating Film Berdasarkan Tahun Rilis:
{2019: 8.5, 2020: 7.35, 2021: 7.142857142857143}

Pilih tugas yang ingin dilakukan:
1. Menghitung jumlah film berdasarkan genre
2. Menghitung rata-rata rating film berdasarkan tahun rilis
3. Menemukan film dengan rating tertinggi
4. Cari judul film untuk menampilkan informasi rating, tahun rilis, dan genre
5. Selesai
Masukkan nomor tugas (1/2/3/4/5): 3
Film dengan Rating Tertinggi:

Informasi Film: Parasite
Rating: 8.6
Tahun Rilis: 2019
Genre: Drama

Pilih tugas yang ingin dilakukan:
1. Menghitung jumlah film berdasarkan genre
2. Menghitung rata-rata rating film berdasarkan tahun rilis
3. Menemukan film dengan rating tertinggi
4. Cari judul film untuk menampilkan informasi rating, tahun rilis, dan genre
5. Selesai
Masukkan nomor tugas (1/2/3/4/5): 4
Masukkan judul film yang ingin dicari: 5
Film dengan judul tersebut tidak ditemukan.
```

```
Pilih tugas yang ingin dilakukan:
1. Menghitung jumlah film berdasarkan genre
2. Menghitung rata-rata rating film berdasarkan tahun rilis
3. Menemukan film dengan rating tertinggi
4. Cari judul film untuk menampilkan informasi rating, tahun rilis, dan genre
5. Selesai
Masukkan nomor tugas (1/2/3/4/5): 4
Masukkan judul film yang ingin dicari: Dune

Informasi Film: Dune
Rating: 7.9
Tahun Rilis: 2021
Genre: Sci-Fi

Pilih tugas yang ingin dilakukan:
1. Menghitung jumlah film berdasarkan genre
2. Menghitung rata-rata rating film berdasarkan tahun rilis
3. Menemukan film dengan rating tertinggi
4. Cari judul film untuk menampilkan informasi rating, tahun rilis, dan genre
5. Selesai
Masukkan nomor tugas (1/2/3/4/5): 5
```

KRITERIA & DETAIL PENILAIAN TUGAS PRAKTIKUM

Kriteria	Soal A	Soal B	Program identik*
1. Program dapat berfungsi	10	20	0
2. Program menggunakan materi yang sudah dipelajari pada modul serta **mengimplementasikan paradigma pemrograman fungsional**	10	20	0
3. Menjelaskan program yang dibuat dengan **baik dan lancar**	20	20	20*
4. Menjelaskan implementasi materi modul dalam program yang dibuat **beserta alasan penggunaannya**	20	20	20*
5. Menjawab pertanyaan dari asisten dengan **baik dan lancar**	20	20	20*
Total Nilai (maksimal)	80	100	60*

*) Jika program identik dengan praktikan lain. Maka akan ada pengurangan nilai pada kedua praktikan.