MODUL 4

Higher Order Function dan Curried Function

Praktikum Pemrograman Berbasis Fungsi

SAINS DATA
INSTITUT TEKNOLOGI SUMATERA

4.1 Tujuan

1. Mahasiswa dapat memahami penerapan *Higher Order Function* pada Pemrograman Fungsional dengan menerapkan fungsi Map, Reduce, dan Filter pada pemrosesan data.
2. Mahasiswa dapat memahami penerapan *Curried Function* pada Pemrograman Fungsional

4.2 Konsep Dasar

4.2.1 Higher Order Functions

Sebuah fungsi yang dapat menerima fungsi lain sebagai parameter dikenal sebagai fungsi orde tinggi (*Higher Order Function*) dan merupakan salah satu konsep utama dalam Pemrograman Fungsional.

Fungsi orde tinggi adalah fungsi yang melakukan setidaknya salah satu dari hal berikut (atau bahkan keduanya):

1) Menerima satu atau lebih fungsi sebagai parameter,
2) Mengembalikan fungsi sebagai hasil.

Semua fungsi lain dalam Python yang tidak memenuhi kriteria ini disebut fungsi orde pertama (*First Order Function*).

Contoh *Higher Order Function*

```
def apply(x, function):
    result = function(x)
    return result
def mult(y):
    return y * 10.0

apply(5, mult)
```

Fungsi apply adalah fungsi orde tinggi karena perilaku dan hasilnya bergantung pada fungsi lain yang diteruskan sebagai parameter. Fungsi baru kita definisikan sebagai mult misalnya untuk mengalikan sebuah angka dengan 10.0 . Kemudian panggil fungsi apply dan mult Karena apply adalah fungsi orde tinggi, kita bisa mengganti mult dengan fungsi lain tanpa mengubah struktur apply.

Python memiliki beberapa fungsi bawaan yang merupakan *higher-order function* diantaranya:

1) Fungsi map() : Transforming Data
   Fungsi map menerapkan fungsi yang diberikan ke semua item dalam suatu iterable dan mengembalikan suatu iterable baru beserta hasilnya.

   Contoh 1 : menggunakan def

```
# Define a function to square a number
def square(x):
    return x ** 2

# Apply the square function to a list of numbers
numbers = [1, 2, 3, 4, 5]
squared = list(map(square, numbers))
# Result: [1, 4, 9, 16, 25]
```

Contoh 2 : menggunakan fungsi Lambda

```
# Using a lambda function to square numbers
numbers = [1, 2, 3, 4, 5]
squared = list(map(lambda x: x ** 2, numbers))
# Result: [1, 4, 9, 16, 25]
```

2) Fungsi filter() : Selecting Elements

Fungsi filter menyaring elemen dari suatu iterable berdasarkan fungsi yang diberikan (yang mengembalikan True atau False) dan mengembalikan iterable baru dengan elemen yang memenuhi kondisi.

Contoh 1 : menggunakan def

```
# Define a function to filter even numbers
def is_even(x):
    return x % 2 == 0

# Filter even numbers from a list
numbers = [1, 2, 3, 4, 5]
evens = list(filter(is_even, numbers))
# Result: [2, 4]
```

Contoh 2 : menggunakan fungsi Lambda

```
# Using a lambda function to filter even numbers
numbers = [1, 2, 3, 4, 5]
evens = list(filter(lambda x: x % 2 == 0, numbers))
# Result: [2, 4]
```

3) Fungsi reduce() : Aggregating Data

Fungsi reduce tersedia dalam modul functools menerapkan fungsi biner secara kumulatif ke item yang dapat diulang untuk menguranginya menjadi satu nilai tunggal.

Contoh 1 : menggunakan def

```python
from functools import reduce

# Define a function to find the product of two numbers
def multiply(x, y):
    return x * y

# Find the product of all numbers in a list
numbers = [1, 2, 3, 4, 5]
product = reduce(multiply, numbers)
# Result: 120 (1 * 2 * 3 * 4 * 5)
```

Contoh 2 : menggunakan fungsi Lambda

```python
from functools import reduce

# Using a lambda function to find the sum of numbers
numbers = [1, 2, 3, 4, 5]
sum = reduce(lambda x, y: x + y, numbers)
# Result: 15 (1 + 2 + 3 + 4 + 5)
```

4.2.2 Curried Functions

Curried Function (Currying) adalah teknik yang memungkinkan pembuatan fungsi baru dari fungsi yang sudah ada dengan mengikat satu atau lebih parameter ke nilai tertentu. Teknik ini merupakan salah satu cara utama untuk menggunakan kembali fungsi dalam Python. Suatu fungsi dapat ditulis sekali di satu tempat dan kemudian digunakan kembali dalam berbagai situasi.

Misalkan kita memiliki sebuah fungsi yang menerima dua parameter

```
operation(x, y): return x * y
```

Fungsi operation() kemudian dapat digunakan sebagai berikut:

```
total = operation(2, 5)
```

Jika kita perlu menggandakan sebuah angka, kita dapat menggunakan kembali fungsi operation() berkali-kali, misalnya:

```
operation(2, 5)
operation(2, 10)
operation(2, 6)
operation(2, 151)
```

angka 2 tidak berubah di setiap pemanggilan fungsi operation(). Bagaimana jika kita menetapkan parameter pertama selalu 2? Ini berarti kita bisa membuat fungsi baru yang tampaknya hanya menerima satu parameter (yaitu angka yang akan digandakan).

```
double = operation(2, *)
```

```
double(5)
double(151)
```

Kapan *Curried Functin* digunaan?
Ketika Anda memiliki fungsi yang akan digunakan kembali dengan beberapa argumen yang tetap. *Currying* memungkinkan Anda untuk membuat versi yang lebih spesifik dari suatu fungsi umum, sehingga meningkatkan penggunaan kembali kode.

Regular Function vs Curried Function

Regular Function	Curried Function

`def add(a, b):`
`  return a + b`
 	
`result = add(2, 3)`
`print(result)  # Output: 5`
	`def curried_add(a):`
	`  def add_b(b):`
	`  return a + b`
	`  return add_b`
	
	`add_2 = curried_add(2)`
	`result = add_2(3)`
	`print(result)  # Output: 5`

4.3 Prosedur Percobaan

4.3.1 Penggunaan *Higher Order Functions* : Map, Reduce, dan Filter

1) Gunakan fungsi map() untuk membuat list yang berisi pangkat dari isi list tersebut.[1,2,3,4,5,6,7,8,9,10].

```
lst = [1,2,3,4,5,6,7,8,9,10]
print(list(map(lambda x: x**2, lst)))
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

2) Gunakan fungsi map untuk menggandakan setiap angka dan menguranginya dengan satu dalam list dengan menggunakan fungsi lambda

```
numbers = [4, 11, 15, 23, 20, 10, 100]
more_nums = [2, 3, 4, 5, 7, 9]

print(list(map(lambda x, y: (x, y), numbers, more_nums))) #data awal
print(list(map(lambda x, y: (x *2 -1, y*2-1 ), numbers, more_nums)))
```

```
[(4, 2), (11, 3), (15, 4), (23, 5), (20, 7), (10, 9)]
[(7, 3), (21, 5), (29, 7), (45, 9), (39, 13), (19, 17)]
```

3) Gunakan fungsi filter() dapat membuat list yang berisikan nilai genap antara angka 1 sampai 20.

```
evenFilteredBelowTwenty = list(filter(lambda x: x % 2 == 0, range(1,21)))
print(evenFilteredBelowTwenty)
```

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

4) Tulislah program yang mana fungsi map() and filter() dapat membuat list yang berisikan pangkat dari list tersebut yang bernilai genap [1,2,3,4,5,6,7,8,9,10].

```
evenFiltered = list(filter(lambda x: x % 2 == 0, lst))
print(list(map(lambda x: x**2, evenFiltered)))
```

```
[4, 16, 36, 64, 100]
```

5) Carilah nilai intersection dari dua list, gunakan fungsi filter.

```
lst2 = [5,6,7,8,9,10,11,12,13,14,15]
print(list(filter(lambda x: x in lst2, lst)))
print([x for x in lst if x in lst2])
```

```
[5, 6, 7, 8, 9, 10]
[5, 6, 7, 8, 9, 10]
```

6) Gunakan fungsi filter untuk menampilkan nama dengan awalan huruf tertentu.

```
names = ["April", "aTasha", "Fred", "Tony", "Sandra", "Jose", "Soloman", "Ashliegh"]
new_names_lambda = list(filter(lambda name: True if name[0].lower() == "a" else False, names))
print(new_names_lambda)
```

```
['April', 'aTasha', 'Ashliegh']
```

7) Hitunglah jumlah angka dari 1 sampai 100, menggunakan reduce.

```
print(reduce(lambda previous, current: previous + current, range(1,101)))
```

```
5050
```

8) Gunakan fungsi reduce untuk mengalikan angka dalam daftar di bawah ini bersama dengan fungsi lambda.

```
my_list = [1,3,5,7,9]

mult_lam = reduce(lambda x, y: x * y, my_list)
print(mult_lam)
```

```
945
```

4.3.2 Penggunaan Curried Functions

1) Membuat fungsi curried untuk menjumlahkan dua angka.

```
def add(x):
    return lambda y: x + y

add5 = add(5)  # Fungsi yang selalu menambahkan 5
print(add5(3))  # Output: 8
print(add5(10)) # Output: 15
```

2) Membuat fungsi curried untuk membuat format string dengan template tertentu.

```
def greet(salutation):
    return lambda name: f"{salutation}, {name}!"

say_hello = greet("Hello")
say_good_morning = greet("Good morning")

print(say_hello("Alice"))  # Output: Hello, Alice!
print(say_hello("Bob"))
print(say_good_morning("Bob"))  # Output: Good morning, Bob!
print(say_good_morning("Alice"))
```

3) Membuat fungsi curried untuk menghitung harga setelah diskon.

```
def discount(percentage):
    return lambda price: price - (price * percentage / 100)

discount_10 = discount(10)
discount_25 = discount(25)

print(discount_10(200))  # Output: 180.0
print(discount_25(300))  # Output: 225.0
```

4.3.3 Penggunaan *Higher Order Function* dengan *Currying*

Berikut contoh penerapkan *Higher Order Function* dan *Currying* untuk sistem login, di mana fungsi Login menerima **username** dan **password** secara bertahap, lalu memvalidasi dengan **check_valid_User**. Jika valid, pengguna disambut dengan welcome_user, jika tidak, ditampilkan pesan error.

```
print('\nHigher order fucntion with Currying\n')

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'))
```

Output jika berhasil login:

```
Higher order fucntion with Currying

Username:sainsdata
Password:sainsdata08
Welcome sainsdata
```

Output jika tidak berhasil login:

```
Higher order fucntion with Currying

Username:sainsdata
Password:1234
invalid username or password
```