Modul 6 Praktikum Pemrograman Berbasis Fungsi

March 26, 2023

1 Modul 6 Praktikum Pemrograman Berbasis Fungsi

Tujuan Praktikum:

1. Mahasiswa dapat memahami penerapan Higher Order Function pada functional programming
2. Mahasiswa dapat memahami peran higher order function pada paradigma functional programming

Higher Order Function merupakan fungsi yang menggunakan fungsi lain sebagai parameter atau sebagai hasil return

Terdapat tiga jenis Higher order Function, yaitu: 1. Fungsi yang menerima fungsi sebagai salah satu argumennya

2. Fungsi yang mengembalikan fungsi
3. Fungsi yang menerima fungsi dan mengembalikan fungsi

Berikut penerapan Higher Order Function pada paradigma functional programming:

1.1 Lambda Function

Fungsi lambda juga dikenal sebagai Fungsi Anonim (Anonymous FUnctions). Kita dapat menggunakan lambda sebagai Fungsi yang hanya sebaris (inline).

lambda adalah kata kunci python yang digunakan untuk mendefinisikan Fungsi lambda untuk operasi baris tunggal dalam pemrograman. Kita tidak dapat menggunakan Fungsi lambda untuk multiline, Jadi saat membuat Fungsi lambda hanya dapat dilakukan sebaris.

1.1.1 How to Define lambda Function:

```
syntax:
lambda_Function = lambda return_value: parameter + 10
lambda_Function(User Define)
e.g:-
  l_fun = lambda x:x+10
  print(l_fun(10))
Note:
  l_fun = Generate by lambda Function which need some parameter to
          run the above code.
  x = as you can see that i have passed '10' as parameter to my lambda
      Function (l_fun(10)). '10' is the value of x.
```

1.1.2 How to convert Normal Function into lambda Function:

```
1. Normal Function
  def add(x):
    return x+10

2. lambda Function
  lambda x:x+10
Note:
  To convert the Normal Function into lambda follow this steps.
    1. Replace 'def function_name' with 'lambda'.
    2. Declare variable after lambda which will hold you parameter,
       in our case we have 'x'.
    3. Now You are good to go with programing logic after ':'.
    e.g:-
        1. Replaceing 'def add'
           def add -> lambda
        2. Now We need a parameter i.e 'x' in our case
           def add(x): -> lambda x:

        3. Logic Part for Addition
           x+10
        Once you have done with above part you will get your
        lambda Function, which is something like this.

          Function: lambda x:x+10
```

Berikut Contoh penggunaan Lambda Function

[ ]:
```
#Addiotion Using lambda Function
add = lambda a:a+10
print('\nAddition is:',add(20))
```

```
Addition is: 30
```

[ ]:
```
#Lambda Function To Concatinate Two String
First_Last_name = lambda a,b:str(a)+' '+str(b)
print('\nFull name:',First_Last_name('Sains','Data'))
```

```
Full name: Sains Data
```

[ ]:
```
#Printing Table using lambda Function
table = lambda n:[n*i for i in range(1,11)]
print('\nTable:',table(5))
```

```
Table: [5, 10, 15, 20, 25, 30, 35, 40, 45, 50]
```

[ ]:
```
#Sorting Name with last name using lambda Function
names=['Farah Faizah','Wulan Sabina','Kharisma Gumilang','Amar Sidiq','Salwa␣
 ↪Amelia','Zaki Abdillah']
names.sort(key=lambda name: name.split(' ')[-1].lower())
print('\nName Sorted by Last Name:\n',names)
```

```
Name Sorted by Last Name:
 ['Zaki Abdillah', 'Salwa Amelia', 'Farah Faizah', 'Kharisma Gumilang', 'Wulan
Sabina', 'Amar Sidiq']
```

[ ]:
```
#lambda Function within Function for (a+b)^2
def Multplication(b):
  return lambda x: x**2+b**2+2*x*b

lam_Fun=Multplication(4)
print('\n(a+b)^2 is:',lam_Fun(2))
```

```
(a+b)^2 is: 36
```

1.2 Aplikasi Higher Order Function

[ ]:
```
# 1 Membuat Login User dan Password
print('\nHigher Order Function\n')
def Login(func,username,password):
  isValid = func(username, password)
  if isValid:
    return f'Welcome {username}'
  else:
    return 'Invalid username or password... ?'

def validate_user_data(temp_uname, temp_pass):
  if temp_uname.strip()=='sainsdata' and temp_pass.strip()=='sainsdata08':
    return True

def get_user_details():
  uname = str(input('Enter your username:'))
  passwrd = str(input('Enter your password:'))

  return uname,passwrd
```

```
Higher Order Function
```

Masukkan username = sainsdata dan password sainsdata08

```
[ ]: usrname, paswd = get_user_details()
     print(f'1.{Login(validate_user_data, usrname, paswd)}')
```

```
1.Welcome sainsdata
```

Dalam contoh di atas kita telah membuat 3 fungsi tetapi jika Anda Perhatikan Dalam Fungsi "Login" kita telah meneruskan "validate_user_data" sebagai argumen

Kita dapat menggunakan Higher Order Function untuk mengonversi fungsi yang menggunakan beberapa argumen menjadi rangkaian fungsi yang masing-masing menggunakan satu argumen. Lebih khusus lagi, mengingat fungsi f(x, y), kita dapat mendefinisikan fungsi g sehingga g(x)(y) setara dengan f(x, y). Di sini, g adalah fungsi tingkat tinggi yang menerima satu argumen x dan mengembalikan fungsi lain yang menerima satu argumen y. Transformasi ini disebut currying.

```
[ ]: print('\n\nCurrying\n')
     # 1.
     def get_1st_number(num_1):
       def get_2nd_number(num_2):
         return num_1+num_2

       return get_2nd_number

     print(f'1. Addition of two Number: {get_1st_number(10)(20)}')
```

```
Currying

1. Addition of two Number: 30
```

Fungsi di atas adalah *pure function* karena sepenuhnya bergantung pada input.

```
[ ]: # 2.
     def pas_function(user_func):
       def get_x(x):
         def get_y(y):
           return user_func(x,y)
         return get_y
       return get_x

     def mul_function(a,b):
       return a+b

     pas_func = pas_function(mul_function)
     print(f'2. Currying With user define or pre define function:{pas_func(2)(4)}\n')
```

```
2. Currying With user define or pre define function:6
```

Dalam contoh di atas Anda dapat menerapkan fungsi apa pun yang berlaku untuk dua argumen seperti "max", "min", "pow". dll. Anda juga dapat melewati fungsi yang ditentukan pengguna..

dalam kasus kita, telah melewati “mul_function”, kita dapat menggunakan Fungsi tersendiri tetapi pastikan fungsi tersebut akan berfungsi pada 2 parameter.

Berikut contoh Higher Order Function dengan Currying

[ ]:
```
print('\nHigher order fucntion with Currying\n')
#1.

def Login(func,welcom_func):
    def get_username(uname):
        def get_password(pas):
            isValid = func(get_user_details,uname,pas)
            if isValid:
                return welcom_func(uname)
            else:
                return Invalid_user()
        return get_password
    return get_username

def check_valid_User(func_user_input,usernm,userpas):
    tempUname,tempPass = func_user_input()
    return ((tempUname.strip()==usernm) and (tempPass.strip()==userpas.strip()))

def welcome_user(uname):
    return f'Welcome {uname}'

def Invalid_user():
    return 'invalid username or password'

def get_user_details():
    tempName = str(input('Username:'))
    tempPass = str(input('Password:'))
    return str(tempName).strip(), str(tempPass).strip()

login = Login(check_valid_User,welcome_user)
print(login('sainsdata')('sainsdata08'))
```

```
Higher order fucntion with Currying

Welcome sainsdata
```

1.3 Penerapan Higher Order Function pada Decorator

Contoh menggabungkan kata dengan higher order function kemudian digunakan decorator untuk menyatukan kalimat tersebut.

[ ]:
```
def reversal(sentence_func):
    def reversed_sentence(*args, **kwargs):
```

```
        original_sentence = sentence_func(*args, **kwargs)
        return f"Reversed: {''.join(reversed(original_sentence))}"
    return reversed_sentence
```

[ ]:
```
@reversal
def pbf():
    return "Pemrograman Berorientasi Fungsi Sains Data"
def bigdata():
    return "Functional Programming menggunakan map and reduce untuk aplikasi␣
 ↪big data"


print(pbf())
print(bigdata())
```

```
Reversed: ataD sniaS isgnuF isatneiroreB namargormeP
Functional Programming menggunakan map and reduce untuk aplikasi big data
```