VERSION 2.0

JULI , 2022

[PRAKTIKUM PEMROG. FUNGSIONAL]

MODUL 2 – Pengenalan Pure Function, Lambda Expressions, List Comprehension dan Generator

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

[PRAKTIKUM PEMROG. FUNGSIONAL]

PERSIAPAN MATERI

Praktikan diharapkan juga mempelajari dari sumber external mengenai bahasa python.

TUJUAN PRAKTIKUM

1. Praktikan diharap dapat mengenal pemrograman fungsional basic menggunakan Bahasa pemrograman python

PERSIAPAN SOFTWARE/APLIKASI

- Komputer/Laptop
- Sistem operasi Windows/Linux/Max OS/Android

MATERI POKOK

Pada pemrograman fungsional, modul juga dapat diakses melalalui google collab agar lebih interaktif : Modul 2

MODUL 2 Fungsional 2022

Pure Function, Lambda Expressions, List Comprehension, Generator

↳ 1 cell hidden

Pure Functions

Di modul 1 kita sudah belajar tetang tipe data tuple dan list, masih ingat kan kalau tuple itu sifatnya immutable atau kekal (tidak dapat diubah). Nah, dalam pemrograman fungsional, setiap fungsi yang dibuat sebaiknya bersifat immutable terhadap variabel lain di luar fungsi. Dengan kata lain, fungsi yang ada tidak boleh merubah dan juga tidak boleh terikat dengan variabel lain di luar fungsi. Fungsi tersebut dinamakan **pure function** atau fungsi murni. Hmm bingung ngga? Coba perhatikan contoh berikut:

```
luasKotak=10  #sebuah variabel diluar fungsi

def pure(x,y):  #deklarasi fungsi murni
  luasKotak = (x*y)
  return luasKotak

pure(5,6)
```

```
30
```

Contoh diatas merupakan pure function, dimana fungsi tersebut hanya bergantung pada argumen yang diberikan dan akan mengembalikan nilai yang sama dengan argumen yang sama. Ga percaya? coba kita panggil lagi deh..

```
print(luasKotak)
pure(5,6)
```

```
10
30
```

Nah kan tetep 30 outputnya. Dan juga, fungsi pure() tidak mempengaruhi/mengubah nilai variabel luasKotak yang telah dideklarasikan sebelumnya. Coba bandingkan dengan contoh berikut:

```
phi = 3.14
def impure(r):
  luasLing = phi*r*r
  return luasLing
impure(9)
```

```
254.34
```

Outputnya 254.34, kemudian perhatikan kode berikut:

```
phi = 3
impure(9)
```

```
243
```

Walau dengan argumen yang sama, outputnya berbedaa. Sehingga dapat dikatakan fungsi impure ini **tidak murni**. Kenapa begituu? Hal ini dikarenakan fungsi impure masih terikat dengan variabel lain di luar fungsi, yaitu "phi". Jadi apabila nilai phi diubah, outputnya ikut berubah walau dengan argumen yang sama

Untuk penjelasan lebih detail mengenai pure functions, kalian bisa melihat video ini

Lambda Expressions

Lambda expressions adalah fungsi pada python yang bersifat anonymous. Kalian tau kan anonymous ? yang identitasnya rahasia gitu. Nah kalau di lambda ini fungsi anonymous adalah fungsi yang bisa tidak memiliki nama. Jadi gini, saat kita buat sebuah fungsi di python, kita menggunakan keyword **def** kemudian diikuti dengan nama fungsi kan ? misal **def fungsi1:**, nah jika menggunakan lambda expressions, kalian tidak perlu melakukan seperti itu. cukup ketikkan keyword **lambda** diikuti dengan nama variabel/parameter yang digunakan sebagai input kemudian mendefinisikan apa yang ingin di return. Jadi gini strukturnya:

lambda namaparameter : return/aksi yang dilakukan di fungsi itu
contoh
lambda x : x + 2

pada lambda expressions diatas, kita menggunakan x sebagai input, kemudian memproses x

tersebut untuk ditambahkan dengan 2. Untuk lebih jelasnya kalian bisa simak baris kode dibawah.

```
def kali2(input):
  return input * 2

double = kali2(2)
print(double)
```

```
4
```

```
double = lambda input : input * 2
print(double(2))
```

```
4
```

Kedua cell diatas menampilkan perbedaan antara penggunaan fungsi biasa dan penggunaan lambda expressions. Terlihat lebih simpel menggunakan lambda expressions bukan? :D

List Comprehension

Salah fitur pada python yang cukup menarik dan dapat digunakan untuk pemrograman fungsional yang lebih optimal dan ringkas adalah list comprehension.

Kadang dalam beberapa kasus, mengharuskan kita untuk mengisi elemen dari list sebelum melakukan manipulasi data. Di modul 1 kita sudah belajar tetang list dan cara deklarasinya (list2 = [1, 2, 3, 4, 5]). Nah, untuk membuat daftar list yang lebih panjang, akan merepotkan jika kita harus menuliskan isi datanya satu persatu. Hal tersebut dapat diatasi dengan menggunakan perulangan for dengan append list pada tiap iterasinya. Coba perhatikan contoh berikut:

```
list = []
for i in range(40):
    list.append(i)

print(list)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, ...
```

Cara tersebut memang tidak salah, namun butuh banyak line code untuk mengisi sebuah list saja, dengan kata lain borosss. Coba bandingkan dengan contoh berikut:

```
list = [i for i in range(40)]
print(list)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, ...
```

Nahh kan tinggal 2 baris. Kode tersebut merupakan contoh sederhana penerapan list comprehension. Cara ini akan sangat bermanfaat khususnya jika kita perlu membuat list dengan daftar yang panjang tanpa harus menuliskan kode yang panjang pula(yang mana memerlukan banyak waktu dan resource pastinya). Makanya, sudahi cara lama, mari beralih menggunakan list comprehension :D

Tidak hanya itu, kita juga dapat menyisipkan kondisi serta ekspresi lain dalam list comprehension. Perhatikan contoh berikut:

```
#List berisi bilanga kuadrat dari angka genap dalam range 0-10
list = [i*i for i in range(10) if i % 2 == 0 ] print(list)
```

```
[0, 4, 16, 36, 64]
```

eitss, tipe string juga bisa kokk. Perhatikan contoh berikut:

```
#List dengan value konstan
names = ["sayang ayang walau gapunya ayang" for x in range(10)]
print(names)
```

```
['sayang ayang walau gapunya ayang', 'sayang ayang walau gapunya ayang', ...
```

```
#List dengan value semua karakter dalam sebuah
kalimat/string sentence = "sayang ayang walau gapunya ayang"
chars = [char for char in sentence]
print(chars)
```

```
['s', 'a', 'y', 'a', 'n', 'g', ' ', 'a', 'y', 'a', 'n', 'g', ' ', 'w', 'a', 'l', ...
```

Iterator dan Generator

Iterator

Sebelum belajar lebih jauh mengenai generator, pertama-tama kita kenalan dulu dengan yang namanya "iterator". Iterator merupakan objek yang berisi jumlah nilai yang dapat dihitung. Iterator adalah objek yang dapat diulangi, sehingga akan mengembalikan data, satu data per satu waktu.

Secara teknis, dalam Python, sebuah iterator adalah objek apa pun yang kelasnya memiliki fungsi next() (__next __() dalam Python 3), fungsi __iter __() yang menghasilkan return self, dan fungsi __init __().

Metode __iter __() digunakan untuk melakukan operasi (menginisialisasi dll), tetapi harus selalu kembali objek iterator sendiri.

Metode __next __() juga digunakan untuk melakukan operasi, dan harus kembali item berikutnya dalam urutan.

Kemudian untuk mencegah iterasi berjalan selamanya, kita dapat menggunakan pernyataan Stop Iteration. Coba perhatikan contoh berikut:

```
class Count:

    # fungsi init untuk proses inisialisasi
    # objek count secara default akan dimulai dari angka 1(start) hingga 5(stop)
    def __init__(self, start=1, stop=5):
        self.num = start
        self.n = stop

    def __iter__(self):
        return self

    def __next__(self):
        num = self.num
        if num > self.n:  # jika num > n/sudah mencapai nilai stop
          raise StopIteration # raise Stop Iteration untuk menghentikan iterasi
        self.num += 1
        return num

# membuat objek iterator dengan parameter (start=5, stop=8)
my_iter = Count(5,8)

# iterasi secara manual mengunakan method next()
print(next(my_iter)) # counting pertama --> print 5 (sesuai nilai start saat inisialisasi
print(next(my_iter)) # counting kedua --> print 6
```

```
5
6
```

```
# bisa juga dengan cara berikut:
# dimana next(obj) sama dengan obj.__next__()
print(my_iter.__next__())   # counting ketiga --> print 7
print(my_iter.__next__())   # counting keempat --> print 8 (nilai akhir/stop)
```

```
7
8
```

```
# Berikut ini akan memunculkan error karena item sudah habis
next(my_iter)
```

Karena Iterator adalah objek yang dapat diulangi, maka kita juga bisa menggunakan perulangan untuk mengakses nilai dari suatu iterator. Seperti contoh berikut:

```
# looping objek iterator dengan nilai default/tanpa
parameter for n in Count():
  print(n)
```

```
1
2
3
4
5
```

Suatu objek dikatakan iterable jika dari objek tersebut bisa kita buat iterator. Sebagian besar objek di Python seperti list, tuple, string, dan lain-lain adalah iterable. seperti contoh berikut:

```
# mendefinisikan list
my_list = [5, 4, 7,
3] print(my_list)
print(type(my_list))

# membuat iterator dengan iter()
my_iter = iter(my_list)
print(my_iter)
print(type(my_iter))
```

```
[5, 4, 7, 3]
<class 'list'>
<list_iterator object at 0x7f5816870e10>
<class 'list_iterator'>
```

Sebenarnya Iterator ini sudah diterapkan di dalam looping for, while dan list comprehension yang sudah kita pelajari sebelumnya, hanya saja tidak tampak secara langsung.

Paham kan gais? agak ribet yaa, soalnya iterator itu pake konsep OOP (dari awal bahas class

sama objek). Gak fungsional banget kan :D

Nahh untuk menghindari hal tersebut, khususnya di pemrograman fungsional, ketika kita menginginkan sebuah iterator, kita membuat sebuah generator.

Generator

Sederhananya, generator dalam python adalah fungsi yang mengembalikan sebuah objek iterator. Generator adalah cara cepat dan ringkas untuk membuat sebuah iterator.

Secara khusus, generator adalah subtipe dari iterator. Sehingga untuk pemanggilan objek generator dapat dilakukan dengan method **next()** seperti pada iterator.

Generator dibuat dengan memanggil fungsi yang memiliki satu atau lebih ekspresi yield, seperti contoh berikut:

```
# membuat fungsi count yang sama dengan class Count pada materi iterator
# fungsi dengan satu ekspresi yield
def count(start=1, stop=5):
  for i in range(start,stop+1):
    yield i

print(type(count)) # count adalah sebuah fungsi biasa yang mengembalikan nilai dengan keyw
```

```
<class 'function'>
```

Fungsi count dibuat untuk membuat sebuah iterasi dengan parameter default start = 1 dan stop = 5. Kemudian dibuat perulangan menggunakan for untuk mencetak kembalian dengan keyword yield pada range 1-6 (angka 6 didapatkan dari stop+1 berarti 5+1). Loh kenapa bukan return? kenapa malah yield?

keyword yield mirip dengan keyword return, bedanya yield mengembalikan objek generator. Coba perhatikan kode lanjutan untuk pemanggilan objek generator berikut

```
# membuat generator dengan memanggil fungsi count(start=5, stop=8)
my_gen = count(5,8) # saat fungsi count dipanggil,
print(type(my_gen)) # dia mengembalikan sebuah generator

# cara mengakses iterasi sama seperti mengakses iterator:
print('counting pertama dengan fungsi next():',next(my_gen)) # --> print 5 (sesuai nilai
print('counting kedua dengan method .__next__():',my_gen.__next__() ) # --> print 6
print('counting sisany dengan looping for:', end=" ")
```

```
for n in my_gen:
  print(n, end=" ")  # a --> print 7 & 8
```

```
<class 'generator'>
counting pertama dengan fungsi next(): 5
counting kedua dengan method .__next__(): 6
counting sisany dengan looping for: 7 8
```

Generator maupun iterator sama sama menggunakan fungsi next ataupun method ._ *next*_() untuk melakukan pemanggilan dan sama sama menyimpan progres iterasi. Namun coba bandingkan dengan kode iterator dalam kelas Count sebelumnya. Dengan tujuan dan output yang sama kita dapat membuat sebuah kode iterator menjadi lebih ringkas dan efisien dengan menggunakan Generator.

Gimana? paham kann gais? Coba kita beralih ke contoh yang lebih sederhana dari penggunaan generator. Perhatikan contoh berikut!

```
def my_generator(n):

    print('Pertama nih boss')
    yield n

    print('Kedua banget')
    yield n + 1

    print('Ketiga')
    yield n + 2


g = my_generator(1)
print(next(g))
```

```
Pertama nih boss
1
```

Loh kok cuma yang pertama aja yang diprint? Inilah kunci dari keyword yield yang kita gunakan.

Tidak seperti keyword return yang biasa kita gunakan untuk membuat fungsi pada umumnya, yang mana akan menghentikan(terminate) function secara keseluruhan.

Yield hanya akan menghentikan sementara(pause) function dan menyimpan semua state variable yang ada didalamnya, sehingga nantinya bisa dilanjutkan kembali. Perhatikan kode berikut

```
print(next(g))
```

```
Kedua banget
2
```

```
print(next(g))
```

```
Ketiga
3
```

Nah Kalau habis iterasinya gimana? yaudah selesaii

```
#Nih StopIteration
print(next(g))
```

Jadi object generator hanya bisa diiterasi untuk sekali saja. Kita harus membuat object generator baru untuk melakukan iterasi kembali.

Generator Expression

Selain itu, generator juga dapat dibuat dengan cara yang lebih ringkas lagi tanpa harus membuat fungsi yang menggunakan keyword yield terlebih dahulu. Kita menyebutnya dengan istilah **generator expression**. Coba perhatikan contoh berikut

```
count = (i for i in range(5,9))

for i in count:
    print(i, end=" ")
```

```
5678
```

kode diatas adalah cara yang sama untuk membuat iterasi dengan range 5-9 yang akan mencetak nilai 5-8

Tidak hanya itu, kita juga dapat menyisipkan kondisi serta ekspresi lain sama seperti saat kita membuat list comprehension. Perhatikan contoh berikut:

```
generator_expression = (i for i in range(11) if i % 2 == 0)

print(generator_expression)
```

```
<generator object <genexpr> at 0x7fc28866ced0>
```

Namun berbeda dengan list comprehension, kita tidak bisa mencetak nilainya secara langsung menggunakan fungsi print. Kenapa begitu??? karena Generator merupakan implementasi dari **Lazy Evaluation**. Apa lagi ini Lazy Evaluation? :(

Lazy Evaluation merupakan strategi evaluasi yang menunda evaluasi ekspresi hingga nilainya diperlukan dan yang juga menghindari evaluasi berulang. Pada lazy evaluation, dia hanya melakukan evaluasi pada variabel yang dibutuhkan saja, yaah namanya juga malass, ngapain dieval semua kalau tidak dibutuhkan ya kann..

Namun menjadi malas bukan berarti bad attitude dalam pemrograman fungsional, dia justru dapat meningkatkan efisiensi kode dan menghemat banyak sumber daya. Inilah salah satu keunggulan dari paradigma fungsional.

Untuk lebih memahami lazy evaluation, coba kita bandingkan generator dengan list comprehension (yang tidak lazy). cekidott

Generator vs List Comprehension

Pertama tama, coba perhatikan cara membuat list comprehension dengan generator expresion berikut dan temukan apa perbedaannya!

```
list_comp = [i for i in range(10000)]
gen_exp = (i for i in range(10000))
```

Nah dari contoh di atas sudah kelihatan kan bedanya Iyess, kalau List Comprehension dia pakai kurung siku, sedangkan generator pakai kurung biasa, sehingga List Comprehension akan menghasilkan data squence list dan generator menghasilkan sebuah objek iterator

Selain format penulisan, generator menghasilkan satu item pada satu waktu dan menghasilkan item hanya saat dibutuhkan. Padahal, dalam List Comprehension, Python menyimpan memori untuk seluruh isi list. Dengan demikian dapat dikatakan bahwa ekspresi generator lebih hemat memori daripada List Comprehension. Ga percaya? coba cek kode berikut

```
from sys import getsizeof

#memberikan size untuk list comprehension
x = getsizeof(comp)
print("x = ", x)
```

```
#memberikan size untuk generator expression
y = getsizeof(gen)
print("y = ", y)
```

```
x =  87632
y =  128
```

Dapat dilihat pada output yang dihasilkan, bahwa generator lebih hemat memori. Tapi apakah hemat waktu eksekusi juga? skuy buktikann

```
import timeit

print(timeit.timeit('''list_com = [i for i in range(100) if i % 2 == 0]''', number=1000000
```

```
7.81876667799952
```

```
import timeit

print(timeit.timeit('''gen_exp = (i for i in range(100) if i % 2 == 0)''', number=1000000)
```

```
0.4818928770000639
```

Iyess, generator menang telak :) jadi dapat disimpulkan bahwa generator lebih hemat memori serta lebih efisien waktu.

Nah mungkin cukup itu aja ya untuk di modul 2 ini. Sekarang waktunya tugas praktikum, hehehe

Tugas Praktikum

Kegiatan 1

NIM Ganjil

Modifikasi program dengan dua akun sebelumnya (tugas modul 1), yaitu admin dan anggota perpustakaan mini dengan ketentuan :

1. Admin dapat melakukan CRUD (create, read, update, delete) pada buku yang tersedia maupun yang sedang dipinjam.
2. Admin dapat melakukan CRUD (create, read, update, delete) pada akun anggota

3. Terdapat 4 kolom untuk data anggota, yakni ID (untuk mempermudah, id sebaiknya berurut dari 0 dst.), Nama, riwayat pinjaman, lama keanggotaan(dihitung mulai dari hari pertama mendaftar dalam hitungan hari)
4. Buku yang sudah dipinjam tidak dapat dipinjam lagi oleh anggota lain kecuali sudah dikembalikan
5. Anggota yang sama hanya boleh meminjam 3 buku setiap harinya
6. Akun anggota bisa menampilkan point apresiasi keanggotaan((jumlah buku yang pernah dipinjam+lama keanggotaan)/2) dan daftar riwayat peminjaman buku
7. Buatlah program sekreatif mungkin dengan mengimplementasikan materi yang sudah dipelajari dalam modul ini
8. Kode yang identik akan mendapat pengurangan nilai

NIM GENAP

Modifikasi program dengan dua akun sebelumnya (tugas modul 1), akun teacher dan akun student dengan ketentuan :

1. Guru dapat melakukan CRUD (create, read, update, delete) pada nilai murid.
2. Guru dapat melakukan CRUD (create, read, update, delete) pada akun murid
3. Terdapat 4 kolom untuk data student, yakni ID (untuk mempermudah, id sebaiknya berurut dari 0 dst.), Nama, nilai ulangan harian, nilai UAS.
4. Guru diberi batasan untuk mengadakan ulangan harian maksimal 3 kali sehari
5. Akun student bisa menampilkan nilai akhir dari student itu sendiri dan daftar nilai ulangan harian + UAS
6. Nilai akhir diperoleh dari nilai (nilai ulangan harian + UAS) / 2
7. Buatlah program sekreatif mungkin dengan mengimplementasikan materi yang sudah dipelajari dalam modul ini
8. Kode yang identik akan mendapat pengurangan nilai

Kegiatan 2

NIM GANJIL

Buatlah sebuah program yang dapat menampilkan sebuah deret fibonacci dengan ketentuan:

1. Menggunakan Fungsi Generator
2. Jumlah bilangan merupakan inputan user

3. Buatlah dengan kode seringkas dan seefektif mungkin

NIM GENAP

Buatlah sebuah program yang dapat menampilkan sebuah deret dengan ketentuan:

1. Menggunakan Generator Expression
2. Deret berisi bilangan prima dibawah angka 100
3. Buatlah dengan kode seringkas dan seefektif mungkin

Rubrik Penilaian

Kegiatan 1 :

1. Program dapat berfungsi (max. 10)
2. Program harus sesuai dengan ketentuan yang diberikan (max. 20)
3. Menjelaskan dengan baik dan lancar kepada asisten saat demo (max. 30):
    - Menjelaskan program yang dibuat (10 poin)
    - Menjelaskan yang sudah dipelajari dari modul (10 poin)
    - Menjawab pertanyaan dari asisten (10 poin)

Kegiatan 2 :

1. Program dapat berfungsi (max. 10)
2. Program harus menggunakan materi yang sudah dipelajari pada modul (max. 15)
3. Menjelaskan dengan baik dan lancar kepada asisten saat demo (max. 15):
    - Menjelaskan program yang dibuat (5 poin)
    - Menjelaskan yang sudah dipelajari dari modul (5 poin)
    - Menjawab pertanyaan dari asisten (5 poin)