VERSION 1.2

AGUSTUS , 2023

[PRAKTIKUM PEMROG. FUNGSIONAL]

MODUL 4 – Pengenalan Inner Functions, Closure, dan decorator dalam studi kasus Grafika Komputer Sederhana

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

[PRAKTIKUM PEMROG. FUNGSIONAL]

PERSIAPAN MATERI

1. Praktikan diharapkan telah memahami materi pada modul-modul sebelumnya.
2. Praktikan diharapkan juga mencari sumber belajar eksternal mengenai bahasa python

TUJUAN PRAKTIKUM

- Praktikan diharapkan mampu mengelola First Class Function secara fleksibilitas berdasarkan paradigma pemrograman fungsional

TARGET MODUL

Penguasaan materi:

1. Inner Function
2. Closure
3. Decorator

PERSIAPAN SOFTWARE/APLIKASI

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

MATERI POKOK

Pada pemrograman fungsional, modul juga dapat diakses melalui google collab agar lebih interaktif :

Modul 4

1. Inner Function

Inner functions, juga dikenal sebagai nested functions. Inner functions adalah fungsi yang didefinisikan di dalam fungsi lain. Pada dasarnya, sebuah fungsi dibuat sebagai inner function untuk memisahkan/membatasi dari semua yang terjadi di luar fungsi induk (outer function). Artinya, inner function hanya bisa mengakses variabel dari lingkup fungsi induk, dan tidak dapat berinteraksi dengan variabel diluar fungsi induk.

```
def OuterFunc():
    x = 2
    y = 5
    print("Ini adalah fungsi induk")
    def InnerFunc():
        z = 3
        n = x*y*z
        print("Ini adalah fungsi Inner")
        return n
    return InnerFunc()

OuterFunc()
```

```
Ini adalah fungsi induk
Ini adalah fungsi Inner
30
```

Dalam contoh diatas function InnerFunc merupakan inner function dari fungsi OuterFunc. Kalian pasti sudah banyak menemukan dan menggunakan inner function pada modul 4 sebelumnya. Tapi pembahasan inner function disini penting untuk dapat memahami materi selanjutnya, yaitu closure dan decorator. Kita akan banyak menggunakan inner function nantinya.

2. Ruang Lingkup Variabel

Sebelum membahas tentang closure, kita perlu belajar tentang ruang lingkup variabel di Python. Ruang lingkup variabel mengacu pada area di mana kita dapat melihat atau mengakses variabel. Ruang lingkup variabel ditentukan oleh lokasi di mana variabel ditetapkan dalam kode sumber. Umumnya, variabel dapat ditetapkan di tiga tempat berbeda, sesuai dengan tiga lingkup yang berbeda yaitu :

- Global Scope : Ketika variabel didefinisikan di luar semua fungsi. Variabel global dapat diakses oleh semua fungsi dalam file.
- Local Scope : Ketika variabel didefinisikan di dalam fungsi, itu adalah lokal untuk fungsi itu.Variabel lokal hanya dapat diakses di dalam fungsi di mana variabel tersebut didefinisikan.

- NonLocal Scope : Di python, variabel nonlocal merujuk ke semua variabel yang dideklarasikan dalam inner function. Lingkup lokal dari variabel nonlokal tidak didefinisikan. Ini berarti bahwa variabel tidak ada dalam lingkup lokal maupun dalam lingkup global.

Kita akan coba bahas masing-masing scope variabel diata. Perhatikan contoh berikut:

```
#Contoh 1: variabel global
x = "Variabel global"
def iniFungsi():
   print(x, "ini dapat diakses dari dalam fungsi")

#pemanggilan fungsi
iniFungsi()

print(x, "juga dapat diakses di luar fungsi ")
```

```
    Variabel global ini dapat diakses dari dalam fungsi
    Variabel global juga dapat diakses di luar fungsi
```

```
#Contoh 2: lain penerapan variabel global
x = 5
def iniFungsiJuga():
 x = x * 2
 print(x)
iniFungsiJuga()
```

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
<ipython-input-3-e4f382b38db4> in <cell line: 6>()
      4   x = x * 2
      5   print(x)
----> 6 iniFungsiJuga()
      7
      8 # kode diatas akan menghasilkan eror:

<ipython-input-3-e4f382b38db4> in iniFungsiJuga()
      2 x = 5
      3 def iniFungsiJuga():
----> 4   x = x * 2
      5   print(x)
      6 iniFungsiJuga()

UnboundLocalError: local variable 'x' referenced before assignment
```

LOHH KOK ERROR? Iya soalnya variabel "x" belum secara eksplisit dideklarasikan sebagai variabel global. Akibatnya, Python memperlakukannya sebagai variabel lokal dan membatasi semua upaya untuk memperbarui nilainya (sebagaimana paradigma fungsional tidak memperkenankan adanya perubahan terhadap data diluar fungsi). Namun, bukan berarti tidak boleh diakses dan dimanipulasi, hanya sebaiknya dihindari, dengan alasan paradigma fungsional(ingat kembali tujuan penggunaan paradigma pemrograman).

Untuk dapat mengakses dan memanipulasi variabel global, kita perlu menggunakan kata kunci "global" di dalam sebuah fungsi. Perhatikan contoh berikut

```
#Nah begini baru bener xixi
x = 5
def iniFungsiJuga():
 global x
 x = x * 2
 print(x)
iniFungsiJuga()
```

```
    10
```

```
#Contoh 3: variabel lokal
x = 5
def foo():
 global x
 y = 10 #y merupakan variabel lokal yang hanya bisa diakses di fungsi foo
 z = x * 2 + y #z juga merupakan variabel lokal
 print(z)
foo()
```

```
    20
```

Tentang variabel global dan lokal diatas mungkin teman-teman sudah terbiasa dan mengenalinya serta tidak jauh berbeda dengan paradigma pemrograman sebelumnya. Tapi **di pemrograman fungsional, sebaiknya kita menghindari penggunaan variabel global**. Tahu kan kenapa??? YA, karena itu menyalahi konsep fungsi murni (pure func) pada paradigma fungsional. Materi diatas hanya sekedar untuk pembahasan lingkup variabel saja. Sekarang kita akan beranjak ke lingkup variabel non-lokal.

Nonlocal scope hampir mirip seperti global scope, hanya saja variabel non-lokal muncul saat kita memanfaatkan inner function (yang mana belum pernah kita temui pada paradigma program sebelumnya: OOP maupun Struktural). Di Python, variabel non-lokal defaultnya adalah bersifat read-only (seperti variabel global). Untuk dapat memodifikasinya, kita harus mendeklarasikannya sebagai variabel non-lokal (menggunakan keyword nonlocal). Coba perhatikan contoh kode berikut:

```
#Contoh 4: variabel non-lokal

tinggi = 20 # ini variabel global
def volumeBalok():
    tinggi = 10  # ini variabel nonlocal
    print("value variabel tinggi awal", tinggi)
    def alas():
      nonlocal tinggi #nama variabel referensi di lingkup atasnya
      tinggi = 4      #timpa valuenya
      # berikut adalah variabel local semua:
      panjang = 5
```

```
      lebar = 2
      volume = panjang*lebar*tinggi
      print("volume kubus adalah : ",volume)
    # panggil inner function
    alas()
    # Print variabel nonlocal
    print("value variabel tinggi akhir",tinggi)

volumeBalok()
# Print variabel global
print("value variabel tinggi global",tinggi)
```

```
# Tulis kembali kode diatas dan jalankan
```

Apa yang terjadi jika kita **tidak** menuliskan kata kunci nonlocal? Maka, secara otomatis python akan menganggapnya sebagai variabel lokal fungsi yang otomatis akan dihapus dari memory saat fungsi telah selesai dieksekusi. Coba kalian hapus atau komen baris program yang mengandung kata kunci nonlocal pada kode diatas dan perhatikan hasilnya!

```
#Contoh 5: modifikasi variabel non-lokal tanpa keyword nonlocal

tinggi = 20 # ini variabel global
def volumeBalok():
    tinggi = 10  # ini variabel nonlocal
    print("value variabel tinggi awal", tinggi)
    def alas():
      #nonlocal tinggi #nama variabel referensi di lingkup atasnya
      tinggi = 4      #variabel ini tidak ada bedanya dengan
                      #variabel lokal dibawahnya (panjang dan lebar)
      # berikut adalah variabel local semua:
      panjang = 5
      lebar = 2
      volume = panjang*lebar*tinggi
      print("volume kubus adalah : ",volume)
    # panggil inner function
    alas()
    # Print variabel nonlocal
    print("value variabel tinggi akhir",tinggi)

volumeBalok()
# Print variabel global
print("value variabel tinggi global",tinggi)
```

```
#Tulis kembali kode diatas dan jalankan
```

Sudahkah kalian temukan perbedaannya dimana? Ya, tinggi awal dan tinggi akhir sekarang tidak ada beda. Hal ini dikarenakan inner function alas() tidak lagi dapat mengubah nilai variabel nonlocal tinggi (seperti pada program #contoh 2 yang tidak menggunakan keyword global). Apakah kalian sudah mulai memahami maksud dari penggunaan nonlocal scope?

Coba kalian perhatikan contoh berikut ini agar lebih memahami ruang lingkup variabel:

```
x = 1
y = 5
def func():
```

```
    global y
    x = 2
    y += 1
    z = 10
    print("Lokal Variabel x =", x)
    print("Global Variabel y =", y)
    print("Lokal Variabel z =", z)

func()
print("Global variable x =", x)
print("Global variable y =", y)
```

```
Lokal Variabel x = 2
Global Variabel y = 6
Lokal Variabel z = 10
Global variable x = 1
Global variable y = 6
```

Misalnya didalam contoh diatas, telah didefinisikan dua variabel global x dan y. Ketika x dideklarasikan ulang di dalam fungsi func( ), variabel lokal baru dengan nama yang sama, itu tidak akan mempengaruhi variabel global x. Namun, dengan menggunakan kata kunci global kita dapat mengakses variabel global y di dalam func( ). z adalah variabel lokal func( ) dan tidak dapat diakses di luarnya. Setelah keluar dari func( ), variabel ini tidak ada dalam memori, sehingga tidak dapat diakses lagi. Dengan kata lain:

1. Jika Anda menetapkan ulang variabel global di dalam fungsi, variabel lokal baru dengan nama yang sama akan dibuat dan diperbarui, sehingga variabel global dikatakan samar samar oleh variabel lokal ini.
2. Setiap perubahan pada variabel lokal ini di dalam fungsi, tidak akan mempengaruhi variabel global. Jadi jika Anda ingin mengubah nilai global di dalam fungsi, Anda harus menggunakan kata kunci global di Python untuk ini.

Hal tersebut diatas berlaku juga untuk scope nonlocal. Di Python, semuanya adalah objek, dan variabel adalah referensi ke objek nya. Ketika kita meneruskan variabel ke fungsi, Python meneruskan salinan referensi ke objek yang dirujuk variabel. Ini tidak mengirim objek atau referensi asli ke fungsi. Jadi baik referensi asli dan referensi yang disalin atau yang diterima fungsi sebagai argumennya mengacu pada objek yang sama.

Sudah cukup paham dengan lingkup variabel kan...

Sekarang bagian mana yang menjelaskan tentang closure?

3. Closure

Closure digunakan untuk menghindari penggunaan variabel global (yang sangat tidak disarankan dalam paradigma fungsional) dan menyediakan fungsi penyembunyian data (data hiding) sebagai solusi berorientasi objek terhadap permasalahan.

Dalam paradigma fungsional, biasanya hanya sedikit (umumnya satu buah) metode yang hendak diimplementasikan pada sebuah kelas, sehingga kita lebih baik menggunakan closure daripada mendefinisikan kelas. Tapi, ketika banyak atribut dan metode yang akan dibuat, kita lebih baik menggunakan kelas pada paradigma OOP daripada fungsional.

coba perhatikan kode berikut:

```
def print_msg(msg):
   # This is the outer enclosing function
   def printer():
       # This is the nested function
       print(msg)
   printer()  # call the nested function

# Now let's try calling this function.
# Output: Hello
another = print_msg("Hello")
print(type(another))
print(another)
```

```
Hello
<class 'NoneType'>
None
```

Bisa kita lihat pada contoh di atas, bahwa fungsi printer() dapat mengakses variabel non-lokal msg dari fungsi pembungkusnya/fungsi outer.

Kode diatas adalah contoh inner function biasa. Dimana fungsi print_msg() adalah sebuah fungsi yang memiliki fungsi inner printer() dan kemudian memanggil fungsi printer. Pemanggilan fungsi printer() akan menjalankan perintah print(msg) sehingga kata Hello muncul di output.

Karena fungsi print_msg tidak me-return apapun, maka statement another = print_msg("Hello") akan memberikan nilai None pada variable another. Hal ini terbukti saat kita melakukan print.

Belum closure dan bukan juga HoF. Mari kita modifikasi kode diatas untuk menjadikannya sebagai HoF untuk memahami apa itu closure:

```
def print_msg(msg):
   # This is the outer enclosing function
   pesan = msg # nonlocal
   def printer():
       # This is the nested function
       print(pesan)
   return printer  # return the nested function

# Now let's try calling this function.
# Output: Hello
```

```
another = print_msg("Hello")
print(type(another))
print(another)
another()
```

```
<class 'function'>
<function print_msg.<locals>.printer at 0x7d5132038ee0>
Hello
```

Apanya yang berbeda? Selain adanya tambahan nonlocal variabel, output nya jadi beda bukan.

Perhatikan cara kita memperlakukan fungsi inner printer(). Bisakah kalian membedakan cara **memanggil** dan **me-return** fungsi?

Satu perlakuan berbeda tersebut telah merubah tipe dari fungsi print_msg. Yang sebelumnya None karena tidak me-return apapun, sekarang menjadi HoF karena me-return sebuah fungsi inner.

Tentang closure, pada pemanggilan fungsi another(), isi dari variabel msg, yaitu 'Hello' masih tetap diingat meskipun kita sudah selesai dengan pemanggilan fungsi print_msg()/sudah return. Padahal pada fungsi biasa, seharusnya variabel fungsi akan terhapus begitu return/eksekusi terhadap fungsi selesai.

Nilai yang ada pada scope fungsi pembungkus masih diingat meskipun variabel sudah di luar scope fungsi, atau bahkan setelah fungsi tersebut dihapus seperti pada contoh berikut:

```
del print_msg
print_msg("Hello")
#NameError: name 'print_msg' is not defined
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-10-58827fedad6e> in <cell line: 2>()
      1 del print_msg
----> 2 print_msg("Hello")
      3 #NameError: name 'print_msg' is not defined

NameError: name 'print_msg' is not defined
```

Kita sudah tidak lagi bisa memanggil fungsi print_msg() setelah kita hapus. tapi pemanggilan fungsi another() berikut akan tetap mengembalikan hasil dan mengingat nilainya meski fungsi utamanya telah dihapus

```
another()
```

```
Hello
```

Python Closures adalah inner functions yang terlampir dalam outer function. Dimana Closures dapat mengakses variabel dalam scope outer function bahkan setelah fungsi luar menyelesaikan eksekusinya.

Kriteria yang harus ada untuk membuat closure di Python adalah sebagai berikut:

- Kita harus memiliki fungsi bersarang (fungsi dalam fungsi)
- Fungsi yang di dalam harus merujuk ke variabel fungsi pembungkusnya
- Fungsi pembungkus harus mengembalikan (me-return) fungsi yang di dalamnya.

Closure ini banyak dipergunakan dalam decorator di Python.

4. Decorations

Decorator merupakan komponen penting dalam Python yang mendukung fleksibilitas dalam manipulasi fungsi. Per definisi, decorator adalah fungsi yang melakukan operasi terhadap fungsi lain dan memodifikasi perilakunya tanpa harus mengubah secara eksplisit.

Decorator bekerja menggunakan prinsip metaprogramming karena ia akan memodifikasi bagian program lainnya pada saat eksekusi. Secara konsep, decorator menggunakan metode inner function dan Python closure. Selain dalam Decorator, Closure juga penting untuk pemrograman yang asinkron yang efektif dengan callback, dan untuk pengkodean dengan gaya fungsional.

Sesuai namanya, dekorator adalah sebuah fungsi yang dapat dipanggil dan juga yang mengambil fungsi lain sebagai argumen (fungsi yang dihiasi). Dekorator ini dapat melakukan beberapa pemrosesan dengan fungsi yang dihiasi, dan mengembalikannya atau menggantinya dengan fungsi lain atau objek yang dapat dihubungi.

Dengan prasyarat ini, mari kita lanjutkan dan membuat dekorator sederhana yang akan mengubah kalimat menjadi huruf besar. Kami melakukan ini dengan mendefinisikan closure di dalam fungsi tertutup. Seperti yang Anda lihat sangat mirip dengan fungsi di dalam fungsi lain yang kita buat sebelumnya.

Contoh Progam 1 (Single decorator to Single Functions):

```
def halo_decorator(fungsi):
   def halo():
       print('Halo')
       fungsi()
       print('Apa kabar?')
   return halo

def myname():
 print("Monty Python")

decorate = halo_decorator(myname)
decorate()
```

```
Halo
Monty Python
Apa kabar?
```

Fungsi dekorator diatas mengambil fungsi sebagai argumen, ini sama seperti materi HoF kita sebelumnya. Oleh karena itu, perlu untuk mendefinisikan fungsi myname() dan meneruskannya ke dekorator. Mengingat sebelumnya bahwa kita dapat menetapkan fungsi ke variabel, disini Kita akan menggunakan cara itu untuk memanggil fungsi dekorator. Seperti ini:

```
decorate = halo_decorator(myname)
decorate()
```

```
    Halo
    Monty Python
    Apa kabar?
```

Namun, Python menyediakan cara yang jauh lebih mudah bagi kita untuk menerapkan dekorator. Kita cukup menggunakan simbol @ sebelum fungsi yang ingin kita hias. Hal ini juga akan memudahkan kita untuk membedakan apakah fungsi yang dibuat hanyalah sebuah HoF biasa ataukah sebuah dekorator. Mari kita coba dalam praktik berikut ini:

```
@halo_decorator
def myname():
 print("Monty Python")
myname()
```

```
    Halo
    Monty Python
    Apa kabar?
```

Contoh Program 2 (Membuat Decorator untuk Fungsi dengan Argumen):

Bagaimana jika kita ingin menginputkan nama kita untuk dicetak pada fungsi myname? Apakah cukup dengan menambahkan argumen nama pada fungsi myname seperti berikut? Mari kita coba

```
@halo_decorator
def myname(nama):
 print(nama)


myname("Python 3")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-953e719d3cff> in <cell line: 5>()
      3   print(nama)
      4
----> 5 myname("Python 3")
      6 #TypeError: halo() takes 0 positional arguments but 1 was given

TypeError: halo_decorator.<locals>.halo() takes 0 positional arguments but 1 was given
```

Ternyata tidak sesederhana itu! Why?? Karena fungsi inner pada dekorator kita (fungsi `halo()`) tidak memiliki parameter, maka kita juga tidak bisa menambahkan parameter untuk fungsi yang akan kita dekorasi.

Jika kita ingin menambahkan parameter untuk fungsi yang akan kita dekorasi, maka kita juga perlu memodifikasi fungsi dekorator kita agar bisa menerima input argumen dan memprosesnya. Bagaimana caranya? perhatikan kode berikut:

```
def halo_decorator(fungsi):
 #penambahan argumen agar bisa memproses fungsi dengan satu atau lebih argumen
   def halo(*args, **kwargs):
       print('Halo')
       fungsi(*args,**kwargs)
       print('Apa kabar?')
   return halo

@halo_decorator
def myname(nama):
 print(nama)

myname("Google Collab")
```

```
Halo
Google Collab
Apa kabar?
```

Supaya decorator`@halo_decorator` dapat mendekorasi fungsi dengan argumen, perlu ditambahkan argumen `*args` dan `**kwargs` di dalam fungsi `halo()` pada decorator.

Penambahan argumen tersebut menyebabkan fungsi `halo()` bisa menerima sejumlah argumen. Dengan demikian, `@halo_decorator` bisa mendekorasi fungsi `myname()` yang menerima input berupa argumen nama.

Mari kita coba juga dengan multi argumen:

```
@halo_decorator
def whoami(firstName, lastName):
 print(firstName, lastName)

whoami("Petter", "Parker")
```

```
Halo
Petter Parker
Apa kabar?
```

```
@halo_decorator
def multiFungsi(x, y, z):
 print(x, y, z, sep="")

multiFungsi(1, 2, 3)
```

```
Halo
123
Apa kabar?
```

Kesimpulan

Di Python, fungsi dekorator memainkan peran subkelas Dekorator, dan Inner Function yang dikembalikan sebagai instance dekorator. Fungsi yang dikembalikan membungkus fungsi yang akan di decorate, yang dianalogikan dengan komponen dalam pola desain.

Fungsi yang dikembalikan bersifat transparan karena sesuai dengan antarmuka komponen dengan menerima argumen yang sama. Ini meneruskan panggilan ke komponen dan dapat melakukan tindakan tambahan baik sebelum atau sesudahnya. Meminjam dari kutipan sebelumnya, kita dapat mengadaptasi kalimat terakhir untuk mengatakan bahwa "Transparansi memungkinkan Anda menumpuk dekorator secara rekursif, sehingga memungkinkan jumlah perilaku tambahan yang tidak terbatas." Itulah yang memungkinkan dekorator bertumpuk untuk bekerja.

LATIHAN PRAKTIKUM

KEGIATAN PERCOBAAN 1 (penerapan *closure* dengan currying):

- mengakses nonlokal variabel di dalam inner function

```
# curry function
def perkalian(a):
  def dengan(b):
    return a * b
  return dengan # mengembalikan fungsi inner operator
```

```
#Tulis kembali kode diatas kemudian panggil(run) dengan dua cara:
# 1. HOF
# 2. currying
```

KEGIATAN PERCOBAAN 2 (penerapan single decorators to single function):

```
  def uppercase_decorator(function):
      def wrapper():
          func = function()
          #make_uppercase = func.upper()
          return func.upper()

      return wrapper
```

```
#Tulis kembali kode diatas dan tambahkan sebagai decorator pada fungsi say_hi
berikut

def say_hi():
   return 'hello there'
```

```
say_hi()
```

```
   HELLO THERE
```

KEGIATAN PERCOBAAN 3 (penerapan *multiple decorators to single function*):

```
def title_decorator(function):
    def wrapper():
        func = function()
        make_title = func.title()
        print(make_title + " " + "-Data is convert to title case")
        return make_title
    return wrapper

def split_string(function):
    def wrapper():
        func = function()
        splitted_string = func.split()
        print(str(splitted_string) + " " + "- Then Data is splitted")
        return splitted_string
    return wrapper
```

```
#Tulis kembali fungsi decorator diatas dan tambahkan keduanya sebagai decorator
#pada fungsi say_hi berikut

def say_hi():
   return 'hello there'
say_hi()
```

```
   Hello There -Data is convert to title case
   ['Hello', 'There'] - Then Data is splitted
   ['Hello', 'There']
```

KEGIATAN PERCOBAAN 4 (transformasi)

- Buatlah 3 fungsi untuk menghitung tranformasi secara tranlasi, dilatasi, dan rotasi. Dengan contoh kasus sebagai berikut:

Sebuah titik P memiliki koordinat (3, 5).

1. Jika titik tersebut mengalami translasi dengan tx = 2 dan ty = -1, tentukan koordinat titik baru setelah translasi
2. Jika titik tersebut mengalami dilatasi dengan sx = 2 dan sy = -1, tentukan koordinat titik baru setelah dilatasi
3. Jika titik tersebut mengalami rotasi dengan sudut = 30 derajat, tentukan koordinat titik baru setelah rotasi.

Agar lebih mudah saat mengerjakan, mari review kembali materi tranformasi pada link

```
#jangan lupa untuk ngoding secara fungsional ya
#kalian bisa memanfaatkan inner fucntion(bahkan fungsi lambda) dan closure disini

#contoh output:
```

```
   Koordinat setelah translasi: (5, 4)
   Koordinat setelah dilatasi: (6, -5)
```

```
Koordinat setelah rotasi: (0.09807621135331646, 5.830127018922194)
```

KEGIATAN PERCOBAAN 5 (mencari persamaan garis melalui 2 titik)

- Diketahui dua titik A(1, 3) dan B(5, 9). Tentukan persamaan garis lurus yang melalui kedua titik tersebut.

Bentuk Persamaan Garis Lurus secara eksplisit dituliskan berdasarkan rumus fungsi berikut:

$$y = mx + c$$

dimana m adalah gradien/kemiringan garis dan c adalah konstanta (bisa juga disebut sebagai y-intercept/titik perpotongan sumbu y).

Untuk mencari suatu persamaan garis lurus, kamu bisa menggunakan rumus diatas. tentunya kalian perlu menghitung nilai m dan c terlebih dahulu untuk dapat melengkapi persamaan. Agar lebih paham cara perhitungan, kalian bisa mengunjungi link

```python
# lengkapi kode berikut untuk mendapatkan persamaan garis lurus bagi NIM GANJIL

def point(x,y):
 return x,y

def line_equation_of(p1, p2):
   #TODO 1: gunakan inner function dan closure untuk menghitung nilai M

   M = #TODO 2: panggil fungsi inner untuk mendapatkan nilai M
   C = #TODO 3: tulis rumus untuk mendapatkan nilai C disini
   return f"y = {M:.2f}x + {C:.2f}"

print("Persamaan garis yang melalui titik A dan B:")
print(line_equation_of(point(4,-2),point(-1,3)))
#ubah nilai input dengan 4 digit nim akhir kalian
#dan lakukan perhitungan manual sesuai rumus yang kalian gunakan dalam fungsi
#untuk membuktikan bahwa output sudah benar
```

```
Persamaan garis yang melalui titik A dan B:
y = 1.50x + 1.50
```

```python
# lengkapi kode berikut untuk mendapatkan persamaan garis lurus bagi NIM GENAP

def point(x,y):
 return x,y

def line_equation_of(p1, p2):
   M = #TODO 1: tulis rumus untuk mendapatkan nilai M disini
   C = #TODO 2: tulis rumus untuk mendapatkan nilai C disini
   return f"y = {M:.2f}x + {C:.2f}"

print("Persamaan garis yang melalui titik A dan B:")
print(line_equation_of(point(4,-2),point(-1,3)))
#ubah nilai input dengan 4 digit nim akhir kalian
#dan lakukan perhitungan manual sesuai rumus yang kalian gunakan dalam fungsi
```

```
#untuk membuktikan bahwa output sudah benar
```

KEGIATAN PERCOBAAN 6 (mencari persamaan garis melalui 1 titik)

- Tentukan persamaan garis lurus yang melalui titik (6,-2) dan bergradien 2.

Untuk mencari suatu persamaan garis lurus, kita tetap bisa menggunakan rumus sebelumnya diatas. Namun disini nilai `m` telah diketahui, maka kalian hanya perlu menghitung nilai `c` saja. Kalian bisa menggunakan rumus berikut untuk mencari nilai C:

$$y - y_1 = m(x - x_1)$$

```
# lengkapi kode berikut untuk mendapatkan persamaan garis lurus bagi NIM GANJIL

def point(x,y):
 return x,y

def line_equation_of(p1, M):
   C = #TODO 1: tulis rumus untuk mendapatkan nilai C disini
   return f"y = {M:.2f}x + {C:.2f}"

print("Persamaan garis yang melalui titik (6,-2) dan bergradien -2:")
print(line_equation_of(6,-2,-2)) #ubah nilai input dengan 3 digit nim akhir kalian
#dan lakukan perhitungan manual sesuai rumus yang kalian gunakan dalam fungsi
#untuk membuktikan bahwa output sudah benar
```

```
# lengkapi kode berikut untuk mendapatkan persamaan garis lurus bagi NIM GENAP

def point(x,y):
 return x,y

def line_equation_of(p1, M):
   #TODO 1: gunakan inner function dan closure untuk menghitung nilai C

   C = #TODO 2: panggil fungsi inner untuk mendapatkan nilai C
   return f"y = {M:.2f}x + {C:.2f}"

print("Persamaan garis yang melalui titik (6,-2) dan bergradien 2:")
print(line_equation_of(point(6,-2),-2))
#ubah nilai input dengan 3 digit nim akhir kalian
#dan lakukan perhitungan manual sesuai rumus yang kalian gunakan dalam fungsi
#untuk membuktikan bahwa output sudah benar
```

```
Persamaan garis yang melalui titik (6,-2) dan bergradien -2:
y = -2.00x + 10.00
```

TUGAS PRAKTIKUM

Modifikasi kode pada percobaan 4, 5 dan 6 untuk menjawab pertanyaan dibawah ini.

- **(nim ganjil)** Diketahui sebuah sebuah titik A (3,4) dengan gradien 2
- **(nim genap)** Diketahui sebuah sebuah titik A (3,4) dan titik B (5,6)

baik soal **nim ganjil** maupun **nim genap** mengalami transformasi beruntun sbb:

- pertama, translasi dengan tx = 2 dan ty = -3,
- kemudian rotasi sejajar sumbu x positif sebesar 60 derajat,
- dan perbesaran skala dengan faktor skala 1.5 pada sumbu x dan faktor skala 2 pada sumbu y.

Tentukan persamaan garis hasil transformasi!

Sehingga output yang dihasilkan seperti gambar dibawah ini (buat lebih kreatif diperbolehkan) perhatikan juga ketentuan rubrik nilai pada dokumen modul sebagai acuan pengerjaan!

NIM yang digunakan adalah 3 digit terakhir [xyz]. Sehingga titik A (x,y); titik B (y,z); gradien = x; tx = y; ty = z; sx = z; sy = x.

```
Persamaan garis yang melalui titik (3,4) dan bergradien 2:
y = 2.00x + -2.00
Persamaan garis baru setelah transformasi:
y = 2.00x + 4.76


Persamaan garis yang melalui titik (3,4) dan (5,6):
y = 1.00x + 1.00
Persamaan garis baru setelah transformasi:
y = -4.98x + 21.86
```

KRITERIA & DETAIL PENILAIAN TUGAS PRAKTIKUM

Kriteria	Detail Penilaian	Poin	Program identik**
Percobaan 1	5 poin per metode run (HoF & currying)	10	10
Percobaan 2 & 3	5 poin per percobaan	10	10
Percobaan 4	5 poin per tranformasi	15	5
Percobaan 5 & 6	Sesuai dengan contoh (5 poin per percobaan)	10	5*
	Berdasarkan nim & disertai perhitungan (10 poin per percobaan)	20	10*
Tugas Praktikum	Menggunakan Decorator untuk transformasi gabungan	15	10
	Menggunakan inputan user untuk semua data yang diperlukan	10	5
	Data input Berdasar nim & disertai perhitungan	10	5
	Total Nilai (maksimal)	**100**	**60**

*) Nilai jika percobaan 5 & 6 dinilaikan saat demo.

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