VERSION 2.0

JULI , 2022

[PRAKTIKUM PEMROG. FUNGSIONAL]

MODUL 3 – Higher Order Functions pada python.

DISUSUN OLEH :
Andi Shafira Dyah K.

Mutaqhin Dean

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

PERSIAPAN SOFTWARE/APLIKASI

- Komputer/Laptop
- Sistem operasi Windows/Linux/Max 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.

MODUL 3 Fungsional 2022

Higher Order Functions pada python

↳ 1 cell hidden

PERSIAPAN MATERI

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

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. 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
```

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'
```

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

buat 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:

```
def cetakNIM(kodeJurusan):
  def cetak(nim):
    return "2021 1 " + kodeJurusan + " 0311 " + nim
  return cetak

nimMesin = cetakNIM('012')
print(nimMesin('123'))
```

```
     2021 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
```

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.

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 = ['Pavard', 'Lewandowski', 'Kimmich', 'Muller'] #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 0x7f85ddd59890>
```

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 Pavard
Hai Lewandowski
Hai Kimmich
Hai Muller
```

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

print(nilai_tambah7)
```

```
[0, 7, 14, 21, 28, 35, 42, 49, 56, 63]
```

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 element 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 map tidak bisa di print langsung. Harus melalui sebuah iterasi/looping, atau bisa juga langsung di parsing menjadi tipe data sequence lainnya:

```
print(tuple(div_by_5))
```

```
(35,)
```

3. Reduce

Fungsi reduce seperti fungsi map(), menerima dua argumen yakni sebuah fungsi dan iterable. Namun tidak seperti fungsi map, fungsi reduce harus diimport 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
```

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 multipel parameter/argumen menjadi pecahan banyak fungsi, tiap 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
```

Tugas Praktikum

Kegiatan 1

Modifikasi program yang telah anda kerjakan pada modul 2, kegiatan 1 agar mengimplementasikan materi Higher Order Function, Filter, Map, dan Reduce. NIM ganjil-genap tetap berlaku sesuai studi kasus tugas sebelumnya.

PS: Anda dapat menambahkan fungsionalitas lain diluar skenario jika memang diperlukan.

Kegiatan 2

Modifikasi percobaan 5 (currying) untuk mengkonversi beberapa hal berikut:

1. NIM Ganjil: konversi sejumlah hari lengkap dengan jamnya menjadi detik
2. NIM Genap: konversi sejumlah minggu dan hari lengkap dengan jamnya menjadi menit

berikut adalah input dan outputnya:

```
data1 = ["21 hari 20 jam 9 menit 20 detik",
         "19 hari 14 jam 0 menit 13 detik",
         "1 hari 1 jam 1 menit 1 detik"]

data2 = ["3 minggu 3 hari 7 jam 21 menit",
         "5 minggu 5 hari 8 jam 11 menit",
         "7 minggu 1 hari 5 jam 33 menit"]
```

```
outputData1 = [1886960, 1692013, 90061]
outputData2 = [35001, 58091, 72333]
```

Rubrik Penilaian

1. Kegiatan 1:
    - Membuat dan mengimplementasikan Higher Order Function (+20)
    - Mengimplementasikan Build-in HOF (map, filter dan reduce) (+15)
2. Kegiatan 2:
    - Program berjalan dengan baik sesuai ketentuan yang berlaku (+20)
3. Menjelaskan dengan **baik dan lancar** kepada asisten saat demo (+45)
    - Menjelaskan program yang dibuat
    - Menjelaskan yang sudah dipelajari dari modul
    - Menjawab pertanyaan dari asisten
4. Jika program identik dengan praktikan lain. Maka akan ada pengurangan nilai pada kedua praktikan.