VERSION 1.0

JULI , 2022

[PRAKTIKUM PEMROG. FUNGSIONAL]

MODUL 4 – Pengenalan Inner Functions, Closure dan Decorator Dalam Studi Kasus Matematika dan Fisika

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 juga mempelajari dari sumber external mengenai bahasa python.

TUJUAN PRAKTIKUM

1. Mahasiswa mampu mengimplementasikan pemrograman fungsional dalam penerapan matematika dan fisika.

PERSIAPAN SOFTWARE/APLIKASI

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

MATERI POKOK

Pada pemrograman fungsional, modul juga dapat diakses melalui google collab agar lebih interaktif :
Modul 4 :
https://colab.research.google.com/drive/18IdClXiooZjf5wFHFbLc0zJK1qvI0pea#scrollTo=99ES0vwT-KTW

Modul 4 Pemrograman Fungsional

Inner Functions, Closure, and Decorator in Case Study of Math and Physic Functions

PERSIAPAN MATERI

Inner Functions

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 dapatberinteraksi dengan variabel di luar 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.

Closures

Untuk lebih memahami closure, kita perlu belajar tentang ruang lingkup variabel di Python. Ruang lingkup variabel mengacu pada area di mana Anda dapat melihat atau mengakses variabel. Ruang lingkup variabel ditentukan oleh 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 dapatdiakses oleh semua fungsi dalam file.
- Local Scope : Ketika variabel didefinisikan di dalam fungsi, itu adalah lokal untuk fungsiitu.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.

Perhatikan contoh berikut:

```
#Contoh variabel global
x = "Variabel global"
def iniFungsi():
    print(x, "ini dapat diakses dari dalam 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 lain penerapan variabel global
x = 5
def iniFungsiJuga():
  x = x * 2
  print(x)
iniFungsiJuga()

# kode diatas akan menghasilkan eror:
# UnboundLocalError: local variable 'x' referenced before assignment
```

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
<ipython-input-6-4e3d917433a9> in <module>()
      4   x = x * 2
      5   print(x)
----> 6 iniFungsiJuga()

<ipython-input-6-4e3d917433a9> 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 membatasisemua upaya untuk memperbarui nilainya. Untuk menghindari situasi seperti ini, kita perlu menggunakan kata kunci "global". Perhatikan contoh berikut:

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

```
10
```

```
#Contoh 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. Sekarangkita akan beranjak ke lingkup variabel non-lokal.

Di Python, variabel non-lokal defaultnya adalah bersifat read only. Untuk dapat memodifikasinya, kita harus mendeklarasikannya sebagai variabel non-lokal (menggunakan keyword nonlocal). Seperti pada contoh kode berikut:

```
#Contoh variabel nonlokal
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()
```

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

```
value variabel tinggi awal 10
volume kubus adalah : 40
value variabel tinggi akhir 4
value variabel tinggi global 20
```

Okeii lanjut ke contoh berikutnya

```
 1 x = 1
 2 y = 5
 3 def func():
 4   global y
 5   x = 2
 6   y += 1
 7   z = 10
 8   print("Lokal Variabel x =", x)
 9   print("Global Variabel y =", y)
10   print("Lokal Variabel z =", z)
11
12 func()
13 print("Global variable x =", x)
14 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 di dalam 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 kitadapat 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.

Di Python, semuanya adalah objek, dan variabel adalah referensi ke objek nya. Ketika Anda 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 yangsama.

Sudah cukup paham dengan lingkup variabel kan...

Sekarang bagian mana yang menjelaskan tentang closure?
Closure digunakan untuk menghindari penggunaan variabel global (yang sangat tidak disarankan dalam paradigma fungsional) dan menyediakan fungsi penyembunyian data (datahiding) 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 closuredaripada 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-lokalmsg 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 0x7f35f13ce4d0>
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                                 Traceback (most recent call last)
<ipython-input-22-2c954777807e> in <module>()
      1 del print_msg
----> 2 print_msg("Hello")

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

```
1 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.

Decorators

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 program 1 (Single decorator to Single Function):

```
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.

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 tunjukkan itu dalam praktik di bawah 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-33-b2688f7ddfde> in <module>()
      3   print(nama)
      4
----> 5 myname("Python 3")

TypeError: 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 yangakan 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 dikembalikannya adalah 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.

KEGIATAN PERCOBAAN

Percobaan 1 (penerapan *inner function*):

```
def factorial(number):
  # Validasi inputan
  if not isinstance(number, int):
    error = "Maap gabisa, harus angka ya syg."
    return error
  if number < 0:
    error = "Maap gabisa, harus angka positipp kak."
    return error

  # Hitung angka faktorial
  def inner_factorial(number):
    if number <= 1:
      return 1
    return number * inner_factorial(number - 1)
  return inner_factorial(number) # mengembalikan nilai dari pemanggilan fungsi

```

```
print(factorial(4))
f7 = factorial(7)
print(type(f7),f7)
print(factorial(-7))
print(factorial("ahayy"))
```

```
24
<class 'int'> 5040
Maap gabisa, harus angka positipp kak.
Maap gabisa, harus angka ya syg.
```

Percobaan 2 (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

op10x=perkalian(10)  # membuat fungsi perkalian 10
op5x=perkalian(5)    # membuat fungsi perkalian 5

print(op10x(3)) #hitung 10x3 = 30
print(op5x(7))  #hitung 5x7 = 35
print(op5x(op10x(2))) # 5x10x2 = 100
```

```
30
35
100
```

```
# pemanggilan dengan teknik currying
a = perkalian(10)(3) #hitung a = 10x3
print(a)  # cetak a --> 30
print(perkalian(5)(7)) #hitung 5x7 = 35
print(perkalian(5)(perkalian(10)(2))) #5x10x2 = 100
```

```
30
35
100
```

Percobaan 3 (penerapan *single decorators to single function*):

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

    return wrapper

```

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

say_hi()
```

```
'HELLO THERE'
```

Percobaan 4 (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

@split_string
@title_decorator
def say_hi():
    return 'hello there'
say_hi()
```

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

Karena dekorator hanya dilaksanakan ketika fungsi berada di bawah simbol @, Jadi:

1. Fungsi title_decorator akan mendekor fungsi utama terlebih dahulu.
2. Fungsi split_data akan mendekor fungsi utama setelah fungsi title_decorator.
3. Fungsi utama adalah say_hi
4. Sedangkan title_decorator dan split_data adalah dekorator.

Percobaan 5 (latihan tugas modul part 1):

Hitung gaya gravitasi Planet Scarif dengan mengikuti kriteria berikut:

Rumus hukum gravitasi: $F = G\frac{m_1 m_2}{r^2}$

Ketentuan:

1. Jadikan konstanta G = 9.8 sebagai variabel global.
2. Buatlah fungsi perkalian.
3. Buatlah fungsi pembagian dengan menggunakan closure-inner function untuk diakses dengan teknik currying.
4. Buatlah fungsi perpangkatan kuadrat.
5. Gabungkan ketiga fungsi tersebut menggunakan closure sehingga membentuk rumus Gaya Gravitasi, jadikan rumus sebagai fungsi.

Soal: Hitung gaya gravitasi F, jika m1 = 5, m2 = 4, dan r = 2

```
G = 9.8 #Variabel global
m1 = 5
m2 = 4
r = 2

#Fungsi perkalian
def kalikan(a,b):
  return a*b

#Fungsi pembagian
def pembagian(d):
  def operator(e):
    return d / e
  return operator

#Fungsi perpangkatan kuadrat
def kuadrat(f):
  return f * f

#Fungsi dengan closure
def rumus(g, m1, m2, r):
  def f():
    m1m2 = kalikan(m1, m2)        # Perkalian m1 dan m2
    hasil_kali = kalikan(m1m2, g) # Perkalian m1,m2 dengan G
    r_kuadrat = kuadrat(r)     # Perpangkatan r kuadrat
    #Pembagian dari hasil kali m1,m2,G dengan r kuadrat
    hasil = pembagian(hasil_kali)(r_kuadrat) #diakses dengan teknik currying.
    return hasil
  return f

F = rumus(G, m1, m2, r)
print("Gaya gravitasi = " + str(F()))
```

```
Gaya gravitasi = 49.0
```

Percobaan 6 (latihan tugas modul part 2):

Dikarenakan tembakan death star, Planet Scarif mengalami anomali pada gaya gravitasinya

yang mengakibatkan konstanta G bernilai 1.49. Perubahan juga dialami pada nilai r menjadi 0.75. Anomali ini juga mengakibatkan perubahan pada rumus gaya Gravitasi Planet Scarif yangbaru sebagai berikut:

$$F_{baru} = \frac{F_{lama} * r_{baru}^{2}}{G_{baru}}$$

Hitung nilai Fbaru dengan menggunakan nilai F sebelumnya menjadi nilai Flama. Dengan ketentuan sebagai berikut:

1. Deklarasikan nilai G di suatu fungsi dekorator sehingga tidak mengubah variabel global Gyang asli.
2. Buat fungsi decorators untuk r.
3. Hitung nilai Fbaru dengan menggunakan decorators.

```
#Fungsi G
def Gbaru(function):
  G = 1.49
  def wrapper():
    func = function()
    hasil = pembagian(func)(G)
    return hasil
  return wrapper

#Fungsi r
def rbaru(function):
  r = 0.75
  def wrapper():
    func = function()
    hasil = kalikan(func, kuadrat(r))
    return hasil
  return wrapper


@Gbaru
@rbaru
def fbaru():
  F = rumus(G, 5, 4, 2)
  fbaru = F()
  return fbaru

fbaru()
```

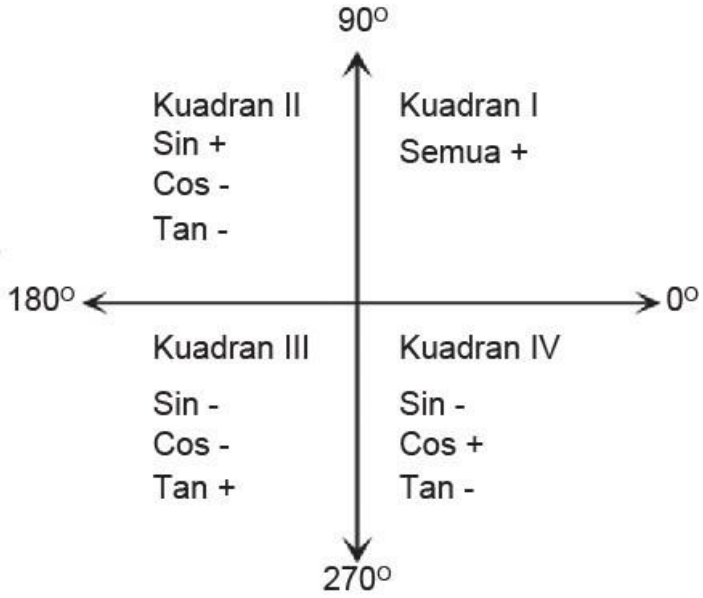
```
18.498322147651006
```

PRAKTIKUM MODUL

Kegiatan 1

Diketahui rumus menghitung sebuah poligon dengan jumlah sisi n adalah :

KETENTUAN KUADRAN



Hitunglah luas poligon dengan inputan berupa n dan r dengan ketentuan sebagai berikut :

1. Gunakan closure yang sesuai untuk fungsi alpha
2. Fungsi induk alpha berisi ketentuan kuadran dan fungsi inner berisi perhitungan sinus n menjadi nilai alpha (contoh : sinus 90 derajat = 1) Khusus untuk perhitungan sinus diperbolehkan menggunakan library
3. Apabila alpha bernilai negatif (berada di kadran III atau IV) maka beri peringatan "Alpha berada di kuadran '...', Nilai khayal"
4. Buatlah error handling untuk menghindari inputan nilai negatif dan string

5. Gunakan deklarasi fungsi pada percobaan 5 untuk menghitung luas
6. Perhatikan format output untuk luas

Silahkan modifikasi kode berikut tanpa merubah susunan serta nama variabel dan fungsi

```
def alpha(n):
  #Disini isikan ketentuan kuadran
  def inner_alpha():
    return #Masukkan kalkulasi perhitungan sinus n menjadi nilai alpha
  return inner_alpha


def luas_poligon(n, r):
  #Masukkan perhitungan luas poligon sesuai rumus yang sudah diberikan.
  return

#Test case 1:
#Input:
#n = 9
#r = 21
#Output: luas = 1275.612

#Test case 2:
#Input:
#n = 1.3
#r = 7
#Output: "Alpha berada di kuadran IV, Nilai khayal"

#Test case 3:
#Input:
#n = 360
#r = 21
#Output: luas = 1385.372

#Test case 4:
#Input:
#n = -5
#r = 21
#Output: "Inputan salah"
```

Kegiatan 2

Diketahui:

matrix X =

$$\begin{matrix} 4 & 7 \\ 8 & 5 \end{matrix}$$

matrix Y =

$$\begin{matrix} 3 & 2 \\ 7 & 5 \end{matrix}$$

matrix A =

$$\begin{matrix} 9 & 7 \\ 6 & 2 \end{matrix}$$

transpose matrix: $\mathbf{A} = \begin{bmatrix} 2 & 3 \\ 0 & 1 \end{bmatrix}$ **Hasilnya** ➔ $\mathbf{A^T} = \begin{bmatrix} 2 & 0 \\ 3 & 1 \end{bmatrix}$

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, B = \begin{bmatrix} p & q \\ r & s \end{bmatrix}$$

perkalian matrix : $A.B = \begin{bmatrix} ap + br & aq + bs \\ cp + dr & cq + ds \end{bmatrix}$

Ditanyakan:

Cari hasil perhitungan dari matrix berikut :

resultMatrix^T = Y^T + Z

dimana Z = A^T * X

*Silahkan modifikasi kode berikut tanpa merubah susunan serta nama variabel dan fungsi

```
#Fungsi ini akan dijadikan dekorasi untuk mengembalikan hasil transpose matrix
#dari matrix yang terdapat pada parameter fungsi.
def transpose(function):
  def wrapper(*args, **kwargs):
    matrix =
    #Lakukan operasi transpose matrix disini
    return matrix
  return wrapper

#Isikan decoration untuk men-transpose-kan matrix m
def Tmatrix(m):
  Tm = #Pastikan fungsi ini pure (tidak merubah nilai matrix asli)
  return Tm

#Fungsi ini digunakan untuk menjumlahkan matrix Yt + Z.
def penjumlahanMatrix(function1, function2):
  #Matrix Yt dan Z dipanggil sesuai parameter fungsi yaitu function1 dan function2.
  result =  # Gunakan Variabel ini untuk menyimpan hasil penjumlahan matrix.
```

```
            #Lakukan operasi penjumlahan matrix disini
  return result

#Fungsi ini digunakan untuk mengalikan matrix At dan X.
def perkalianMatrix(function):
  #Dilarang mendeklarasikan matrix X pada function ini.
  #Anda bisa memperlakukan fungsi ini sebagai currying
  def kali(matrix):
    AtX =  # Gunakan Variabel ini untuk menyimpan hasil perkalian matrix.
           # Lakukan operasi perkalian matrix disini
    return AtX
  return kali

#Isikan decoration untuk men-transpose result matrix
def resultMatrix(X, Y, A):
    result = #Tuliskan rumus perhitungan matrix disini
    return result

#Note* Untuk lebih mengerti Kegiatan 2,
#silahkan dipahami Contoh program 2, Percobaan 5 dan 6 pada modul ini!

#Test case 1:

#Input:
#matrixX = [4,7,8,5]
#matrixY = [3,2,7,5]
#matrixA = [9,7,6,2]
#Diasumsikan matrixA = |9  7|
                      #|6   2| demikian pula dengan matrix lainnya
#Output: [87, 46, 100, 64]
```

INDIKATOR PENILAIAN

Kegiatan 1:

- Lulus Test Case 1: 5
- Lulus Test Case 2: 15
- Lulus Test Case 3: 10
- Lulus Test Case 4: 15

Kegiatan 2:

- Lulus Test Case 1: 55

Note:

- Harap diperhatikan bahwa praktikan wajib menjelaskan alur program yang dibuat untuk menghindari kecurigaan duplikat kode dengan praktikan lain. Apabila terjadi duplikat kode dan praktikan tidak bisa menjelaskan program yang dibuat, nilai otomatis 30-40.