

Perancangan dan Pemrograman Berbasis Objek

IF2105

By:

Apriyanto Halim, S.Kom., M.Kom.

COURSE OVERVIEW

COURSE OVERVIEW

Mata kuliah ini membekali mahasiswa dengan pengetahuan dan keterampilan mengenai konsep dari pemrograman berbasis objek dan lanjutannya pada konsep Solid dan Design Pattern. Dengan mempelajari mata kuliah ini, mahasiswa diharapkan dapat mempermudah pengembangan program dengan cara mengikuti model yang telah ada di kehidupan sehari-hari.

COURSE GOALS

Adapun tujuan yang ingin dicapai pada matakuliah ini, yaitu :

- Memimpin dan bekerja dalam tim, mandiri, dan bertanggung jawab terhadap pekerjaannya.
- Berpikir kreatif dan inovatif.
- Mampu bertanggung jawab atas pencapaian hasil kerja kelompok dan melakukan supervisi serta evaluasi terhadap penyelesaian pekerjaan yang ditugaskan kepada pekerja yang berada di bawah tanggung jawabnya.
- Menguasai konsep teoritis bidang pengetahuan Teknik Informatika secara umum dan konsep teoritis bagian khusus dalam bidang pengetahuan Teknik Informatika secara mendalam, serta mampu memformulasikan penyelesaian masalah prosedural.
- Mempunyai pengetahuan dalam penyusunan algoritma pemrograman yang efektif dan efisien serta dapat merancang, membangun, dan mengelola solusi perangkat lunak yang orisinal dan holistik terhadap permasalahan bisnis ataupun ilmiah.

COURSE OBJECTIVES

Pencapaian yang diharapkan dapat dicapai oleh mahasiswa, adalah :

- Mahasiswa mampu menerapkan konsep kelas dan objek.
- Mahasiswa mampu menerapkan konsep inheritance dan polymorphism.
- Mahasiswa dapat menangani masalah dengan tepat menggunakan Exception.
- Mahasiswa mampu memanfaatkan penggunaan Iterator Pattern.
- Mahasiswa mampu menerapkan Design Pattern.
- Mahasiswa mengetahui langkah-langkah yang digunakan dalam melakukan testing pada OOP.
- Mahasiswa mampu menerapkan prinsip SOLID pada kode yang dibuat.

REQUIREMENT

Sebelum kita masuk ke dalam modul ini, ada beberapa kebutuhan yang harus kalian lakukan, yaitu :

- Pastikan kalian sudah melakukan instalasi Python versi 3 keatas.
- Untuk UML yang digunakan menggunakan Class Diagram.
- Untuk software pembuatan UML berupa website Draw.IO.
- Untuk software pembuatan kode Python berupa Visual Studio Code.

DAFTAR ISI

Course Overview	i
Course Overview	i
Course Goals	i
Course Objectives.....	i
Requirement	i
Daftar Isi	ii
UNIT 1 PENGENALAN OOP.....	1
Unit Overview	1
Unit Objectives.....	1
Unit Contents	1
Pre Lab	2
Question.....	2
Content Lesson	2
Case Study / Project	2
Identification concept of problem / Project	2
Lesson 1: Kelas.....	2
Lesson 1: Overview.....	2
Lesson 2: Menambahkan Atribut	4
Lesson 2: Overview.....	4
Lesson 3: Method in class	6
Lesson 3: Overview.....	6
Solution	8
Instruction.....	8
Exercise	9
Exercise Objectives	9
Task 1:.....	9
Task 2:.....	9
Task 3:.....	9
Unit 2 INHERITANCE DAN POLYMORPHISM (1).....	11
Unit Overview	11
Unit Objectives.....	11
Unit Contents	11
Pre Lab	12
Question.....	12
Content Lesson	12

Case Study / Project	12
Identification concept of problem / Project	12
Lesson 1: Inheritance	13
Lesson 1: Overview.....	13
Lesson 2: Fungsi Super.....	14
Lesson 2: Overview.....	14
Solution	15
Instruction.....	16
Exercise.....	17
Exercise Objectives	17
Task 1:.....	17
Task 2:.....	17
Unit 2 INHERITANCE DAN POLYMORPHISM (2).....	18
Unit Overview	18
Unit Objectives	18
Unit Contents	18
Pre Lab	19
Question.....	19
Content Lesson	19
Case Study / Project	19
Identification concept of problem / Project	19
Lesson 3: Polymorphism	20
Lesson 3: Overview.....	20
Lesson 4: Membuat abstrak Class	22
Lesson 4: Overview.....	22
Solution	25
Instruction.....	25
Exercise	26
Exercise Objectives	26
Task 1:.....	26
Task 2:.....	27
UNIT 3 EXCEPTION	28
Unit Overview	28
Unit Objectives	28
Unit Contents	28
Pre Lab	29

Question.....	29
Content Lesson	29
Case Study / Project	29
Identification concept of problem / Project	29
Lesson 1: Error pada Python	29
Lesson 1: Overview.....	29
Lesson 2: Penanganan Error.....	32
Lesson 2: Overview.....	32
Lesson 3: Custom Pesan Kesalahan	33
Lesson 3: Overview.....	33
Solution	35
Instruction.....	35
Exercise	38
Exercise Objectives.....	38
Task 1:.....	38
Task 2:.....	38
UNIT 4 THE ITERATOR PATTERN (1)	39
Unit Overview	39
Unit Objectives.....	39
Unit Contents	39
Pre Lab	40
Question.....	40
Content lesson.....	40
Case Study / Project	40
Identification concept of problem / Project	40
Lesson 1: Perulangan dengan For	40
Lesson 1: Overview.....	40
Lesson 2: Perulangan dengan Iter	43
Lesson 2: Overview.....	43
Solution	45
Instruction.....	45
Exercise	46
Exercise Objectives.....	46
Task 1:.....	46
Task 2:.....	47
UNIT 4 THE ITERATOR PATTERN (2)	48

Unit Overview	48
Unit Objectives.....	48
Unit Contents	48
Pre Lab	49
Question.....	49
Content Lesson	49
Case Study / Project	49
Identification concept of problem / Project	49
Lesson 3: Tempat Penyimpanan Sementara Python	49
Lesson 3: Overview.....	49
Lesson 4: Comprehension pada Python.....	55
Lesson 4: Overview.....	55
Solution	57
Instruction.....	57
Exercise.....	59
Exercise Objectives.....	59
Task 1:.....	59
Task 2:.....	60
UNIT 4 THE ITERATOR PATTERN (3)	61
Unit Overview	61
Unit Objectives.....	61
Unit Contents	61
Pre Lab	62
Question.....	62
Content Lesson	62
Case Study / Project	62
Identification concept of problem / Project	62
Lesson 5: Pengecekan dengan Generator	62
Lesson 5: Overview.....	62
Lesson 6: Pengecekan dengan Coroutine.....	66
Lesson 6: Overview.....	66
Solution	67
Instruction.....	67
Exercise.....	69
Exercise Objectives	69
Task 1:.....	70

Task 2:.....	70
UNIT 5 DESIGN PATTERN 1 (1).....	72
Unit Overview	72
Unit Objectives.....	72
Unit Contents	72
Pre Lab	73
Question.....	73
Content Lesson	73
Case Study / Project	73
Identification concept of problem / Project	73
Lesson 1: The Decorator Pattern.....	73
Lesson 1: Overview.....	73
Lesson 2: The Observer Pattern	75
Lesson 2: Overview.....	75
Lesson 3: The Strategy Pattern.....	78
Lesson 3: Overview.....	78
Solution	80
Instruction.....	80
Exercise	82
Exercise Objectives	82
Task 1:.....	82
UNIT 5 DESIGN PATTERN 1 (2).....	83
Unit Overview	83
Unit Objectives.....	83
Unit Contents	83
Pre Lab	84
Question.....	84
Content Lesson	84
Case Study / Project	84
Identification concept of problem / Project	84
Lesson 4: The Singleton Pattern.....	84
Lesson 4: Overview.....	84
Lesson 5: The Template Pattern	87
Lesson 5: Overview.....	87
Solution	90
Instruction.....	90

Exercise	91
Exercise Objectives	91
Task 1:.....	92
UNIT 6 DESIGN PATTERN 2 (1).....	93
Unit Overview	93
Unit Objectives.....	93
Unit Contents	93
Pre Lab	94
Question.....	94
Content Lesson	94
Case Study / Project	94
Identification concept of problem / Project	94
Lesson 1: The Adapter Pattern	94
Lesson 1: Overview.....	94
Lesson 2: The Façade Pattern	96
Lesson 2: Overview.....	96
Lesson 3: The Command Pattern.....	98
Lesson 3: Overview.....	98
Solution	100
Instruction.....	101
Exercise	102
Exercise Objectives.....	102
Task 1:.....	102
UNIT 6 Design Pattern 2 (2)	103
Unit Overview	103
Unit Objectives.....	103
Unit Contents	103
Pre Lab	104
Question.....	104
Content lesson.....	104
Case Study / Project	104
Identification concept of problem / Project	104
Lesson 4: The Abstract Factory Pattern.....	104
Lesson 4: Overview.....	104
Lesson 5: The Composite Pattern	108
Lesson 5: Overview.....	108

Solution	110
Instruction.....	110
Exercise	113
Exercise Objectives	113
Task 1:.....	113
UNIT 7 TESTING OOP	114
Unit Overview	114
Unit Objectives.....	114
Unit Contents	114
Pre Lab	115
Question.....	115
Content Lesson	115
Case Study / Project	115
Identification concept of problem / Project	115
Lesson 1: Testing dengan unittest	115
Lesson 1: Overview.....	115
Lesson 2: Testing dengan Py.Test	118
Lesson 2: Overview.....	118
Solution	120
Instruction.....	120
Exercise	121
Exercise Objectives	121
Task 1:.....	121
UNIT 8 PRINSIP SOLID PADA PYTHON (1).....	122
Unit Overview	122
Unit Objectives.....	122
Unit Contents	122
Pre Lab	123
Question.....	123
Content lesson.....	123
Case Study / Project	123
Identification concept of problem / Project	123
Lesson 1: Single Responsibility Principle	123
Lesson 1: Overview.....	123
Lesson 2: Open-Closed Principle	126
Lesson 2: Overview.....	126

Solution	128
Instruction.....	128
Exercise	129
Exercise Objectives	129
Task 1:.....	130
UNIT 8 PRINSIP SOLID PADA PYTHON (2).....	131
Unit Overview	131
Unit Objectives.....	131
Unit Contents	131
Pre Lab	132
Question.....	132
Content Lesson	132
Case Study / Project	132
Identification concept of problem / Project	132
Lesson 3: Liskov Substitution Principle	132
Lesson 3: Overview.....	132
Lesson 4: Interface Segregation Principle	135
Lesson 4: Overview.....	135
Lesson 5: Dependency Inversion Principle	136
Lesson 5: Overview.....	136
Solution	138
Instruction.....	138
Exercise	141
Exercise Objectives	141
Task 1:.....	141

UNIT 1

PENGENALAN OOP

UNIT OVERVIEW

Pada pengenalan OOP kali ini kita akan membahas bagaimana cara dalam membuat sebuah kelas di dalam bahasa pemrograman Python. Hal ini tentunya menjadi sangat penting, karena dalam pengerjaan sebuah project kita biasanya berdasarkan dari kelas-kelas yang ingin kita tentukan.

UNIT OBJECTIVES

Adapun capaian yang akan kita dapatkan pada modul ini, yaitu :

- Cara membuat kelas dan instance di dalam Python.
- Cara membuat atribut pada objek Python.
- Cara mengkonversi kelas menjadi sebuah packages (bungkusan) dan modules.

UNIT CONTENTS

Lesson 1: Kelas	2-4
Lesson 2: Menambahkan Atribut	4-6
Lesson 3: Method in Class.....	6-8

PRE LAB

Sebelum masuk ke dalam lab pertama ini, terlebih dahulu kita coba jawab beberapa pertanyaan ini berdasarkan hasil penjelasan pada modul teori!

QUESTION

1. Menurut kalian apa saja bisa jadi objek?
2. Menurut kalian, apakah ciri khas dari objek tersebut?
3. Dari daftar nama-nama berikut, manakah yang bisa dibilang sebuah objek :
 - Sekolah
 - Data
 - Informasi
 - Buku
 - Pulpen

CONTENT LESSON

CASE STUDY / PROJECT

Pada sebuah perpustakaan “Home Learning is Best” terdapat berbagai jenis buku yang berbeda-beda. Pada suatu hari seorang pegawai perpus diminta oleh bosnya untuk mendaftarkan setiap buku yang ada diperpustakaan. Tentunya daftar tersebut harus dapat memuat nama, jenis, tahun terbit, penerbit, penulis serta jumlah halaman dari buku tersebut. Setelah itu, bosnya juga mau untuk nama penerbit hanya bisa diakses dari pada pendataan saja tidak boleh diketahui sama orang lain. Hal ini supaya menjaga kerahasiaan dari buku tersebut.

Bisa dibilang bosnya pegawai tersebut banyak maunya. Hanya saja pada saat ini, hal tersebut bukanlah sebuah hal yang aneh lagi. Karena, ada sebagai perusahaan sudah menerapkan seperti itu. Oleh karena itu, yuk kita mulai memahami permasalahannya terlebih dahulu!

IDENTIFICATION CONCEPT OF PROBLEM / PROJECT

Setelah kita baca kembali studi kasus yang diberikan tersebut, tentunya ada beberapa hal yang harus kita perhatikan, yaitu menentukan objek yang terdapat di dalam studi kasus tersebut. Setelah itu, kita juga harus menentukan kira-kira apa saja ciri khusus dari objek tersebut. Terakhir apa yang harus dilakukan ketika kita sudah bisa membuat kelas tersebut, apakah ada tindakan yang perlu dilakukan oleh pegawai tersebut?

Sebelum kita menjelaskan solusi dari permasalahan tersebut, mari kita simak dulu beberapa penjelasan untuk membantu kita dalam menyelesaikan permasalahan yang terdapat dalam studi kasus tersebut !

LESSON 1: KELAS

LESSON 1: OVERVIEW

Kelas merupakan sebuah objek yang terdapat pada bahasa pemrograman OOP yang memiliki ciri khusus serta dapat diakses oleh pengguna. Bagaimana cara membuat kelas dalam bahasa Pemrograman Python? Pada pelajaran pertama ini kita akan membahas bagaimana cara untuk membuat kelas dalam bahasa pemrograman Python.

Untuk perintah yang dapat kita gunakan untuk membuat 1 buah kelas dalam bahasa Pemrograman Python, yaitu :

```
class MyFirstClass:  
    pass
```

Figure 1 Pembuatan MyFirstClass

Sangatlah simple bukan? Pada awal kelas kita perlu menuliskan kata *class*, sebagai penanda bahwa itu adalah sebuah kelas. Setelah itu kita ketikkan nama kelas yang kita inginkan. Sebagai contoh disana, saya menulis **MyFirstClass**. Dalam penamaan sebuah kelas, ada beberapa aturan yang harus kita ikuti, yaitu :

- Untuk permulaan nama kelas harus dimulai dengan abjad (A-Z atau a-z) atau bisa juga diawali dengan tanda *underscore* (“_”).
- Untuk nama kelas yang lebih dari 2 suku kata, biasanya kita bisa menerapkan prinsip dari *CamelCase*. Sebagai contoh : NamaSaya. Hal ini bertujuan supaya lebih memudahkan kita dalam mengingat nama kelas tersebut.
- Pastikan kembali nama kelas tersebut belum ada sebelumnya pada sistem. Sebagai contoh kelas Point, dimana sudah ada di dalam Python. Sehingga, kita tidak perlu lagi membuatnya, hanya menggunakan saja.

Setelah kita memberikan nama kelas, jangan lupa kita memberikan tanda titik dua (“：“) untuk mengakhiri penulisan kelas. Pada bagian bawahnya (isinya) kita ketikkan saja sebuah argumen *pass*. Bagian ini akan kita perjelas lagi pada lesson berikutnya. Setelah semua sudah ok, silahkan simpan nama file tersebut dengan nama **KelasSaya.py**. Setelah itu, muncul pertanyaan baru, gimana cara jalankan perintah tersebut ke dalam Visual Studio Code?

Terlebih dahulu kita harus masuk ke dalam folder yang memuat file **KelasSaya.py** tersebut dengan perintah **cd [lokasi folder]**. Setelah itu ketikkan perintah berikut :

```
\Contoh\Pert-01> Python -i KelasSaya.py
```

Figure 2 Perintah untuk menjalankan kelas

Pada bagian awal kita ketikkan kata **Python** yang merupakan Aplikasi Python yang terpasang di dalam laptop kita. Setelah itu kita perlu menambahkan argumen **-i**, yang digunakan sebagai interpreter (penerjemah) yang berfungsi untuk mengeksekusi sejumlah intruksi yang tertulis dalam bahasa Pemrograman Python. Setelah itu, pada bagian akhir, kita tuliskan **nama file** tempat kelas kita berisi. Setelah semua sudah sesuai, jangan lupa tekan tombol **Enter** pada *keyboard* untuk menjalankan perintah.

NB : Apabila ada error berupa : **ModuleNotFoundError: No module named 'typing_extensions'**, maka jalan perintah ini terlebih dahulu : **pip install typing_extensions**.

Setelah jalan, ketikkan baris berikut :

```
a = MyFirstClass()  
b = MyFirstClass()
```

Figure 3 Perintah untuk mengakses kelas

Perintah tersebut digunakan untuk mengakses sebuah kelas. Pertama-tama kita harus menginisialisasi sebuah variabel terlebih dahulu. Seperti contoh diatas, saya menggunakan nana variabel *a* dan *b* untuk dapat mengakses kelas *MyFirstClass* yang kita buat sebelumnya. Setelah itu, apabila kita ingin memastikan kelas tersebut memang sudah bisa diakses, kita hanya perlu menjalankan perintah print, dan akan muncul tampilan berikut :

```
>>> a = MyFirstClass()
>>> b = MyFirstClass()
>>> print(a)
<__main__.MyFirstClass object at 0x0000025573BFD4E0>
>>> print(b)
<__main__.MyFirstClass object at 0x0000025573BFD550>
```

Figure 4 Hasil tampilan akses dan print kelas

Nah terlihat mudah bukan? Yuk coba-coba buat kelas kalian masing-masing supaya lebih bisa memahami apa yang harus dilakukan!

Setelah kita belajar ini, berikutnya kita akan memahami pendeklarasian sebuah atribut ke dalam kelas. Gimana caranya ya? Yuk kita simak Lesson berikutnya!

LESSON 2: MENAMBAHKAN ATRIBUT

LESSON 2: OVERVIEW

Setelah kita belajar mengenai kelas, tentunya hanya kelas saja tentunya tidak bisa melakukan atau menyimpan sesuatu. Oleh karena itu, kita perlu menambahkan atribut ke dalam kelas tersebut. Ada 3 cara yang bisa gunakan untuk menambahkan atribut ke dalam kelas tersebut, yaitu :

Cara 1: Secara Langsung

```
>>> a = MyFirstClass()
>>> a.x = 10
>>> a.y = 11
>>> print(a.x)
10
>>> print(a.y)
11
```

Figure 5 Menambahkan Atribut Cara 1

Pada cara ini, kita hanya perlu menuliskan langsung nama atribut yang ingin kita berikan. Hanya saja ada ketentuan dalam penulisan nama atribut, yaitu [nama variable kelas](dot/titik)[nama atribut]. Sebagai contoh diatas *a.x*.

Cara 2: Melalui inisialisasi nilai awal ke dalam kelas

Terlebih dahulu kita ubah kelas kita sebelumnya menjadi seperti berikut :

```
class MyFirstClass :
    x = 10
    y = 10
```

Figure 6 Mengubah MyFirstClass

Setelah itu, kita akan mencoba mengakses variable tersebut seperti tampilan berikut :

```
>>> a = MyFirstClass()
>>> a.x
10
>>> a.y
10
```

Figure 7 Menambahkan Atribut Cara 2

Pada cara ini kita tidak perlu lagi membuat 1 buah atribut baru atau memasukkan nilai ke dalam atribut. Kita hanya perlu memanggilnya atribut dengan cara yang sama.

Cara 3: Memasukkan pada inisialisasi

Untuk proses ininisialisasi ini, kita perlu menambahkan sebuah method (**double underscore**)`init(double underscore)`. Contoh : `__init__`. Method init berfungsi sebagai kelas utama yang berjalan ketika proses inisialisasi kelas.

Untuk menjalankan kita perlu mengubah kelas kita tadi yang bernama `MyFirstClass` menjadi, seperti tampilan berikut :

```
class MyFirstClass :
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

Figure 8 MyFirstClass dengan init

Pada kelas tersebut terdapat 3 atribut, berupa : `self`, `x`, dan `y`. `Self` disini digunakan sebagai penanda dari kelas yang kita buat saat ini. Jadi, apabila kita menggunakan `self`, lebih kepada `MyFirstCalss.x`. Sedangkan atribut `x` dan `y`, merupakan atribut tambahan yang kita tambahkan ke dalam kelas.

Untuk mengakses atribut dalam kelas tersebut, juga sama seperti cara sebelumnya, kita hanya perlu menggunakan langkah berikut :

```
>>> a = MyFirstClass(10, 20)
>>> a.x
10
>>> a.y
20
```

Figure 9 Menambahkan Atribut Cara 3

Ketiga cara tersebut bisa kalian gunakan secara bersamaan dalam kode. Hanya penting untuk dipahami pada konteks penggunaannya, jangan sampai salah cara membuat dan mengaksesnya.

Pada saat pengaksesan pada atribut kelas Python, kita juga bisa membuat atribut tersebut hanya bisa diakses dari dalam kelas atau biasa kita bilang atribut *private* (tidak bisa diakses dari luar kelas), dengan menambahkan `__` (**double underscore**) pada awal nama atribut tersebut. Sebagai contoh : `__x`

Saya merubah sedikit kelas sebelum menjadi seperti ini :

```
class MyFirstClass :  
    def __init__(self, x, y):  
        self.__x = x  
        self.y = y
```

Figure 10 Mengubah atribut MyFirstClass

Pada contoh tersebut, coba perhatikan pada bagian atribut x. Saya ada menambahkan __ ke dalam atribut tersebut. Coba kita jalankan programnya :

```
>>> a = MyFirstClass(10,20)  
>>> a.x  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
AttributeError: 'MyFirstClass' object has no attribute 'x'  
>>> a.y  
20
```

Figure 11 Hasil Penambahan double underscore

Dapat terlihat ketika kita ingin mengakses atribut x, tidak bisa diakses. Sehingga, kita perlu cara ekstra untuk mengakses atribut tersebut. Adapun cara tambahan yang harus kita lakukan, dengan menggunakan [nama variable].(single underscore)[nama kelas](double underscore)[Nama atribut]. Sebagai contoh : a._MyFirstClass__x

```
>>> a = MyFirstClass(10,20)  
>>> a.x  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
AttributeError: 'MyFirstClass' object has no attribute 'x'  
>>> a._MyFirstClass__x  
10
```

Figure 12 Mengakses Atribut Private

LESSON 3: METHOD IN CLASS

LESSON 3: OVERVIEW

Method atau fungsi bisa juga kita deklarasikan di dalam kelas. Hal ini tentunya sangat membantu kita, apalagi kalau fungsi tersebut dipakai lintas program atau lintas kode. Mari kita simak cara pembuatan fungsi di dalam kelas serta gimana cara untuk mengaksesnya.

Perhatikan kelas berikut dan simpan dengan nama PersegiPanjang.py :

```
class persegiPanjang:
    def __init__(self, p, l):
        self.p = p
        self.l = l
    def luas(self):
        return self.p * self.l
    def keliling(self):
        print(f"Keliling = {2*(self.p +self.l)}")
    def hasil(self, p, l):
        print(f"Panjang = {p}")
        print(f"Lebar = {l}")
        print(f"Luas = {p*l}")
        print(f"Keliling = {2*(p+l)}")
```

Figure 13 Kelas dengan Fungsi dan Method

Pada kelas tersebut terdapat 1 inisialisasi kelas, 1 fungsi dan 2 method. Mana yang merupakan inisialisasi kelas, mana yang merupakan Method dan mana yang merupakan Fungsi? Seperti yang telah kita pelajari sebelumnya, dimana :

- Inisialisasi, yaitu `__init__` yang merupakan kelas dijalankan pada saat proses inisialisasi.
- Fungsi, yaitu `luas(self)` yang merupakan sebuah proses yang memiliki nilai keluaran (`return`).
- Method, yaitu `keliling(self)` dan `hasil(self,p,l)` yang merupakan sebuah proses yang tidak memiliki nilai keluaran (`return`).

Nah setelah kita membuat kode tersebut, gimana cara kita mengakses kelasnya? Perhatikan potongan eksekusi kode berikut :

```
>>> a = persegiPanjang(10,20)
>>> print(f"Luas = {a.luas()}")
Luas = 200
>>> a.keliling()
Keliling = 60
>>> a.hasil(15,25)
Panjang = 15
Lebar = 25
Luas = 375
Keliling = 80
```

Figure 14 Pengaksesan Fungsi

Untuk mengakses kita hanya perlu menggunakan perintah `[nama variable kelas].[nama fungsi/method](inputan sesuai atribut yang ada pada kelas)`. Sebagai contoh : `a.keliling()`.

Hal ini tentunya sangat memudahkan kita untuk mengakses fungsi atau method yang terdapat di dalam kelas. Namun, bagaimana cara kita menambahkan fungsi atau method yang berasal dari luar kelas? Solusinya adalah menggunakan teknik **import** kelas.

Terlebih dahulu kita buat sebuah file dengan nama main.py dan isikan kode berikut :

```
from PersegiPanjang import persegiPanjang
n = int(input("Panjang = "))
m = int(input("Lebar = "))

a = persegiPanjang(n,m)
print(f"Luas = {a.luas()}")
a.keliling()
a.hasil[20,30]
```

Figure 15 Kode main.py

Pada kode baris tersebut ada 1 bagian paling atas yang baru, yaitu **from PersegiPanjang import persegiPanjang**. Maksud dari baris tersebut apa ya? Ini merupakan teknik yang kita gunakan untuk melakukan import kelas ke dalam file utama. Adapun arti dari baris tersebut adalah :

```
from [nama file] import [nama kelas/ nama fungsi/ method yang ingin kita gunakan]
```

Tentunya hal ini sangat membantu kita dalam melakukan manajemen terhadap kode. Dimana kode untuk SQL kita simpan di dalam 1 kelas, terus kode yang mengelola data kita simpan lagi dalam 1 kelas yang berbeda, sehingga kita bisa dengan mudah untuk memahami arti dari setiap kode dan cara mengaksesnya.

SOLUTION

Setelah kita membaca berbagai macam topik diatas, yuk kita kembali ke studi kasus kita diawal.

Nah setelah kita baca kembali studi kasus tersebut, tentunya kita harus bisa menjelaskan terlebih dahulu mana objek yang terdapat di dalam perpustakaan tersebut dan objek yang terdapat di dalam perpustakaan tersebut merupakan **BUKU**. Pada buku tersebut nantinya terdapat nama, jenis, tahun terbit, penerbit, penulis serta jumlah halaman. Itu semua merupakan bagian behaviour dari sebuah buku atau bisa kita bilang atribut dari buku tersebut. Namun, ada 1 atribut, yaitu nama yang harus menggunakan private. Jadi, jangan sampai lupa. Setelah itu, si pegawai harus bisa mencetak semua buku, sehingga kita membutuhkan sebuah fungsi untuk dapat mencetak isi buku tersebut.

Nah supaya lebih jelas, yuk kita bahas 1 per 1.

INSTRUCTION

1. Tahap awal untuk menyelesaikan masalah tersebut kita membuat kelas BUKU sesuai dengan ketentuan tersebut. Jadi, untuk perintah kita hanya perlu mengetikkan :

```
class BUKU:
```

2. Setelah kita membuat mendeklarasikan kelas tersebut, sekarang kita tambahkan atribut yang diperlukan sesuai dengan studi kasus yang diberikan.

```
def __init__(self, nama, jenisBuku, tahunTerbit, penerbit, penulis, jumlahHalaman):
    self.__nama = nama ##karena bersifat private
    self.jenisBuku = jenisBuku
    self.tahunTerbit = tahunTerbit
    self.penerbit = penerbit
    self.penulis = penulis
    self.jumlahHalaman = jumlahHalaman
```

NB : tanda ## merupakan komentar pada Python.

3. Setelah kita membuat pemasukan atribut, kita buat fungsi untuk mencetak semua bagian yang dibuat :

```
def cetak(self):
    print(f"Nama Buku      = {self.__nama}")
    print(f"Jenis Buku      = {self.jenisBuku}")
    print(f"Tahun Terbit Buku = {self.tahunTerbit}")
    print(f"Penerbit Buku    = {self.penerbit}")
    print(f"Penulis Buku     = {self.penulis}")
    print(f"Jumlah Halaman Buku = {self.jumlahHalaman}")
```

4. Untuk mengakses kelas tersebut, kita lakukan ketikkan perintah berikut :

```
buku = BUKU("Pemrograman Python", "Pemrograman", 2021, "No-Name", "No-Name", 120)
buku.cetak()
```

EXERCISE

EXERCISE OBJECTIVES

Pada latihan berikut ini mahasiswa diharapkan dapat melakukan :

- Melakukan analisis masalah terhadap studi kasus yang diberikan.
- Mencari solusi atas masalah yang diberikan.
- Menerjemahkan masalah tersebut ke dalam bentuk kode pemrograman.

TASK 1:

Pada sebuah kelas di Kampus Mikroskil terdapat mahasiswa dan dosen pada kelas tersebut. Mahasiswa tersebut tentunya memiliki data berupa (*bagian ini kalian cari tahu sendiri*). Sedangkan dosen memiliki data berupa NIP, Nama, jabatan, jenis kelamin, dan no HP. Pada suatu acara, seorang mahasiswa ditujukan untuk melakukan rekap absensi dari acara tersebut. Absensi tersebut nantinya harus bisa dicetak serta bisa dihitung kira-kira berapa banyak jumlah pengunjung pada acara tersebut. Bisakah kalian membantu mahasiswa tersebut?

TASK 2:

Pada sebuah website E-Commerce terdapat berbagai jenis barang yang dijual, diantaranya hand sanitizer, ban mobil, buah-buahan serta botol minuman bayi. Tentunya barang yang dijual tersebut tidak lebih dari 1 barang sehingga penjual harus tahu kira-kira barang apa saja yang mau habis dan mencetak setiap laporan pengeluaran bulanan yang dilakukan. Dapatkan Anda membantu penjual tersebut?

TASK 3:

Pada sebuah private les terdapat sejumlah murid yang diajarkan. Murid-murid tersebut, memiliki Nama, jenis kelamin, kelas, dan jenis les yang berikan. Hal ini tentunya tidak terlepas dari pada biaya les yang ditawarkan. Adapun daftar biaya les yang ditawarkan berdasarkan tingkat pendidikan yang diajarkan, yaitu :

Tingkatan	Biaya Les
TK	Rp. 300.000,00
SD	Rp. 500.000,00
SMP	Rp. 700.000,00
SMA	Rp. 1.000.000,00

Untuk jam pembelajaran yang diberikan juga didasarkan dari tingkatan yang diajarkan :

Tingkatan	Jam Pengajaran
TK	2 jam
SD	2 jam
SMP	1 jam
SMA	1 jam

Setelah itu, untuk setiap murid yang berhasil mendaftar diberikan sebuah kartu yang dicetak. Dan dikartu tersebut terdapat data berupa : Nama, Tingkatan, Jam Pengajaran dan Biaya Les.

Dapatkan Anda membantu private les tersebut?

UNIT 2

INHERITANCE DAN POLYMORPHISM (1)

UNIT OVERVIEW

Pada topik kali ini kita akan membahas cara untuk melakukan efisiensi terhadap kode yang kita buat. Hal ini bertujuan supaya penerapan yang kita lakukan tepat pada sasaran serta dapat digunakan kembali untuk kelas-kelas lainnya.

UNIT OBJECTIVES

Adapun capaian yang akan kita dapatkan pada modul ini, yaitu :

- Cara membuat kelas induk dan turunan di dalam Python.
- Cara memanggil fungsi `__init__` yang ada pada kelas induk

UNIT CONTENTS

Lesson 1: Inheritance.....	13-
14	
Lesson 2: Fungsi Super	14-
15	

PRE LAB

Sebelum masuk ke dalam lab kedua ini, terlebih dahulu kita coba jawab beberapa pertanyaan ini berdasarkan hasil penjelasan pada modul teori!

QUESTION

1. Menurut kalian apa saja dimaksud dengan kelas turunan?
2. Menurut kalian mengapa kita perlu menerapkan konsep kelas turunan?
3. Dari daftar nama-nama berikut, kira-kira atribut apa yang bisa kita samakan :
 - Mahasiswa
 - Dosen
 - Staf Kampus

CONTENT LESSON

CASE STUDY / PROJECT

Pada perputakaan sebelum yang memiliki nama “Home Learning is Best”, kini akan memperlebar sayapnya sampai kepada beberapa daerah yang ada Indonesia. Hal ini bertujuan supaya dapat menyelesaikan salah satu permasalahan yang ada di Indonesia, yaitu **mencerdaskan kehidupan bangsa Indonesia** dengan cara membaca buku. Namun, muncul masalah baru untuk setiap daerahnya yang ada berupa minat masyarakat yang berbeda-beda. Hal ini dikarenakan keragaman suku budaya yang berbeda. Seperti pada daerah Medan, karena kebanyakan bersuku Batak sehingga membutuhkan pengajaran dalam mengulos. Pada ulos tersebut banyak sekali jenisnya bergantung kepada daerah-daerahnya serta jenis alat yang digunakan dan jangan tindakan untuk melakukan ulos. Begitu juga dengan daerah Jawa, karena kebanyak bersuku Sunda, sehingga memerlukan pengajaran dalam membatik. Pada batik juga ada beberapa hal yang harus diperhatikan, yaitu daerah asal batik tersebut dibuat, jumlah alat cangkringan yang dipakai, serta jangan lupa untuk melakukan tindakan membantik. Sehingga, si pegawai tersebut harus merancang kembali supaya masalah yang dihadapainya dapat terselesaikan.

Hal ini tentunya sering terjadi dalam kehidupan kita sehari-hari. Oleh karena itu, kita perlu memikirkan kembali semua sebelum kita melakukan perancangan supaya kita tidak melakukan dua kali kerja. Namun, mari kita bantu kembali pegawai tersebut !

IDENTIFICATION CONCEPT OF PROBLEM / PROJECT

Setelah kita membaca kembali studi kasus tersebut, terdapat hal tentunya harus kita pahami, yaitu terdapat daerah yang berbeda serta memiliki keunikan atau ciri khas yang berbeda juga sesuai dengan daerahnya. Semisalnya kita membuat dua kelas untuk Medan dan Jawa apakah sudah cukup tanpa harus membuat kelas Perpustakaan? Apakah ini sudah sesuai dengan harapan yang kita perlukan?

Sebelum kita menjelaskan solusi dari permasalahan tersebut, mari kita simak dulu beberapa penjelasan untuk membantu kita dalam menyelesaikan permasalahan yang terdapat dalam studi kasus tersebut !

LESSON 1: INHERITANCE

LESSON 1: OVERVIEW

Kelas turunan yang merupakan konsep dalam OOP dalam bentuk mewariskan tingkah laku atau ciri khas dari *super class* kepada *sub class*. Tingkah laku tersebut dapat berupa atribut ataupun method/fungsi yang ada pada super kelas. Hal ini tentunya dapat memudahkan pengguna dalam menerapakan efisensi di dalam kode yang dibuat. Sebelum kita membuat *subClass*, terlebih dahulu kita buat *superClass* untuk kelas segiEmpat:

```
class segiEmpat:
    sisi1 = 0
    sisi2 = 1
    sisi3 = 2
    sisi4 = 3
    def keliling(self):
        return self.sisi1 + self.sisi2 + self.sisi3 + self.sisi4
```

Figure 16 Kelas Utama (SuperClass)

Setelah kita membuat *superClass* tersebut, berikut kita menambahkan *subClass* pada untuk kelas persegi. Karena, persegi juga merupakan kelas segiEmpat. Untuk membuat *subClass* dari *superClass* pada Python langkah yang harus kita lakukan ada *class subClass(superClass)*. Oleh karena itu, berikut bentuk tampilan untuk kelas persegi :

```
class persegi(segiEmpat):
    pass
```

Figure 17 subClass persegi

Dari kode tersebut, kita tidak perlu menambahkan apapun ke dalam kelas tersebut, karena kita hanya ingin melihat penerapan pada kode tersebut.

Untuk menjalankan seperti biasa kita menggunakan perintah **Python -i SuperClass.py** (karena, nama file yang saya buat menggunakan SuperClass.py). Adapun hasil jalannya seperti berikut :

```
>>> a = segiEmpat()
>>> b = persegi()
>>> print(a)
<__main__.segiEmpat object at 0x000001A946299D68>
>>> print(b)
<__main__.persegi object at 0x000001A946299DA0>
```

Figure 18 Objek superClass dan subClass

Pada objek yang tertampilkan tidak terlihat penerapan pada *superClass* dan *subClass*. Namun, bagaimana kalau kita mengakses atribut yang ada pada *superClass* dari *subClass* berikut :

```
>>> print(b.sisi4)
3
```

Figure 19 Mengakses Atribut superClass

Terlihat sedikit aneh. Karena, pada kelas persegi sebelumnya kita tidak ada membuat atribut sisi4 dan memasukkan nilai 3, tetapi ketika dicetak bisa menghasilkan nilai 3 tersebut. Namun, coba kita lihat kembali pada kelas segiEmpat yang merupakan *superClass* dari persegi, disana kita ada membuat atribut sisi4 dan kita masukkan juga nilainya 3. Hal ini tentunya dapat memperlihatkan penerapan dari *Inheritance* tersebut, dimana kita bisa menurunkan atribut *superClass* kepada *subClass*.

Lalu bagaimana dengan fungsi yang kita buat pada *superClass*, apakah bisa diturunkan juga kepada *subClass*. Perhatikan kode berikut :

```
>>> print(f"Keliling = {b.keliling()}")
Keliling = 6
```

Figure 20 Mengakses Fungsi *superClass*

Ternyata bisa. Hal ini membuat kita menjadi lebih tahu, bahwa hal yang bisa diturunkan dari *superClass* ke *subClass*, yaitu atribut dan fungsi/method.

Terus, bagaimana kalau kita buat fungsi yang ada pada *subClass*, apakah bisa diakses oleh *superClass*? Kita coba dengan mengubah sedikit kelas persegi kita menjadi seperti berikut :

```
class persegi(segiEmpat):
    def luas(self, s):
        return s * s
```

Figure 21 Mengubah kelas *persegi*

Setelah kita mengubah, kita jalankan kembali seperti berikut :

```
>>> a = segiEmpat()
>>> b = persegi()
>>> print(f"Luas = {b.luas(50)} cm")
Luas = 2500 cm
>>> print(f"Luas = {a.luas(50)} cm")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'segiEmpat' object has no attribute 'luas'
```

Figure 22 Hasil Pengubahan Kode

Dari kode tersebut telihat bahwa, kita bisa menurunkan atribut atau method/fungsi ke dalam kelas turunan namun, kita tidak bisa mengakses tersebut dari *superClass*.

Nah muncul pertanyaan lagi, bagaimana cara kita bisa memanggil fungsi inisialisasi yang ada pada kelas Induk? Hal ini akan kita pelajari pada lesson berikutnya!

LESSON 2: FUNGSI SUPER

LESSON 2: OVERVIEW

Pada saat tertentu, tentunya pastinya ada sebuah fungsi yang biasanya mengharus kita untuk mengakses fungsi pada kelas induk (*superClass*) di dalam kelas turunan (*subClass*). Hal ini bertujuan supaya kita tidak perlu lagi memanggil *superClass*, kita langsung mengakses saja pada *subClass*. Namun, bagaimana cara supaya kita bisa memanggil *superClass* di dalam *subClass*? Caranya adalah dengan menggunakan fungsi *Super()*. Fungsi *Super()* diibaratkan sebagai pengganti kelas induk.

Sehingga, apabila kita ingin memanggil fungsi/method atau atribut yang ada pada superClass, yang kita lakukan hanya perlu super().[nama atribut atau fungsi/method]. Perhatikan contoh dua kelas berikut :

```
class segiEmpat:
    def __init__(self, p, l):
        self.p = p
        self.l = l
    def luas(self):
        return self.p * self.l

class persegi(segiEmpat):
    def __init__(self, p, l):
        super().__init__(p, l)
```

Figure 23 Melakukan pemanggilan Super

Dari contoh tersebut terdapat kelas segiEmpat sebagai superClass, dan kelas persegi sebagai subClass. Pada kelas persegi, kita ada memanggil super untuk memanggil fungsi `__init__` yang ada pada superClass. Hal ini supaya kita tidak perlu lagi memanggil superClass pada saat menjalankan kode program yang kita buat. Sehingga, untuk menjalankan kita hanya perlu melakukan berikut :

```
>>> a = persegi(20,30)
>>> print(f"Luas = {a.luas()} cm pangkat 2")
Luas = 600 cm pangkat 2
```

Figure 24 Hasil Super

Sesuai dengan hasil tersebut, tentunya kita tidak perlu lagi memanggil *superClass* pada kode yang kita jalankan. Kita hanya perlu memanggil *subClass*, dan secara otomatis sudah terupdate pada *superClass*.

NB : Perintah penggunaan super ini hanya bisa dilakukan pada Python versi 3. Jadi, apabila kalian ingin melakukan pada Python versi 2, pastikan kembali strukturisasi pemanggilannya.

SOLUTION

Setelah kita membaca berbagai macam topik diatas, yuk kita kembali ke studi kasus kita diawal.

Nah setelah kita baca kembali studi kasus tersebut, tentunya kita harus bisa menjelaskan terlebih dahulu mana objek yang terdapat di dalam perpustakaan tersebut dan objek yang terdapat di dalam perpustakaan tersebut merupakan **perpus**. Pada objek tersebut terdapat atribut berupa namaDaerah sebagai inisialisasi daerah tempat perpus tersebut berada.

Setelah itu, untuk daerah Medan, kita ada beberapa atribut baru berupa namaSuku, jenisPerlatalan serta tindakan berupa mengulos. Sedangkan untuk daerah Jawa, kita ada namaAsal, jumlahCangkringan, dan tindakan berupa membatik.

Nah supaya lebih jelas, yuk kita bahas 1 per 1.

INSTRUCTION

1. Tahap awal untuk menyelesaikan masalah tersebut kita membuat kelas **perpus** beserta atributnya sesuai dengan analisis yang kita lakukan sebelumnya, sehingga dihasilkan berupa :

```
class perpus:  
    def __init__(self, namaDaerah):  
        self.namaDaerah = namaDaerah
```

2. Tahap berikutnya kita membuat kelas untuk Medan dan jangan lupa untuk membuat kelas induknya berasal dari kelas perpus dengan kode seperti berikut :

```
class Medan(perpus):  
    def __init__(self, namaDaerah, namaSuku, jenisPeralatan):  
        super().__init__(namaDaerah)  
        self.namaSuku = namaSuku  
        self.jenisPeralatan = jenisPeralatan  
    def mengulos(self):  
        print(f"Perpustakaan {self.namaDaerah}")  
        print(f"Nama Suku      = {self.namaSuku}")  
        print(f"Jenis Peralatan = {self.jenisPeralatan}")
```

3. Tahap terakhir jangan lupa untuk membuat kelas Jawa, dan pastinya jangan lupa menghubungkannya juga dengan kelas induk perpus, sehingga kodennya seperti berikut :

```
class Jawa(perpus):  
    def __init__(self, namaDaerah, namaAsal, jumlahCangkringan):  
        super().__init__(namaDaerah)  
        self.namaAsal = namaAsal  
        self.jumlahCangkringan = jumlahCangkringan  
    def membatik(self):  
        print(f"Perpustakaan {self.namaDaerah}")  
        print(f"Nama Asal      = {self.namaAsal}")  
        print(f"Jumlah Cangkringan = {self.jumlahCangkringan} buah")
```

4. Setelah semuanya sudah berhasil dibuat, jangan lupa untuk menjalankan kelas tersebut dan lihat hasilnya gimana :

```
>>> mdn = Medan("Medan", "Batak Toba", "Meja Jahit")  
>>> mdn.mengulos()  
Perpustakaan Medan  
Nama Suku      = Batak Toba  
Jenis Peralatan = Meja Jahit  
>>> jawa = Jawa("Jawa", "Jawa Barat", 10)  
>>> jawa.membatik()  
Perpustakaan Jawa  
Nama Asal      = Jawa Barat  
Jumlah Cangkringan = 10 buah
```

Figure 25 Hasil Studi Kasus

EXERCISE

EXERCISE OBJECTIVES

Pada latihan berikut ini mahasiswa diharapkan dapat melakukan :

- Melakukan analisis masalah terhadap studi kasus yang diberikan.
- Mencari solusi atas masalah yang diberikan.
- Menerjemahkan masalah tersebut ke dalam bentuk kode pemrograman.

TASK 1:

Pada sebuah kelas di Kampus Mikroskil terdapat mahasiswa dan dosen. Data mahasiswa dan dosen tersebut memiliki jenis data yang sama berupa, nomor induk, nama, jenis kelamin, serta no HP. Mahasiswa tersebut tentunya memiliki data berupa (*bagian ini kalian cari tahu sendiri*). Dosen dengan mahasiswa memiliki tindakan yang berbeda. Dimana mahasiswa memiliki tindakan berupa absensi yang memunculkan nomor induk dan namanya. Sedangkan dosen memiliki tindakan berupa perkenalan yang memunculkan nama dan no HPnya.

Keterangan tambahan : Untuk kelas Induknya, silahkan dibuat berdasarkan [namaAnda]. Hindari mengcopy dan paste jawaban teman Anda. Karena, apabila ketahuan akan dikenakan sangsi berupa pengurangan nilai untuk yang mengcopy jawaban.

TASK 2:

Pada sebuah website E-Commerce terdapat berbagai jenis barang yang dijual, diantaranya hand sanitizer, ban mobil, buah-buahan serta botol minuman bayi. Tentunya barang yang dijual tersebut tidak lebih dari 1 barang sehingga penjual harus tahu kira-kira barang apa saja yang mau habis dan mencetak setiap laporan pengeluaran bulanan yang dilakukan. Dapatkan Anda membantu penjual tersebut?

Keterangan tambahan : Untuk kelas induknya, buatlah [namaOrangYangKalianKenal]. Hindari mengcopy dan paste jawaban teman Anda. Karena, apabila ketahuan akan dikenakan sangsi berupa pengurangan nilai untuk yang mengcopy jawaban.

UNIT 2

INHERITANCE DAN POLYMORPHISM (2)

UNIT OVERVIEW

Pada chapter kali ini kita akan membahas bagaimana cara membuat sebuah fungsi yang dapat di daur ulang menjadi lebih spesifik dan lebih rinci sehingga bisa lebih jelas fungsi untuk digunakan. Pada chapter ini juga kita akan mempelajari kelas abstrak yang berfungsi memberikan karakteristik yang harus dipenuhi kelas keturunan tersebut, sehingga tujuan dari pembuatan kelas dapat tercapai.

UNIT OBJECTIVES

Adapun capaian yang akan kita dapatkan pada modul ini, yaitu :

- Membuat method overriding dan overloading.
- Membuat kelas abstraks.

UNIT CONTENTS

Lesson 3: Polymorphism	20-
	22
Lesson 4: Membuat Kelas Abstract	22-
	25

PRE LAB

Sebelum masuk ke dalam lab pertama ini, terlebih dahulu kita coba jawab beberapa pertanyaan ini berdasarkan hasil penjelasan pada modul teori!

QUESTION

1. Menurut kalian apa yang bisa kita sebut sebagai inheritance?
2. Menurut kalian, apakah ayah kepada paman merupakan inheritance?
3. Dari daftar nama-nama berikut, kelas induk apa yang bisa kita buat untuk menampung objek-objek berikut :
 - Masker
 - Hand sanitizer
 - Desinfektan
 - Tisu basah
 - Alkohol

CONTENT LESSON

CASE STUDY / PROJECT

Kita kembali pada permasalahan yang dihadapi oleh perpustakaan “Home Learning is Best”. Karena, sebelumnya sudah membuka cabang pada beberapa daerah dan setiap daerah memiliki ciri khas yang berbeda, perpustakaan tersebut mengalami kendala dalam pencetakan brosur dan ketidak konsistenan dalam data serta tindakan yang dilakukan. Oleh karena itu, untuk setiap cabangnya, perpustakaan tersebut mengharuskan :

1. Harus mempunyai 1 buah data berupa data buku, dalam bentuk nama buku.
2. Harus dapat mencetak buku tersebut supaya bisa melakukan pendataan yang disesuaikan dengan daerah masing-masing, seperti :
 - a. Daerah Jawa, harus bisa mencetak jenis gamelan yang ada.
 - b. Daerah Batak, harus bisa mencetak jenis ulos yang dibuat.

Tentunya hal ini bertujuan supaya proses yang dilakukan dapat berjalan dengan lancar serta bisa diproses seperti perpustakaan pusat dan mudah untuk dikelola.

Nah kali ini agak sedikit berbeda karena ada syarat yang harus dipenuhi untuk setiap cabangnya. Gimana ya cara membuat syarat tersebut di dalam kelas?

IDENTIFICATION CONCEPT OF PROBLEM / PROJECT

Setelah kita baca kembali studi kasus tersebut, ada beberapa hal yang harus diperhatikan, yaitu ada syarat yang harus dipenuhi serta ada proses yang berbeda namun memiliki tujuan yang sama. Nah, pertanyaan muncul bagaimana cara kita membuat syarat? Terus, gimana cara buat dua tindakan yang sama tapi memiliki proses yang berbeda?

Dari pada pusing, yuk simak penjelasan berikut !

LESSON 3: POLYMORPHISM

LESSON 3: OVERVIEW

Seperti yang telah dijelaskan diawal, dimana polymorphism merupakan sebuah proses yang dapat kita gunakan untuk membuat 2 buah tindakan dengan nama yang sama namun memiliki proses dan tujuan yang mungkin berbeda. Kenapa bisa begitu? Dan gimana caranya? Sebelum kita lanjut, kita lihat baris kode yang ada dibawah ini :

```
class kakek:  
    def panggil(self):  
        print("Panggil saya kakek ya!")  
  
class ayah(kakek):  
    def panggil(self):  
        print("Panggil saya ayah donk!")  
  
class paman(kakek):  
    def panggil(self, namaIstri):  
        print(f"Tante saya bernama {namaIstri}")
```

Figure 26 Contoh Dasar Polymorphism

Pada contoh tersebut terdapat 1 kelas induk, yaitu kakek dan 2 kelas keturunan, yaitu ayah dan paman. Dimana kelas ayah dan kelas paman merupakan keturunan dari kelas kakek. Hanay saja, ada yang membedakan dari kelas tersebut, yaitu method panggil yang digunakan. Dimana setiap kelas keturunan memiliki nama method yang sama. Apakah ini bisa berjalan?

Yuk coba kita jalankan terlebih dahulu, dan kita lihat gimana hasilnya! Silahkan perhatikan data berikut :

```
>>> k = kakek()  
>>> a = ayah()  
>>> p = paman()  
>>> k.panggil()  
Panggil saya kakek ya!  
>>> a.panggil()  
Panggil saya ayah donk!  
>>> p.panggil()  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: panggil() missing 1 required positional argument: 'namaIstri'  
>>> p.panggil("Donita")  
Tante saya bernama Donita  
>>> █
```

Figure 27 Hasil Kode Polymorphism

Dari hasil kode yang kita jalankan, ada sedikit keanehan pada kelas ayah dan kelas paman. Dimana hasil diberikan tidak sama. Mengapa demikian? Inilah merupakan konsep dari kelas Morphism yang bertujuan untuk membuat nama method yang sama namun memiliki tingkah laku/tindakan yang berbeda. Hal ini bertujuan supaya kita dapat membuat sebuah fungsi yang lebih spesifik dan lebih mudah untuk dimengerti oleh programmer maupun pengguna yang menggunakan.

Hal ini tentunya dapat membantu kita dalam menangani proses yang mengharuskan kita untuk menggunakan nama method yang ada, namun memiliki tingkah laku yang berbeda. Sebagai contoh proses perhitungan luas pada **persegi** dengan **persegi panjang**. Dimana, kedua bidang tersebut merupakan bentuk dari segi empat yang memiliki tindakan berupa **luas** dan **keliling**. Hanya saja dalam proses pencarian yang dilakukan untuk kedua tersebut menggunakan cara yang berbeda. Silahkan perhatikan potongan kode berikut :

```
class segiEmpat:
    def __init__(self, sisi1, sisi2, sisi3, sisi4):
        self.sisi1 = sisi1
        self.sisi2 = sisi2
        self.sisi3 = sisi3
        self.sisi4 = sisi4
    def luas(self):
        return 0
    def keliling(self):
        return 1

class persegi(segiEmpat):
    def luas(self):
        return self.sisi1 * self.sisi1
    def keliling(self):
        return 4 * self.sisi1

class persegiPanjang(segiEmpat):
    def luas(self):
        return self.sisi1 * self.sisi2
    def keliling(self):
        return 2 * (self.sisi1 + self.sisi2)
```

Figure 28 Contoh Polymorphism (2)

Pada kelas tersebut terlihat jelas, bahwa ada terdapat 1 kelas induk berupa kelas segiEmpat dan 2 kelas keturunan berupa kelas persegi dan kelas persegiPanjang. Hanya saja pada kedua kelas keturunan memiliki method dengan yang sama, hanya saja proses yang ada kelas keturunan tersebut yang berbeda. Apakah hasil keluarannya akan sama atau berbeda?

Untuk tidak menjadi tanda tanya, mari kita jalankan kode tersebut :

```
>>> a = segiEmpat(10, 20, 30, 40)
>>> b = persegi(10, 20, 30, 40)
>>> c = persegiPanjang(10, 20, 30, 40)
>>> a.luas()
0
>>> a.keliling()
1
>>> b.luas()
100
>>> b.keliling()
40
>>> c.luas()
200
>>> c.keliling()
60
>>> █
```

Figure 29 Hasil Polymorphism (2)

Pada keluaran kode tersebut terlihat meskipun data yang kita input sama namun, tetap memiliki keluaran yang berbeda bergantung dari tindakan yang kita lakukan pada masing-masing kelas tersebut.

Namun, bagaimana cara kita membuat syarat untuk kelas-kelas tertentu, sehingga kelas tersebut dibuat dengan sesuai yang kita harapkan? Untuk menjawab hal ini, mari kita pelajari lesson berikutnya!

LESSON 4: MEMBUAT ABSTRAK CLASS

LESSON 4: OVERVIEW

Sebagai cara untuk membuat kriteria yang harus dibuat supaya kelas tersebut sesuai dengan yang kita harapkan, maka kita membutuhkan ABC (Abstract Base Class). ABC merupakan sebuah kelas yang dapat kita gunakan untuk membuat persyaratan terhadap kelas yang ingin kita buat. Pada kelas ABC ini, nanti akan ada sekumpulan method dan properties yang harus diterapkan pada kelas yang menggunakan kelas ini. Sebagai contoh perhatikan kode berikut :

```
import abc

class syaratSegiEmpat(metaclass=abc.ABCMeta):
    # Untuk syaratnya saya buat dalam bentuk komentar berikut
    # Untuk syarat yang pertama kelas tersebut harus ada 2 properties (atribut) sisi1 dan sisi2
    @abc.abstractproperty
    def sisi1(self):
        pass
    @abc.abstractproperty
    def sisi2(self):
        pass

    # Untuk syarat berikut harus memiliki fungsi berupa luas dan keliling
    @abc.abstractmethod
    def luas(self):
        pass
    @abc.abstractmethod
    def keliling(self):
        pass
```

Figure 30 Kelas Abstract (ABC)

Untuk menggunakan abstract class, terlebih dahulu kita harus memanggil module abc yang nantinya akan kita gunakan untuk membuat kriteria dari sebuah kelas. Pada kelas syaratSegiEmpat, memiliki kelas induk berupa `metaclass=abc.ABCMeta`, bagian ini tidak akan dijelaskan lebih lanjut. Karena, bagian ini tidak begitu sering digunakan untuk pelajaran-pelajaran berikutnya. Hanya bagi kalian yang ingin berlajar lebih lanjut, bisa mencari pada Google atau website pencarian lainnya.

Pada kelas tersebut, kita berfokus kepada bagian `@abc.abstractproperty` dan `@abc.abstractmethod` yang biasanya kita sebut sebagai *constructs*. Bagian ini merupakan bagian yang sangat penting untuk membuat method dan properties menjadi abstrak atau menjadi sebuah syarat dalam pembuatan sebuah kelas. Terus, gimana cara kita bisa menggunakannya? Perhatikan kode berikut :

```
from abstract import syaratSegiEmpat

class segiEmpat:
    def __init__(self, sisi1, sisi2):
        self.sisi1 = sisi1
        self.sisi2 = sisi2

class persegi(segiEmpat, syaratSegiEmpat):
    def luas(self):
        return self.sisi1 * self.sisi2

class persegiPanjang(segiEmpat, syaratSegiEmpat):
    def luas(self):
        return self.sisi1 * self.sisi2
    def keliling(self):
        return 2 * (self.sisi1 + self.sisi2)
```

Figure 31 Mengakses Kelas Abstract

NB : Pada konsep penerapan ini, kita sebenarnya bisa membuat 1 kelas turunan yang memiliki 2 kelas induk seperti pada contoh diatas. Kita hanya perlu menambahkan tanda koma dan dilanjutkan dengan nama kelas induk berikutnya. Sebagai contoh : `class kelasTurunan(kelasInduk1, kelasInduk2)`. Hal ini merupakan merupakan penerapan dari multi inheritance.

Dari kelas tersebut terlihat kita melakukan import modul yang ada pada file abstract yang telah saya buat sebelumnya, dan ada terdapat 1 kelas induk berupa `segiEmpat`, 1 kelas abstract berupa `syaratSegiEmpat` dan 2 kelas keturunan, yaitu `persegi` dan `persegiPanjang`. Kedua kelas turunan tersebut merupakan kelas turunan dari kelas `segiEmpat` dan kelas `syaratSegiEmpat`. Terus, gimana hasilnya? Apakah akan sama atau terjadi error (kesalahan dalam program)? Mari kita lihat hasilnya sebagai berikut :

```
>>> a = persegi(10, 20)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't instantiate abstract class persegi with abstract methods keliling, sisi1, sisi2
>>> b = persegiPanjang(10, 20)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't instantiate abstract class persegiPanjang with abstract methods sisi1, sisi2
>>> |
```

Figure 32 Hasil Keluaran Kelas Abstract (1)

Ternyata terjadi kesalahan dalam program ketika kita ingin melakukan inisialisasi ke dalam kelas tersebut. Mengapa demikian? Coba perhatikan kembali bagian kelas syaratSegiEmpat! Apa saja syarat yang kita buat sebelumnya? Ternyata harus ada 2 properties berupa sisi1 dan sisi2 serta harus ada 2 buah method berupa luas dan keliling. Nah, mari kita coba sesuaikan kembali, dan lihat hasilnya gimana. Untuk hasil perbaikan kelas tersebut, seperti tampilan berikut :

```
from abstract import syaratSegiEmpat

class segiEmpat:
    def __init__(self, sisi1, sisi2):
        self.sisi1 = sisi1
        self.sisi2 = sisi2

    class persegi(segiEmpat, syaratSegiEmpat):
        sisi1 = 0
        sisi2 = 0
        def luas(self):
            return self.sisi1 * self.sisi2
        def keliling(self):
            return 4 * self.sisi1

    class persegiPanjang(segiEmpat, syaratSegiEmpat):
        sisi1 = 0
        sisi2 = 0
        def luas(self):
            return self.sisi1 * self.sisi2
        def keliling(self):
            return 2 * (self.sisi1 + self.sisi2)
```

Figure 33 Mengakses Kelas Abstract (Perbaikan)

Dari kelas turunan telah kita buat sesuai dengan syaratnya. Nah, gimana kalau kita jalankan kembali? Apakah masih ada error? Silahkan perhatikan keluaran berikut :

```
>>> a = persegi(10, 10)
>>> b = persegiPanjang(10, 20)
>>> a.luas()
100
>>> a.keliling()
40
>>> b.luas()
200
>>> b.keliling()
60
>>> █
```

Figure 34 Hasil Keluaran Kelas Abstract (2)

Ternyata setelah kita jalankan kembali, sudah tidak terjadi error. Oleh karena itu, penting untuk kita pahami kembali bagaimana cara membuat dan tentunya hal ini akan sangat membantu bagi programmer dalam membuat kelas sesuai dengan yang kita harapkan.

Nah bagaimana sekarang, apakah sudah siap untuk menjawab studi kasus kita?

SOLUTION

Setelah kita membaca berbagai macam topik diatas, yuk kita kembali ke studi kasus kita diawal.

Setelah kita baca kembali studi kasus tersebut, ada hal atau syarat yang harus kita lakukan supaya bisa menjawab permasalahan tersebut. Hal ini tentunya dapat terselesaikan dengan menerapkan penerapan dari kelas abstract yang telah kita pelajari sebelumnya. Setelah itu, ada 1 buah method yang memiliki nama yang sama, hanya saja proses yang dilakukan berbeda-beda. Tentunya untuk mengatasi masalah tersebut kita bisa menerapkan prinsip dari polymorphism.

Nah setelah kita pelajari semuanya, mari kita jawab studi kasus tersebut!

INSTRUCTION

1. Terlebih dahulu mari kita buat dulu kelas abstract sesuai dengan syarat yang ada pada studi kasus, dimana syaratnya, yaitu :
 - Harus mempunyai 1 buah data berupa data buku, dalam bentuk nama buku.
 - Harus dapat mencetak buku tersebut supaya bisa melakukan pendataan.

```
import abc
```

```
class syaratPerpus(metaclass=abc.ABCMeta):  
    @abc.abstractproperty  
    def namaBuku(self):  
        pass  
  
    @abc.abstractmethod  
    def cetakData(self):  
        pass
```

2. Setelah kita membuat syaratnya, selanjutnya kita buat terlebih dahulu kelas induk perpus sebagai dasar dalam pembuat kelas untuk kelas keturunannya.

```
class perpus:  
    def __init__(self, jlhBuku):  
        self.jlhBuku = jlhBuku
```

3. Setelah kita membuat kelas induknya, selanjutnya kita membuat kelas untuk kelas keturunannya, yaitu perpusJawa dan perpusBatak. Namun, pada kelasnya jangan lupa juga kita tambahkan sesuai dengan studi kasus, berupa :
 - Daerah Jawa, harus bisa mencetak jenis gamelan yang ada.
 - Daerah Batak, harus bisa mencetak jenis ulos yang dibuat.

```
class perpusJawa(perpus, syaratPerpus):  
    namaBuku = ""  
    def __init__(self, namaBuku, jlhBuku):  
        super().__init__(jlhBuku)  
        self.namaBuku = namaBuku  
    def cetakData(self, jenisGamelan):
```

```

print("Pada Perpus Jawa")
print(f"Terdapat buku berjudul {self.namaBuku} dengan jumlah {self.jlhBuku} buku")
print(f"Dan terdapat jenis gamelan berupa {jenisGamelan}")

class perpusBatak(perpus, syaratPerpus):
    namaBuku = ""
    def __init__(self, namaBuku, jlhBuku):
        super().__init__(jlhBuku)
        self.namaBuku = namaBuku
    def cetakData(self, jenisUlos):
        print("Pada Perpus Batak")
        print(f"Terdapat buku berjudul {self.namaBuku} dengan jumlah {self.jlhBuku} buku")
        print(f"Dan terdapa ulos yang berasal dari {jenisUlos}")

```

4. Setelah semua selesai, yuk kita jalankan dan lihat gimana hasilnya :

```

>>> a = perpusJawa("Habis Gelap Terbi Terang", 10)
>>> b = perpusBatak("Pedoman Upacara Batak", 5)
>>> a.cetakData("Bonang")
Pada Perpus Jawa
Terdapat buku berjudul Habis Gelap Terbi Terang dengan jumlah 10 buku
Dan terdapat jenis gamelan berupa Bonang
>>> b.cetakData("Batak Toba")
Pada Perpus Batak
Terdapat buku berjudul Pedoman Upacara Batak dengan jumlah 5 buku
Dan terdapa ulos yang berasal dari Batak Toba
>>> []

```

Figure 35 Hasil Studi Kasus

EXERCISE

EXERCISE OBJECTIVES

Pada latihan berikut ini mahasiswa diharapkan dapat melakukan :

- Melakukan analisis masalah terhadap studi kasus yang diberikan.
- Mencari solusi atas masalah yang diberikan.
- Menerjemahkan masalah tersebut ke dalam bentuk kode pemrograman.

TASK 1:

Pada kelas yang ada di Mikroskil, ada dosen dan mahasiswa yang memiliki identitas yang berbeda-beda. Hal ini bertujuan supaya proses yang dilakukan jelas dan tidak bertumpang tindih. Namun, pada data mahasiswa dan dosen harus ada berupa nama dan nomor identitas. Dan untuk melakukan absensinya kita harus membedakan antara dosen dan mahasiswa. Dimana ketika dosen melakukan absensi maka muncul pesan berupa “Silahkan mulai pelajaran”. Sedangkan ketika mahasiswa melakukan absensi maka akan muncul pesan berupa “Terima kasih sudah hadir”.

Untuk menyelesaikan ini, harus ada 1 buah abstract class, 1 buah kelas induk dan 2 buat kelas turunan. Untuk penamaan kelas induk dan abstract class dibebaskan. Jadi jangan sampai dijumpain nama abastract class dan kelas induk yang sama.

TASK 2:

Pada sebuah private les, terdapat berbagai murid yang memiliki berbagai tingkatan yang berbeda diantaranya : SD, SMP dan SMA. Hal ini tentunya disesuaikan juga dengan biaya les yang diberikan, dimana biaya disesuaikan dengan data berikut :

Tingkatan	Biaya Les
SD	Rp. 500.000,00
SMP	Rp. 800.000,00
SMA	Rp. 1.200.000,00

Setelah itu, untuk data dari murid les, harus melengkapi data berupa Nama, dan NO HP orangtua yang digunakan. Hal ini bertujuan supaya ketika murid les sudah selesai dapat menghubungi orangtua masing-masing supaya bisa dijemput untuk pulang.

Untuk data tambahan yang dibutuhkan oleh private les tersebut disesuai juga berdasarkan tingkatannya. Dimana :

- Tingkat SD diperlukan data tambahan berupa Jenis Kelamin.
- Tingkat SMP diperlukan data tambahan berupa Umur.
- Tingkat SMA diperlukan data tambahan berupa jenis kelas (IPA atau IPS).

Untuk menyelesaikan ini, harus ada 1 buah abstract class, 1 buah kelas induk dan 3 buat kelas turunan. Untuk penamaan kelas induk dan abstract class dibebaskan. Jadi jangan sampai dijumpain nama abastract class dan kelas induk yang sama.

UNIT 3

EXCEPTION

UNIT OVERVIEW

Pada modul kali ini kita akan mempelajari cara yang digunakan untuk mengatasi kendala yang dihadapi oleh pengguna ketika inputan yang diberikan salah. Hal ini tentunya dapat membantu pengguna dan developer untuk lebih memperhatikan setiap kesalahan inputan yang diberikan.

UNIT OBJECTIVES

Adapun capaian yang akan kita dapatkan pada modul ini, yaitu :

- Mengenal kesalahan (error) yang terdapat pada program Python.
- Cara untuk menangani kesalahan tersebut.
- Membuat penanganan kesalahan sendiri.

UNIT CONTENTS

Lesson 1: Error pada Python	29-
32	
Lesson 2: Penanganan Error	32-
33	
Lesson 3: Custom Pesan Kesalahan.....	33-
35	

PRE LAB

Sebelum masuk ke dalam lab pertama ini, terlebih dahulu kita coba jawab beberapa pertanyaan ini berdasarkan hasil penjelasan pada modul teori!

QUESTION

1. Menurut kalian, apa yang dimaksud dengan polymorphism?
2. Jenis-jenis polymorphism apa saja yang kalian ketahui?
3. Beberapa tindakan berikut, manakah yang lebih tepat untuk ke bagian Polymorphism:
 - Tindakan berjalan.
 - Tindakan mencetak data.
 - Tindakan mencari buku.

CONTENT LESSON

CASE STUDY / PROJECT

Seperti yang kita ketahui bahwa perpustakaan “Home Learning is Best” telah membuka cabang pada beberapa daerah. Hanya saja pada proses pendataan yang dilakukan tidak sesuai dengan yang pada database pusat. Adapun pendataan yang seharusnya dilakukan, yaitu :

1. Pada kode buku hanya boleh terdiri dari huruf kapital dan angka saja.
2. Pada kode buku, harus memiliki 10 buah karakter.
3. Pada nama buku, hanya terdiri dari huruf saja.
4. Pada jumlah buku hanya boleh terdiri dari bilangan bulat saja.
5. Pada bagian nomor HP pengguna hanya diperbolehkan menggunakan karakter tambah “+” dan angka saja.
6. Pada bagian nomor HP pengguna juga diharuskan hanya memiliki panjang tidak kurang dari 8 karakter dan tidak lebih dari 15 karakter.

Pendataan tersebut merupakan pendataan yang dilakukan sama perpustakaan pusat. Tujuannya dilakukan seperti itu supaya data yang dimasukkan dapat disesuaikan dengan format yang ada di perpustakaan pusat.

Bagaimana dengan syarat? Tentunya sangatlah ribet. Tapi, ini harus dilakukan supaya proses yang pendataan dapat berjalan dengan lancar serta pada perpustakaan pusat dapat berjalan dengan lancar.

IDENTIFICATION CONCEPT OF PROBLEM / PROJECT

Setelah kita baca kembali studi kasus tersebut, tentunya ada hal-hal yang menjadi masalah, yaitu apabila proses tersebut terus berjalan, apakah sistem tetap dapat berjalan? Setelah data tersebut dimasukkan ke dalam database pusat, apakah tidak terjadi kesalahan?

Sebelum kita menjelaskan solusi dari permasalahan tersebut, mari kita simak dulu beberapa penjelasan untuk membantu kita dalam menyelesaikan permasalahan yang terdapat dalam studi kasus tersebut !

LESSON 1: ERROR PADA PYTHON

LESSON 1: OVERVIEW

Kesalahan atau Error pada Python dapat saja berbagai macam kesalahan yang ada. Hal ini tentunya dapat memaksa program untuk berhenti. Hal ini tentunya dapat membuat siapapun yang

menggunakan program yang kita buat menjadi stress. Terlebih lagi apabila program tersebut diharapkan untuk dapat digunakan untuk menyelesaikan masalah dengan cepat. Sebelum kita membuat cara untuk mencegah permasalahan atau kesalahan tersebut tidak terjadi, yuk kita pahami kesalahan-kesalahan yang ada pada Python !

1. ValueError

ValueError atau kesalahan pada nilai merupakan kesalahan yang sering terjadi. Dimana pada umumnya proses yang dilakukan sistem atau fungsi hanya berada pada rentang nilai tertentu, namun kita buat melebihi dari rentang yang ditentukan. Seperti potongan kode berikut :

```
>>> import math
>>> math.sqrt(-10)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: math domain error
```

Figure 36 Contoh ValueError (1)

Pada potongan kode tersebut terdapat pesan kesalahan berupa “ValueError: math domain error”. Ini merupakan kesalahan yang berada pada bagian ValueError yang disebabkan rentang nilai yang kita berikan melebihi batas yang ditetapkan. Dimana pada proses sqrt nilai yang diterima adalah mulai dari 0 sama dengan seterusnya.

ValueError ini juga bisa terjadi pada kesalahan jenis variabel yang kita gunakan. Seperti potongan kode berikut :

```
>>> a = int("Data")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'Data'
```

Figure 37 Contoh ValueError (2)

2. IndexError

IndexError atau kesalahan dalam pengaksesan index yang terdapat list atau sekumpulan data. Hal ini tentunya juga sering terjadi. Dimana kita ketahui bahwa index yang terdapat array (sekumpulan data) mulai dari 0 sampai dengan seterusnya. Sebagai contoh perhatikan potongan kode berikut :

```
>>> a = [1,2,3,4]
>>> a[4]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Figure 38 Contoh IndexError

Pada potongan kode diatas, terdapat kesalahan ketika kita ingin mengakses index array ke-4. Meskipun panjang dari array tersebut adalah 4, tapi kita ingat kembali bahwa index array dimulai dari 0. Jadi, seharusnya index yang bisa diterima berupa 0, 1, 2, 3. Sehingga, apabila kita memasukkan 4 akan memberikan error.

3. NameError

NameError atau kesalahan dalam penamaan variabel merupakan kesalahan yang terjadi ketika kita mengakses nama variabel yang sebelumnya belum kita deklarasikan sebelumnya atau belum kita kenalkan sama Python terlebih dahulu. Sebagai contoh perhatikan potongan kode berikut :

```
>>> aaaa = "Nama saya adalah aaaa"
>>> print(aaa)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'aaa' is not defined
```

Figure 39 Contoh NameError

Pada kode tersebut, sebenarnya kita sudah menuliskan dengan benar. Hanya saja kurang 1 karakter “a” pada bagian akhirnya. Sehingga, ketika kita mengetik kode tersebut tidak berhasil dan memunculkan error (kesalahan). Hal ini tentunya sering terjadi mengingatkan manusia ketika mengetikan sesuai tidak terhindari dari “typo” atau kesalahan pada penulisan.

4. TypeError

TypeError atau kesalahan pada jenis variabel yang kita gunakan merupakan kesalahan yang terjadi apabila kita mendeklarasikan jenis variabel tidak sesuai dengan ketentuan yang ada. Sebagai contoh perhatikan potongan kode berikut :

```
>>> a = 10
>>> b = "dua"
>>> c = a + b
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Figure 40 Contoh TypeError

Pada kode terlihat ketika kita ingin menjumlah dua jenis data yang berbeda, yaitu antara teks dengan angka akan mengakibat menjadi error. Hal ini tentunya harus diperhatikan ketika membuat sebuah program pada Python.

5. SyntaxError

SyntaxError atau kesalahan dalam penulisan merupakan kesalahan yang terjadi ketika kita membuat kesalahan dalam penulisan kode yang dibuat. Kesalahan dalam penulisan ini juga dapat berupa kelupaan menutup, atau menasukkan nilai ke nilai lainnya dan lain sebagainya. Sebagai contoh perhatikan potongan kode berikut :

```
>>> 10 == 10
True
>>> 10 = 10
  File "<stdin>", line 1
SyntaxError: can't assign to literal
```

Figure 41 Contoh SyntaxError

Pada kode tersebut terlihat pada bagian awal ketika kita membandingkan 2 buah bilangan tersebut dengan tanda “==” (2 kali sama dengan) tidak ada masalah. Hanya saja pada kode berikutnya muncul error. Hal ini dikarenakan, dianggap kita ingin memasukkan nilai 10 ke dalam 10. Hal ini tentunya tidak bisa. Karena, kita hanya bisa memasukkan nilai ke dalam variabel bukan nilai itu kembali.

Itu merupakan beberapa kesalahan yang sering terjadi pada kode yang kita buat. Namun, kesalahan tersebut tidak hanya itu saja, ada beberapa lagi kesalahan yang lain. Untuk itu perlu kalian explorer lebih lanjut dan pahami kesalahannya supaya bisa jelas terhadap prosesnya.

Ketika Python menghadapi kode program tersebut tentunya akan memaksa Python untuk berhenti, sehingga kita tidak melanjutkan kepada proses atau baris kode berikutnya. Hal ini tetunya sangat merugikan untuk kita sebagai developer dan pengguna. Namun, apakah ada cara yang bisa kita lakukan untuk tetap menjalankan program meskipun menjumpai kode tersebut? Jawabannya adalah “ada”. Yuk kita pelajari pelajaran selanjutnya !

LESSON 2: PENANGANAN ERROR

LESSON 2: OVERVIEW

Seperti yang telah kita ketahui sebelumnya ketika ada kesalahan atau error pada kode program yang kita buat, Python akan memaksa program yang kita buat untuk berhenti. Sebagai contoh perhatikan potongan kode berikut :

```
def konversi(angka1):
    return int(angka1)

a = konversi(1)
print(a)
b = konversi("dua")
print(b)
c = konversi(3)
print(c)
```

Figure 42 Potongan Kode yang Salah

Pada kode tersebut tidak terlihat adanya kesahan dan tidak ada penanda kesalahan yang ada. Namun, kalau kita jalankan kode tersebut, maka akan muncul tampilan berikut :

```
1
Traceback (most recent call last):
  File "ContohPraktek.py", line 6, in <module>
    b = konversi("dua")
  File "ContohPraktek.py", line 2, in konversi
    return int(angka1)
ValueError: invalid literal for int() with base 10: 'dua'
```

Figure 43 Hasil Kode Kesalahan

Dari kode tersebut terlihat proses awal pada variabel “a” yang mengkonversi nilai 1 ke dalam integer dapat diberikan, sedangkan untuk proses berikutnya pada variabel “b” yang mengkonversi nilai “dua” ke dalam integer tidak dapat dilakukan dan menghasilkan ValueError. Hal ini dikarenakan memang kita tidak bisa mengkonversi teks ke dalam angka, sehingga memunculkan error. Adapun akibat yang

terjadi adalah, pada variabel “c” yang mengkonversi nilai 3 ke dalam interger meskipun benar, hanya saja dikarenakan proses sebelumnya sudah salah mengakibat program langsung keluar. Tentunya hal ini merupakan kerugian bagi developer maupun pengguna.

Terus, bagaimana cara yang harus kita lakukan supaya kode berikutnya tetap berjalan meskipun terdapat kesalahan pada kode? Salah satu caranya adalah menggunakan **try dan except**. Yuk kita perhatikan perubahan pada bagian fungsi kode sebelumnya :

```
def konversi(angka1):
    try:
        return int(angka1)
    except ValueError as e:
        print(str(e))
    return -1
```

Figure 44 Penanganan Error

Setelah itu, baru kita jalankan kembali kode yang tadi, dan hasil tampilannya seperti berikut :

```
1
invalid literal for int() with base 10: 'dua'
-1
3
```

Figure 45 Hasil Penanganan Error

Dari hasil kode tersebut, terlihat bahwa program telah berhasil berjalan dan tidak ada kesalahan serta baris kode berikutnya dapat dieksekusi dengan baik. Hal ini tentunya dapat menyelesaikan permasalahan kita untuk kesalahan yang terjadi pada kode program yang kita buat.

Namun, bagaimana apabila kita ingin membuat penulisan kesalahan dengan versi kita sendiri, supaya pengguna lebih paham ketimbang harus memahami isi pesan kesalahan yang ada. Untuk mengatasi masalah tersebut, mari kita pelajari pelajaran berikutnya !

LESSON 3: CUSTOM PESAN KESALAHAN

LESSON 3: OVERVIEW

Pada proses sebelumnya kita telah belajar mengenai penanganan kesalahan yang ada. Hanya saja pesan kesalahan yang ada sebelumnya masih disesuaikan pesan kesalahan yang ada pada sistem. Namun, gimana cara kita buat pesan kesalahan versi kita sendiri? Caranya masih sama dengan sebelumnya dengan menggunakan **try dan except**, hanya saja kali ini kita mencetak pesan kesalahan yang disesuaikan dengan pesan yang kita buat sendiri.

Seperti pada contoh sebelumnya, saya sedikit mengubah pada bagian except seperti tampilan berikut:

```
def konversi(angka1):
    try:
        return int(angka1)
    except:
        print("Kesalahan pada inputan yang diberikan ... !!!")
        return -1
```

Figure 46 Penyesuaian Pesan Kesalahan (1)

Terus, gimana hasilnya? Apakah sama dengan sebelumnya? Kita lihat saja tampilan hasil berikut :

```
1
Kesalahan pada inputan yang diberikan ... !!!
-1
3
```

Figure 47 Hasil Penyesuaian Pesan Kesalahan

Dari hasil tersebut terlihat pesan kesalahan yang kita sebelumnya telah berhasil muncul. Hal ini membuat kita menjadi lebih mudah memahami terkait pesan kesalahan yang muncul.

Namun, bagaimana kalau kita ingin mengubah “pesan” kesalahan yang terdapat pada sistem dan disesuaikan dengan pesan yang ingin kita buat. Seperti contoh kode berikut :

```
def konversi(angka1):
    if not isinstance(angka1, int):
        raise ValueError("Kamu salah memasukkan data")
    return int(angka1)
```

Figure 48 Penyesuaian Pesan Kesalahan (2)

Pada proses tersebut kita menggunakan perintah baru berupa raise, yang langsung menghasilkan error atau memaksa program untuk berhenti namun dengan pesan kesalahan yang kita buat. Untuk hasil pada kode tersebut akan menghasilkan tampilan berikut :

```
1
Traceback (most recent call last):
  File "ContohPraktek.py", line 8, in <module>
    b = konversi("dua")
  File "ContohPraktek.py", line 3, in konversi
    raise ValueError("Kamu salah memasukkan data")
ValueError: Kamu salah memasukkan data
```

Figure 49 Hasil Penyesuaian Pesan Kesalahan (2)

Pada kode tersebut terlihat pesan kesalahan yang ada sudah disesuaikan dengan pesan kesalahan yang kita buat sebelumnya. Hanya saja pesan kesalahan tersebut dapat memaksa program untuk berhenti dan tidak bisa melanjutkan ke kode berikutnya. Bagaimana cara kita mengatasi kesalahan ini, sama dengan cara sebelumnya menggunakan **try** dan **except** seperti kode berikut :

```
def konversi(angka1):
    try:
        if not isinstance(angka1, int):
            raise ValueError("Kamu salah memasukkan data")
    except ValueError as e:
        print(str(e))
        return -1
    return int(angka1)
```

Figure 50 Contoh Kode Raise dengan Try

Pada kode tersebut kita telah mengubah seperti sebelumnya dan untuk hasilnya seperti tampilan berikut :

```
1
Kamu salah memasukkan data
-1
3
```

Figure 51 Hasil Raise dengan Try

Dari hasil tersebut terlihat prosesnya bisa berjalan dan meskipun ada kesalahan tetap melanjutkan kepada baris kode berikutnya.

Setelah kita belajar, yuk kita jawab studi kasus yang ada !

SOLUTION

Setelah kita belajar pada pelajaran-pelajaran yang ada, yuk kita kembali mencari jawaban untuk studi kasus kita.

Pada studi kasus kita, terdapat permasalahan pada kesalahan yang ada penulisan kode buku, panjang karakter serta kesesuaian karakter yang ada. Hal ini semua dapat kita selesaikan dengan menggunakan Try, Except serta Raise yang telah kita pelajarin sebelumnya.

Nah supaya lebih jelas, yuk kita bahas 1 per 1.

INSTRUCTION

1. Tahap awal kita buat terlebih dahulu inputan ke dalam program kita seperti tampilan kode berikut :

```
class BUKU:
    def inputBuku(self, kode, nama, noHP, jlhBuku):
```

```

if(self.cekKode(kode) and self.cekNama(nama) and self.cekNoHp(noHP) and
self.cekStok(jlhBuku)):
    print("Data Buku Berhasil dimasukkan")

```

2. Tahap berikutnya kita lakukan pengecekan terhadap kode yang dimasukkan :

```

def cekKode(self, kode):
    huruf = "ABCDEFGHIJKLMNPQRSTUVWXYZ"
    angka = "0123456789"
    jlhHuruf = 0
    for i in range(len(kode)):
        for j in range(len(huruf)):
            if(kode[i] == huruf[j]):
                jlhHuruf += 1
                break

    jlhAngka = 0
    for i in range(len(kode)):
        for j in range(len(angka)):
            if(kode[i] == angka[j]):
                jlhAngka += 1
                break

    try:
        if(jlhAngka + jlhHuruf != len(kode)):
            raise ValueError("Kode Hanya Boleh terdiri dari huruf dan angka saja")
        if(len(kode) != 10):
            raise ValueError("Panjang kode harus = 10")
    except ValueError as e:
        print(str(e))
        return False
    return True

```

3. Tahap berikutnya kita lakukan pengecekan terhadap nama yang dimasukkan :

```

def cekNama(self, nama):
    huruf = "ABCDEFGHIJKLMNPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz "
    jlhHuruf = 0
    for i in range(len(nama)):
        for j in range(len(huruf)):
            if(nama[i] == huruf[j]):
                jlhHuruf += 1
                break

    try:
        if(jlhHuruf != len(nama)):
            raise ValueError("Hanya boleh terdiri dari huruf saja")
    except ValueError as e:
        print(str(e))
        return False
    return True

```

4. Tahap berikutnya kita lakukan pengecekan terhadap stok yang dimasukkan :

```
def cekStok(self, stok):
    try:
        if(not isinstance(stok, int)):
            raise ValueError("Harus menggunakan angka saja")
    except ValueError as e:
        print(str(e))
        return False
    return True
```

5. Tahap berikutnya kita lakukan pengecekan terhadap no HP yang dimasukkan :

```
def cekNoHp(self, noHP):
    angka = "0123456789+"
    jlhAngka = 0
    for i in range(len(noHP)):
        for j in range(len(angka)):
            if(noHP[i] == angka[j]):
                jlhAngka += 1
                break
    try:
        if(jlhAngka != len(noHP)):
            raise ValueError("Hanya boleh terdiri dari huruf saja")
        if(len(noHP) < 8 or len(noHP) > 15):
            raise ValueError("Panjang No HP berupa dari 8 sampai 15")
    except ValueError as e:
        print(str(e))
        return False
    return True
```

6. Setelah semua berhasil sesuai dengan yang disarankan pada studi kasus kita lihat hasil program berjalan gimana :

```
>>> b = BUKU()
>>> exit()
PS D:\(2021 2022) Ganjil\OOP with Python\Contoh\Pert-04> python -i JawabanContohPraktek.py
>>> b = BUKU()
>>> b.inputBuku("M123456789", "Pemrograman OOP", 10)
Data Buku Berhasil dimasukkan
>>> b.inputBuku("M123456789+", "Pemrograman OOP", 10)
Kode Hanya Boleh terdiri dari huruf dan angka saja
>>> b.inputBuku("M123456789", "Pemrograman OOP 1", 10)
Hanya boleh terdiri dari huruf saja
>>> b.inputBuku("M123456789", "Pemrograman OOP", "dua")
Harus menggunakan angka saja
```

Figure 52 Hasil Studi Kasus

EXERCISE

EXERCISE OBJECTIVES

Pada latihan berikut ini mahasiswa diharapkan dapat melakukan :

- Melakukan analisis masalah terhadap studi kasus yang diberikan.
- Mencari solusi atas masalah yang diberikan.
- Menerjemahkan masalah tersebut ke dalam bentuk kode pemrograman.

TASK 1:

Pada sebuah kelas di Kampus Mikroskil terdapat mahasiswa dan dosen pada kelas tersebut. Mahasiswa tersebut tentunya memiliki data berupa NIM, Nama dan No HP. Sedangkan dosen memiliki data berupa NIP, Nama, no HP. Adapun ketentuan yang harus dicek, yaitu :

1. Untuk kode hanya terdiri dari angka dan harus memiliki panjang 9 karakter.
2. Untuk nama hanya terdiri dari spasi dan huruf saja.
3. Untuk no HP hanya terdiri dari angka dan tanda tambah (“+”) serta untuk panjang berada pada rentang 8 sampai dengan 15 angka saja.

Yuk kita buat pengecekannya ... !!!

Untuk teknik pengecekan dan pesan kesalahan tidak ada batasan. Silahkan buat kreasi sendiri.

TASK 2:

Pada sebuah private les terdapat sejumlah murid yang diajarkan. Murid-murid tersebut, memiliki Nama, jenis kelamin, kelas, dan jenis les yang berikan. Hal ini tentunya tidak terlepas dari pada biaya les yang ditawarkan. Adapun daftar biaya les yang ditawarkan berdasarkan tingkat pendidikan yang diajarkan, yaitu :

Tingkatan	Biaya Les
TK	Rp. 300.000,00
SD	Rp. 500.000,00
SMP	Rp. 700.000,00
SMA	Rp. 1.000.000,00

Untuk jam pembelajaran yang diberikan juga didasarkan dari tingkatan yang diajarkan :

Tingkatan	Jam Pengajaran
TK	2 jam
SD	2 jam
SMP	1 jam
SMA	1 jam

Setelah itu, untuk setiap murid yang berhasil mendaftar diberikan sebuah kartu yang dicetak. Dan dikartu tersebut terdapat data berupa : Nama, Tingkatan, Jam Pengajaran dan Biaya Les.

Dapatkan Anda membantu private les tersebut?

Kita kembali melakukan proses ini. Hanya saja untuk pemasukkan data yang diberikan pastikan kembali. Untuk syarat serta pesan kesalahan disesuaikan kembali dengan kalian sendiri.

UNIT 4

THE ITERATOR PATTERN (1)

UNIT OVERVIEW

Pada modul kali ini kita akan membahas mengenai penjelasan terhadap pola-pola yang berulang. Hal ini tentunya menjadi sangat penting, karena apabila pola berulang tersebut tidak kita tangani dengan baik akan mengakibatkan proses yang berulang dan dapat mengakibatkan proses yang tidak penting. Oleh karena itu, penting untuk dipahami kira-kira pola yang bagaimana yang bisa disebut sebagai pola berulang dan gimana cara mengatasinya.

UNIT OBJECTIVES

Adapun capaian yang akan kita dapatkan pada modul ini, yaitu :

- Cara pembuatan pola berulang dengan for.
- Cara pembuatan pola berulang dengan iter.

UNIT CONTENTS

Lesson 1: Perulangan dengan For.....	40-
42	
Lesson 2: Perulangan dengan Iter	43-
45	

PRE LAB

Sebelum masuk ke dalam lab pertama ini, terlebih dahulu kita coba jawab beberapa pertanyaan ini berdasarkan hasil penjelasan pada modul teori!

QUESTION

1. Menurut kalian pentingnya kita melakukan exception adalah untuk?
2. Apa bedanya statement raise dengan statement try dan except?
3. Kira-kira ketika kita membuat data berupa nama pelanggan, batas-batas atau syarat-syarat apa saja yang harus kita penuhi?

CONTENT LESSON

CASE STUDY / PROJECT

Seperti yang telah kita ketahui, data yang ada perpustakaan “Home Learning is Best” menjadi sangat banyak. Hal ini mengakibatkan ketika pencarian yang dilakukan oleh staf menjadi sangat lama dan membuat kinerja menjadi tidak efisien. Oleh karena itu, perusahaan tersebut ingin membuat cara untuk melakukan pencarian data yang sangat banyak tersebut. Setelah data tersebut dicari, nantikan akan dikelompokkan berdasarkan jenis buku yang digunakan. Hal ini bertujuan supaya ketika pencarian yang dilakukan selanjutnya dapat tercapai dan tidak membuang waktu yang banyak.

Yuk mari bantu kembali perpustakaan kita yang tercinta ini!

IDENTIFICATION CONCEPT OF PROBLEM / PROJECT

Setelah kita membaca kembali studi kasus tersebut ada yang harus kita perhatikan mengenai cara yang harus kita lakukan untuk melakukan pencarian. Apakah hal tersebut dapat dilakukan sekali tahapan saja sudah cukup atau perlu dilakukan secara berulang? Setelah kita mendapatkan datanya apa yang harus dilakukan? Berdasarkan studi kasus tersebut perlu dilakukan pengelompokan. Bagaimana cara pengelompokan yang baik?

Sebelum kita menjelaskan solusi dari permasalahan tersebut, mari kita simak dulu beberapa penjelasan untuk membantu kita dalam menyelesaikan permasalahan yang terdapat dalam studi kasus tersebut !

LESSON 1: PERULANGAN DENGAN FOR

LESSON 1: OVERVIEW

Sebelumnya pada semester 1 kita pernah belajar melakukan perulangan di dalam program Python. Hal ini bertujuan untuk melakukan tindakan yang berulang tanpa harus menghabiskan baris yang banyak dan tetap sesuai dengan yang diharapkan.

Sebagai contoh, mungkin kalian pernah mendengar murid yang disuruh sama gurunya untuk menulis dipapan tulis dengan kata berupa “Saya tidak akan berbohong lagi ... !!!”. Kata tersebut tentunya bukan sekali ditulis tetapi bisa dilakukan beberapa kali sesuai dengan tingkat kesalahan yang dilakukan oleh murid tersebut (Jangan dicontoh tindakan berbohongnya ya ... !!!).

Seperti contoh pada tampilan program berikut :

```
1 print("Saya tidak akan berbohong lagi ... !!!")
2 print("Saya tidak akan berbohong lagi ... !!!")
3 print("Saya tidak akan berbohong lagi ... !!!")
4 print("Saya tidak akan berbohong lagi ... !!!")
5 print("Saya tidak akan berbohong lagi ... !!!")
6 print("Saya tidak akan berbohong lagi ... !!!")
7 print("Saya tidak akan berbohong lagi ... !!!")
8 print("Saya tidak akan berbohong lagi ... !!!")
9 print("Saya tidak akan berbohong lagi ... !!!")
10 print("Saya tidak akan berbohong lagi ... !!!")
```

Figure 53 Contoh Perulangan (1)

Dari hasil program tersebut akan menghasilkan 10 baris dengan kata berupa “Saya tidak akan berbohong lagi ... !!!”. Mungkin ketika kalian lakukan hal tersebut merupakan hal yang biasa saja. Karena, hal tersebut dapat terselesaikan hanya menggunakan copy dan paste sebanyak 9 kali saja (karena yang pertama tidak perlu dilakukan copy paste). Tapi bayangkan, bagaimana kalau yang disuruh adalah 1000 baris? Apakah copy paste bisa mengatasinya? Jawabannya adalah bisa, hanya saja mau sampai kapan? Oleh karena itu, kita bisa memanfaatkan perintah for, yang digunakan untuk melakukan perulangan. Seperti contoh tampilan kode berikut :

```
for i in range(1,1001):
    print("Saya tidak akan berbohong lagi ... !!!")
```

Figure 54 Contoh Perulangan (2)

NB : Kenapa 1001? Karena, pada perulangan hanya dilakukan sampai dengan $n - 1$ atau $1001 - 1 = 1000$. Sehingga, kita perlu menggunakan 1001 untuk membuat 1000 baris kode perintah untuk mencetak.

Dengan adanya 2 baris tersebut, kita bisa mencetak 1000 baris. Otomatis kita bisa menyelesaikan masalah pada murid tersebut. Hanya saja ini hanya sebagai contoh, bukan untuk ditiru. Yang ingin dijelaskan pada contoh ini adalah kita bisa menyederhanakan perintah tersebut menjadi lebih baik. Hal ini juga bisa kita lakukan untuk pengaksesan data yang ada di dalam array (sekumpulan data) seperti contoh berikut :

```
items = ["penghapus", "pulpen", "pensil", "buku", "penggaris", "tas sekolah"]
for i in range(len(items)):
    print(f"Data ke-{i+1} = {items[i]}")
```

Figure 55 Pengaksesan Index Array For (1)

Dari kode tersebut akan memunculkan hasil berikut :

```
Data ke-1 = penghapus  
Data ke-2 = pulpen  
Data ke-3 = pensil  
Data ke-4 = buku  
Data ke-5 = penggaris  
Data ke-6 = tas sekolah
```

Figure 56 Hasil Pengaksesan Index Array For (1)

Dari hasil tersebut ternyata for bukan hanya sebagai mencetak saja, tapi bisa juga kita gunakan untuk mengakses data yang ada di dalam array. Hal ini tentunya dapat membantu kita apabila data tersebut terdapat di dalam sekumpulan data yang banyak, yang mengharus kita untuk melakukan pencarian 1 per 1. Namun, ada cara lain yang juga bisa kita gunakan untuk mengakses data tersebut dengan cara seperti kode berikut :

```
items = ["penghapus", "pulpen", "pensil", "buku", "penggaris", "tas sekolah"]  
for item in items:  
    print(f"Data = {item}")
```

Figure 57 Pengakses Array Index For (2)

Dalam kode tersebut akan menghasilkan tampilan berikut :

```
Data = penghapus  
Data = pulpen  
Data = pensil  
Data = buku  
Data = penggaris  
Data = tas sekolah
```

Figure 58 Hasil Pengaksesan Index Array For (2)

Pada hasil tersebut terlihat, bahwa cara ini juga bisa kita gunakan untuk melakukan pencetakan data tanpa harus data keberapa data tersebut. Jadi, seperti kita ingin memastikan apakah orang A ada di dalam kelas tersebut atau tidak. Hal ini tentunya dapat membuat segala sesuatu menjadi lebih jelas dan tidak perlu dilakukan secara acak.

Namun, untuk melakukan perulangan bukan hanya menggunakan for saja, kita juga bisa menggunakan iter. Ingin tahu bagaimana caranya? Yuk kita pelajari pada pelajaran berikutnya ... !!!

LESSON 2: PERULANGAN DENGAN ITER

LESSON 2: OVERVIEW

Kita telah mengenal teknik untuk perulangan pada pelajaran sebelumnya adalah dengan menggunakan for. Namun, untuk pengaksesan indeks di dalam array ternyata ada 1 lagi satu lagi caranya, yaitu dengan menggunakan fungsi iter. Hal ini mungkin merupakan bagian yang sangat baru untuk kalian semuanya. Namun, apabila kalian telah mengetahui cara yang dilakukan di dalam for item in items, mungkin kalian sudah dapat mengetahui gimana caranya bekerja.

Secara gambaran besar dapat terlihat pada tampilan grafik berikut :

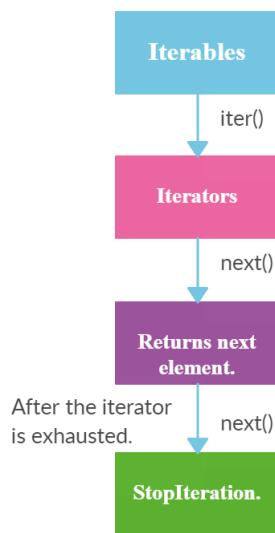


Figure 59 Tahapan Iter

Gambar mungkin telah kalian lihat pada pertemuan teori. Adapun langkah-langkah yang dilakukan proses perulangan dengan menggunakan iter dilakukan dengan 3 buah fungsi, yaitu : iter(), next() dan done(). Ketiga fungsi ini memiliki fungsinya masing-masing, yaitu :

1. Iter() merupakan fungsi yang bisa kita gunakan untuk membuat sebuah objek perulangan menjadi perulangan untuk bisa dilakukan proses secara berulang. Sebagai contoh array, list, dict dan beberapa objek lainnya.
2. Next() merupakan fungsi yang bisa kita gunakan untuk membuat penambahan (increment) ke dalam sekumpulan data tersebut. Hal ini tentunya harus dilakukan supaya data tersebut dapat sampai ke ujung dan dapat berakhir.
3. Done() merupakan fungsi yang bisa kita gunakan untuk mengakhiri perulangan. Meskipun fungsi ini sangat jarang sekali terlihat pada proses perulangan yang dilakukan pada iter, dan proses berulangan juga dapat dilakukan meskipun tanpa adanya fungsi ini. Hanya saja ketika kita ingin mengakhiri proses perulangan, adanya kita menggunakan fungsi untuk mengakhiri perulangan tersebut.

Setelah kita mengetahui fungsi dari 3 fungsi tersebut, mari kita kembali ke array kita sebelumnya. Bagaimana cara penerapannya yang harus kita lakukan? Silahkan perhatikan kode berikut :

```
items = ["penghapus", "pulpen", "pensil", "buku", "penggaris", "tas sekolah"]
datas = iter(items) #Inisialisasi iter
print(f"Data = {next(datas)}") #Mengakses data pertama
print(f"Data = {next(datas)}") #Mengakses data kedua
print(f"Data = {next(datas)}") #Mengakses data ketiga
print(f"Data = {next(datas)}") #Mengakses data keempat
print(f"Data = {next(datas)}") #Mengakses data kelima
print(f"Data = {next(datas)}") #Mengakses data keenam
print(f"Data = {next(datas)}") #Mengakses data ketujuh ?????
```

Figure 60 Pengaksesan Indeks Array Iter (1)

Seperti yang kita lihat sangatlah simple dan tidak perlu melakukan dengan pengaksesan indeks juga. Namun, gimana hasilnya? Kita lihat pada tampilan berikut :

```
Data = penghapus
Data = pulpen
Data = pensil
Data = buku
Data = penggaris
Data = tas sekolah
Traceback (most recent call last):
  File "contoh1.py", line 34, in <module>
    print(f"Data = {next(datas)}") #Mengakses data ketujuh ??????
StopIteration
```

Figure 61 Hasil Pengaksesan Indeks Array Iter (1)

Dari hasil tersebut terlihat hasilnya tidak jauh beda ketimbang menggunakan for yang telah kita lakukan sebelumnya. Hanya saja pada baris akhir ada error dan langsung menghasilkan StopIteration. Hal ini dikarenakan data yang ada di dalam array hanya terdapat 6, sedangkan kita mencetak sampai 7 kali. Hal ini tentunya dapat mengakibatkan kesalahan yang ada pada proses tersebut. Terus, gimana cara menangani masalah ini? Kita bisa menggunakan try except yang telah kita pelajari sebelumnya. Namun, bagaimana cara kita supaya kita tidak perlu menjalankan 1 per 1 datanya, kita bisa memanfaatkan while untuk melakukan perulangan. Hasilnya akan seperti tampilan kode berikut :

```
items = ["penghapus", "pulpen", "pensil", "buku", "penggaris", "tas sekolah"]
datas = iter(items) #Inisialisasi iter

while True:
    try:
        print(f"Data = {next(datas)}") #Mengakses array
    except StopIteration:
        break
```

Figure 62 Pengaksesan Indeks Array Iter (2)

Dari kode tersebut tentunya telihat lebih beda dan tentunya dapat menghemat baris karena kita tidak perlu mencetak sampai dengan jumlah data yang ada. Terus, gimana hasil? Apakah sama saja? Kita lihat pada tampilan dibawah ini :

```
Data = penghapus  
Data = pulpen  
Data = pensil  
Data = buku  
Data = penggaris  
Data = tas sekolah
```

Figure 63 Hasil Pengaksesan Indeks Array Iter (2)

Gimana? Sama dengan menggunakan for bukan. Mungkin kalian berpikir agak sedikit rumit, hanya saja dengan menggunakan perintah ini kita bisa lebih memahami proses yang terjadi dan pada saat akhir kita bisa pencetakan data atau kata.

Gimana? Mudah bukan. Yuk kita bahas solusi dari pemasalahan studi kasus kita ... !!!

SOLUTION

Setelah kita baca kembali studi kasus yang ada diatas, kita bisa memberikan solusi yang ada berupa melakukan perulangan. Karena, apabila datanya lebih dari 1, tentunya sangat efisien dilakukan perulangan. Tanpa perulangan tentunya akan sangat lama. Terus, teknik perulangan mana yang akan kita gunakan? Tentunya saya akan lebih memilih menggunakan iter. Supaya jelas prosesnya dan ketika tidak ditemukan datanya, kita bisa mencetak hasilnya tidak ditemukan tanpa perlu menggunakan percabangan tambahan.

Nah supaya lebih jelas, yuk kita bahas 1 per 1.

INSTRUCTION

1. Supaya prosesnya tidak terlalu panjang, saya buat saja data kita dalam bentuk array yang dimana isinya sudah ada nama beberapa buku yang ada berupa :

```
buku = []  
buku += ["Pemrograman dengan Python"]  
buku += ["Struktur data"]  
buku += ["Data Mining"]  
buku += ["Visualisasi Data"]  
buku += ["Pengolahan Citra"]  
buku += ["Kecerdasan Buatan"]  
buku += ["Pemrograman Web"]
```

2. Setelah kita buat datanya, mari kita buat fungsi pencarian dengan menggunakan iter. Gimana caranya? Perhatikan kode berikut :

```
#Melakukan pencarian buku dalam array bukus  
def cariBuku(bukus, buku):  
    items = iter(bukus)
```

```

i = 0
while True:
    try:
        if(next(items) == buku):
            print(f"Buku \'{buku}\' ditemukan pada tumpukan ke-{i+1} ... !!!")
            break
        i += 1
    except StopIteration:
        print(f"Buku \'{buku}\' tidak ditemukan ... !!!")
        break

```

3. Untuk melakukan pencarinya kita hanya melakukan perintah berikut :

```

cariBuku(buku, "Database")
cariBuku(buku, "Struktur data")
cariBuku(buku, "Pemrograman C")

```

4. Setelah proses tersebut dijalankan, maka akan muncul tampilan berikut :

```

Buku "Database" tidak ditemukan ... !!!
Buku "Struktur data" ditemukan pada tumpukan ke-2 ... !!!
Buku "Pemrograman C" tidak ditemukan ... !!!

```

Figure 64 Hasil Studi Kasus

5. Terus gimana cara kita bisa melakukan pengelompokan? Untuk proses ini kita akan bahas pada pertemuan berikutnya. Jadi, jangan lupa ditunggu untuk pertemuan berikutnya.

EXERCISE

EXERCISE OBJECTIVES

Pada latihan berikut ini mahasiswa diharapkan dapat melakukan :

- Melakukan analisis masalah terhadap studi kasus yang diberikan.
- Mencari solusi atas masalah yang diberikan.
- Menerjemahkan masalah tersebut ke dalam bentuk kode pemrograman.

TASK 1:

Pada kampus Mikroskil tentunya sangat banyak sekali mahasiswa dalam 1 kelas. Hal ini tentunya membuat dosen untuk mengetahui siapa saja yang ada di dalam kelas tersebut. Dimana kita ketahui bahwa data mahasiswa tersebut terdiri dari NIM, Nama dan NoHP. Dapatkah kalian membantu dosen ini untuk melakukan absensi untuk kelasnya?

Ketentuan yang dilakukan pada task-01 ini adalah :

- a. Semua proses dilakukan di dalam kelas.
- b. Nama kelas tersebut diharuskan menggunakan nama kalian.
- c. Untuk tindakan yang dilakukan dikelas tersebut terdapat 2, yaitu pemasukan data dan absensi.
- d. Untuk jenis perulangan yang digunakan pada Task-01 ini adalah dengan menggunakan for.
- e. Untuk contoh data yang digunakan untuk melakukan pencarian ada 5 data. (Silahkan pastikan tidak ada data yang sama)

- f. Untuk hasil pencarian tampilkan :
 - o 1 data yang ada di dalam kelas Mikroskil.
 - o 2 data yang tidak ada di dalam kelas Mikroskil.
- g. Untuk pesan keluaran bisa disesuaikan dengan kebutuhan kalian sendiri.
- h. Dikarenakan ada contoh data, maka perlu ada tampilan hasil proses pencarian data (data ditemukan atau tidak) dalam bentuk screenshot.

TASK 2:

Pada sebuah website E-Commerce terdapat berbagai jenis barang yang dijual, diantaranya hand sanitizer, ban mobil, buah-buahan serta botol minuman bayi. Barang tersebut memiliki atribut berupa kode, nama dan jumlah barang. Tentunya barang yang dijual tersebut tidak lebih dari 1 barang sehingga penjual harus tahu kira-kira barang apa saja yang mau habis. Hal ini tentunya perlu dilakukan pencarian dari data yang ada. Dapatkan kalian membantu pedagang tersebut?

Ketentuan yang dilakukan pada task-02 ini adalah :

- a. Semua proses dilakukan di dalam kelas.
- b. Nama kelas tersebut diharuskan menggunakan nama kalian.
- c. Untuk tindakan yang dilakukan dikelas tersebut terdapat 2, yaitu pemasukan data dan pencarian.
- d. Untuk jenis perulangan yang digunakan pada Task-02 ini adalah dengan menggunakan iter.
- e. Untuk contoh data yang digunakan untuk melakukan pencarian ada 5 data. (Silahkan pastikan tidak ada data yang sama)
- f. Dari data-data yang ada, tentunya harus dicari kira-kira stok mana yang sudah mau habis.
- g. Untuk pesan keluaran bisa disesuaikan dengan kebutuhan kalian sendiri.
- h. Dikarenakan ada contoh data, maka perlu ada tampilan hasil proses pencarian data (kira-kira bagaimana pesan keluaran ketika diketahui barang tersebut sudah mau habis) dalam bentuk screenshot.

UNIT 4

THE ITERATOR PATTERN (2)

UNIT OVERVIEW

Pada pertemuan kali ini kita akan membahas cara yang dilakukan untuk melakukan penyimpanan sementara pada pola berulang. Hal ini bertujuan supaya kita lebih mudah untuk melakukan manajemen terhadap data yang dimasukkan dan lebih mudah untuk dicari serta mudah untuk diurutkan.

UNIT OBJECTIVES

Adapun capaian yang akan kita dapatkan pada modul ini, yaitu :

- Cara menggunakan berbagai tempat penyimpanan sementara yang ada pada Python.
- Cara menggunakan comprehension untuk memudahkan dalam melakukan perulangan dan penyimpanan

UNIT CONTENTS

Lesson 3: Tempat Penyimpanan Sementara Python	49-
54	
Lesson 4: Comprehension pada Python	55-
57	

PRE LAB

Sebelum masuk ke dalam lab pertama ini, terlebih dahulu kita coba jawab beberapa pertanyaan ini berdasarkan hasil penjelasan pada modul teori!

QUESTION

1. Menurut kalian apa yang dimaksud dengan pola berulang?
2. Menurut kalian, apa saja dampak yang akan terjadi ketika pola berulang tersebut tidak kita selesaikan dengan baik?
3. Dari data-data berikut manakah yang mungkin terjadi pola berulang :
 - Pembuatan kelas
 - Pengecekan data
 - Pencarian data
 - Mencetak data pribadi

CONTENT LESSON

CASE STUDY / PROJECT

Seperti pada pertemuan sebelumnya kita telah membahas mengenai perpustakaan “Home Learning is Best” memiliki banyak data. Dan data tersebut dapat berupa kode buku, nama buku, jenis buku dan jumlah halaman buku. Namun, seperti yang kita ketahui sebelumnya data tersebut sangatlah susah untuk diurutkan. Karena, jumlah buku yang banyak serta sulit untuk dicari 1 per 1.

Nah, gimana ya cara kita membantu perpustakaan tersebut?

IDENTIFICATION CONCEPT OF PROBLEM / PROJECT

Setelah kita baca kembali studi kasus yang diberikan tersebut, tentunya masih ada masalah yang belum terselesaikan pada studi kasus sebelumnya mengenai pengurutan data. Namun, sebelum kita mengurutkan data tentunya kita harus membuat tempat penyimpanan sementara yang bertujuan supaya proses pengurutan dapat sesuai dengan yang diharapkan dan tidak berantakan.

Seperti biasa, sebelum menjawab pertanyaan tersebut mari kita belajar terlebih dahulu!

LESSON 3: TEMPAT PENYIMPANAN SEMENTARA PYTHON

LESSON 3: OVERVIEW

Seperti yang kita ketahui, di dalam Python ada beberapa teknik yang dilakukan untuk tempat penyimpanan data sementara. Seperti yang telah kalian pelajari pada semester 1 dengan menggunakan array. Array bisa dibilang merupakan sekumpulan data yang memiliki jenis sama serta tersimpan dalam 1 buah variabel. Nantinya array ini dapat kita akses 1 per 1 dengan mengakses indeks yang ada di dalam array tersebut. Untuk mengakses perlu diketahui bahwa indeks array dimulai dari 0. Jadi, apabila kita ingin mengakses indeks yang pertama, maka kita harus mengakses indeks ke-0. Seperti contoh tampilan berikut :

```
arrAngka = [1, 2, 3, 4, 5, 6, 7, 8, 9]
print(len(arrAngka)) # Memunculkan panjang dari array arrAngka
print(arrAngka[6]) # Mengakses indeks ke-6 atau urutan ke 7 pada array
```

Figure 65 Array pada Python

Pada contoh tersebut pertama-tama kita deklarasikan array dengan nama arrAngka. Pada array tersebut terdapat nilai 1 - 10. Setelah itu, kita ingin memunculkan panjang dari array tersebut dengan fungsi len. Fungsi ini akan mengembalikan nilai berupa panjang dari array tersebut. Setelah itu kita ingin memunculkan angka ke-7 dari array tersebut atau indeks ke-6. Hal ini sesuai dengan aturan dimana angka ke-n = indeks ke-(n-1). Sehingga untuk mengakses angka ke-7 kita perlu menggunakan indeks ke-6. Dan untuk tampilan dari hasil kode tersebut, sebagai berikut :



Figure 66 Hasil Array

Dari hasil tersebut terlihat jelas bahwa pada baris pertama menunjukkan panjang dari array tersebut, yaitu 9. Yang menandakan terdapat 9 angka pada array arrAngka tersebut. Dan baris kedua memberikan hasil berupa 7, yang menandakan bahwa pada angka ke-7 atau indeks ke-6 pada array arrAngka terdapat nilai 7. Hal ini tentunya sangat mudah untuk diterapkan. Namun, apakah kita hanya bisa menggunakan array saja sebagai media penyimpanan sementara? Jawabannya adalah tidak. Karena, masih ada lagi teknik-teknik lainnya. Salah satunya, yaitu list.

List merupakan sekumpulan data yang tidak berurut. List hampir sama dengan penggunaan yang kita lakukan pada array. Hanya saja, pada list ini kita bisa mengisikan data yang sama serta kita bisa mengisikan berbagai macam jenis data pada 1 buah list yang sama. Sebagai contoh lihat potongan kode berikut :

```
listData = ["Apriyanto Halim", True, 21, 3.61]
print(f"Nama = {listData[0]}")
mhs = "Betul" if listData[1] else "Salah" # Operator Ternary
print(f"Mahasiswa Mikroskil = { mhs }")

# Update Data
listData[2] = 28
print(f"Umur sekarang = {listData[2]}")
del listData[3]
print(f"List Akhir = {listData}")
```

Figure 67 List pada Python

Dari data tersebut terlihat perbedaan yang terjadi dengan array yang telah kita pelajari sebelumnya. Dimana pada sekumpulan data tersebut kita bisa menambahkan berbagai macam data yang ada pada sekumpulan data tersebut. Mulai dari string, boolean, integer dan desimal. Setelah itu, kita juga bisa menggunakan berbagai fungsi yang ada pada list tersebut. Hal ini bertujuan supaya lebih mudah untuk melakukan perubahan pada list tersebut.

Terus, gimana dengan hasilnya? Kita lihat pada tampilan berikut :

```
Nama = Apriyanto Halim
Mahasiswa Mikroskil = Betul
Umur sekarang = 28
List_Aakhir = ['Apriyanto Halim', True, 28]
```

Figure 68 Hasil List

Dari hasil tersebut terlihat bahwa, dengan list kita dapat mengakses dengan cara yang sama seperti yang kita lakukan dengan array, yaitu dengan mengakses indeksnya. Namun, tidak hanya itu, kita juga bisa melakukan modifikasi (perubahan) pada list tersebut dengan mengubah isi pada data, serta menghapus data yang tidak diperlukan. Hal ini semua bisa kita lakukan dalam list. Terus, apakah hanya dua tindakan tersebut saja yang bisa kita lakukan di dalam List? Jawabannya adalah tidak. Silahkan lakukan explorer terdapat fungsi-fungsi atau prosedur yang ada pada list.

Setelah kita belajar list, kita juga bisa melakukan penyimpanan data sementara dengan menggunakan set. Dimana set juga merupakan tempat penyimpanan data sementara yang ada Python. Hanya saja set memiliki perbedaan dengan list, dimana pada set kita hanya bisa menyimpan jenis data yang unik saja serta untuk penggunaannya kita harus menggunakan tanda kurung kurawal ("{}") bukan menggunakan tanda kurung siku ("[]"). Jadi, apabila terdapat data yang sama dalam sekumpulan data tersebut, maka data berikutnya akan dianggap tidak ada atau dihapus. Mari kita lihat potongan kode berikut :

```
setData = {"Apri", 1, 2, "Dosen", (10, 20, 30)}
print("Data set Awal = ", end="")
for i in setData:
    print(i, end=" ")
print()

# Modifikasi Set
setData.add("Dosen")
print(setData)

setData.add("dosen")
print(setData)

setData.remove("Dosen")
print(setData)
```

Figure 69 Set pada Python

NB : Untuk mengakses data yang ada di dalam set, kita tidak bisa melakukan 1 per 1, tapi kita bisa menggunakan bantuan for untuk membantu kita dalam mengakses data yang ada set.

Pada kode tersebut terlihat proses awal sudah ada data berupa “Dosen” dengan karakter “D” huruf kapital. Tapi, pada bagian berikutnya saya menambahkan lagi data yang sama berupa “Dosen” dengan karakter “D” huruf kapital dan karakter “d” huruf kecil. Dan terakhir saya menghapus data “Dosen” dengan karakter “D” huruf kapital. Gimana hasil akhirnya? Kita lihat tampilan berikut :

```
Data set Awal = 1 2 Apri Dosen (10, 20, 30)
{1, 2, 'Apri', 'Dosen', (10, 20, 30)}
{1, 2, 'Apri', 'Dosen', (10, 20, 30), 'dosen'}
{1, 2, 'Apri', (10, 20, 30), 'dosen'}
```

Figure 70 Hasil Set

Pada hasil tersebut terlihat ketika kita data “Dosen” sudah ada di dalam set, maka kita tidak bisa lagi menambahkannya. Sehingga, hasilnya akan sama dengan yang diawal atau tidak ada perubahan. Namun, ketika kita menambahkan data “dosen” dengan karakter “d” huruf kecil, bisa ditambahkan. Ini artinya proses yang ada di dalam set termasuk ke dalam case sensitif. Sehingga ketika kita ingin menghapus salah satu datanya, kita harus jelas jangan sampai salah hapus.

Setelah kita belajar list dan set, ada 1 lagi cara yang bisa kita gunakan untuk melakukan penyimpanan data sementara pada Python adalah dengan menggunakan dict (“dictionary”). Sedikit berbeda cara penggunaan dengan list serta set. Dimana pada dict, kita harus memiliki keys dan values. Dimana keys memiliki peran penting yang kita gunakan sebagai pengganti dari indeks. Sedangkan values merupakan bagian yang merupakan bagian penting berikutnya yang bertugas untuk memberikan nilai pada keys yang kita gunakan sebelumnya. Dan sebagai pembatasnya kita menggunakan tanda titik dua (”:”) untuk memisahkan antara keys dan values. Sebagai contoh potongan simple kode berikut :

```
dataDict = {"Data1" : "Nilai awal"}
```

Berdasarkan potongan kode tersebut terdapat 3 bagian, yaitu :

- dataDict kita gunakan sebagai nama variabel dictionary kita.
- Data1 digunakan sebagai key awal untuk menampung.
- Nilai awal digunakan sebagai nilai untuk key awal yang kita gunakan.

NB : Sebagai tambahan keys yang ada kita juga bisa menggunakan angka untuk keysnya, sedangkan untuk valuenya kita bisa isi berbagai macam jenis data yang ada.

Sebagai contoh perhatikan potongan kode Berikut :

```
dataDict = {"Nama" : "Apriyanto Halim", "Umur" : 27, "Dosen" : True}
print(f"Nama saya = {dataDict['Nama']}")

# Modifikasi Dictionary
dataDict["Umur"] = 28
print(dataDict)

dataDict["Status"] = "Belum Nikah"
print(dataDict)
```

Figure 71 Dictionary pada Python

NB : Ketika mengakses dictionary pada baris kedua, harus diperhatikan ketika kita print menggunakan tanda titik dua("), maka untuk mengakses keys yang ada dictionary kita harus menggunakan tanda petik satu(').

Pada baris kode yang ada ketika kita ingin mengakses data yang ada dictionary, kita harus tahu keys yang digunakan. Ketika kita sudah tahu, maka tinggal dicetak saja dan hasilnya sesuai dengan apa yang kita harapkan. Begitu juga ketika kita ingin mengubah value yang ada dictionary, kita juga harus tahu value untuk keys yang mana yang ini kita ubah. Namun, untuk menambahkan keys baru pada dictionary, kita hanya perlu menggunakan nama keys baru yang belum ada terdaftar pada dictionary dan masukkan valuenya. Sehingga, hasilnya akan menjadi seperti berikut :

```
Nama saya = Apriyanto Halim
{'Nama': 'Apriyanto Halim', 'Umur': 28, 'Dosen': True}
{'Nama': 'Apriyanto Halim', 'Umur': 28, 'Dosen': True, 'Status': 'Belum Nikah'}
```

Figure 72 Hasil Dictionary

Dari hasil tersebut terlihat hasil pada keys “Nama”, dan ketika kita ingin mengubah value pada keys “Umur”, kita tinggal langsung ubah saja dan langsung terubah pada dictionary. Dan untuk menambahkan keys baru, kita hanya perlu menuliskan nama keys baru yang belum ada terdaftar di dalam dictionary yang kita buat. Hal ini tentunya sangat memudahkan kita untuk menggunakan teknik ini untuk penyimpanan data sementara.

Sebagai tambahan untuk teknik dictionary sudah sangat sering digunakan pada format file yaml seperti tampilan dibawah ini :

```
name: John Smith
contact:
    home: 1012355532
    office: 5002586256
address:
    street: |
        123 Tornado Alley
        Suite 16
    city: East Centerville
    state: KS
```

Figure 73 Fomat File Yaml (docs Fileformat)

Tidak hanya itu, dictionary juga digunakan untuk file format json seperti tampilan dibawah ini :

```
{  
    "name": "Jack",  
    "age": 30,  
    "contactNumbers": [  
        {  
            "type": "Home",  
            "number": "123 123-123"  
        },  
        {  
            "type": "Office",  
            "number": "321 321-321"  
        }  
    ],  
    "spouse": null,  
    "favoriteSports": [  
        "Football",  
        "Cricket"  
    ]  
}
```

Figure 74 Format File JSON (docs Fileformat)

Sehingga, penting untuk kalian untuk mempelajari bagian ini untuk melakukan penyimpanan file dalam format json maupun format yaml.

Sebagai tambahan, kita juga bisa menggunakan fungsi sort yang ada list untuk mempermudah kita dalam mengurutkan data yang ada Python seperti potongan kode berikut :

```
>>> a = ["Ayam", "Telur", "Kucing", "Kerbau", "Biawak"]  
>>> a.sort()  
>>> a  
['Ayam', 'Biawak', 'Kerbau', 'Kucing', 'Telur']  
>>> b = [ "Ayam", 3, True, 4.56]  
>>> b.sort()  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: '<' not supported between instances of 'int' and 'str'
```

Figure 75 Hasil Penggunaan Fungsi Sort pada List

Seperti yang terlihat kita bisa mengurutkan data string dan tentunya ini sangat mempermudah kita dalam mengurutkan teks yang ada. Namun, hal ini tidak berlaku untuk jenis data yang tidak sama. Seperti pada variabel b, terdapat jenis data string, integer, Boolean dan desimal.

LESSON 4: COMPREHENSION PADA PYTHON

LESSON 4: OVERVIEW

Comprehension merupakan teknik yang sangat membantu kita dalam melakukan penyaringan terhadap data yang kita kelola. Dimana di dalam objek perulangan tersebut kita bisa memberikan penyaringan data-data apa saja yang ingin kita ambil dan data apa saja yang tidak perlu. Hal ini tentunya sangat membantu kita dalam membuat program. Adapun format yang digunakan pada Comprehension sebagai berikut :

```
newList = [expression for member in iterable (if conditional)]
```

Pada format tersebut terbagi ke dalam 5 bagian, yaitu :

- a. **newList** merupakan nama variabel list yang baru. Variabel ini nantinya akan digunakan untuk menampung hasil penyaringan list yang kita buat sebelumnya.
- b. **expression** merupakan bagian nama variabel yang kita gunakan untuk pengganti pada proses ini. Namun, pada bagian expression ini kita juga bisa memanggil fungsi atau prosedur. Hal ini tentunya sangat membantu kita sekali bukan?
- c. **member** merupakan variabel pengganti yang digunakan untuk mengakses data yang ada di dalam list.
- d. **iterable** merupakan list, array, set, ataupun dictionary yang kita gunakan untuk dilakukan penyaringan. Tentunya ini harus disesuaikan dengan nama variabel list yang kita gunakan.
- e. **if conditional** merupakan kondisi yang digunakan untuk melakukan penyaringan list yang kita gunakan.

Sebagai contoh perhatikan potongan kode berikut :

```
nilaiMhs = [10, 0, -20, 60, 100, 80, 65, 70, 50, 90]
print(nilaiMhs)

# Sebagai nilai untuk dikatakan lulus = 65 keatas
mhsLulus = []
for i in range(len(nilaiMhs)):
    if(nilaiMhs[i] >= 65):
        mhsLulus.append(nilaiMhs[i])
print(mhsLulus)
```

Figure 76 Penyaringan Nilai Mahasiswa tanpa Comprehension

Pada baris kode tersebut terlihat ketika kita ingin melakukan panyaringan kita harus melakukan if (percabangan) untuk melakukan pengecekan nilai mahasiswa, apakah sudah sesuai? Jika sudah sesuai, maka akan ditambahkan ke dalam list baru. Namun, bagi nilai mahasiswa yang tidak sesuaikan akan diabaikan. Sehingga, hasilnya akan menjadi tampilan berikut :

```
[10, 0, -20, 60, 100, 80, 65, 70, 50, 90]
[100, 80, 65, 70, 90]
```

Figure 77 Hasil Penyaringan tanpa Comprehension

Hasilnya tentunya sesuai dengan yang kita harapkan. Dimana pada list kedua, tidak ada nilai yang dibawah dari 65. Hanya bisa dibilang banyak sekali baris kode yang terbuang hanya untuk melakukan pengecekan. Bagaimana caranya kita bisa mengurangi baris tersebut? Caranya adalah dengan menggunakan comprehension. Perhatikan potongan kode berikut :

```
nilaiMhs = [10, 0, -20, 60, 100, 80, 65, 70, 50, 90]
print(nilaiMhs)

# Sebagai nilai untuk dikatakan lulus = 65 keatas
mhsLulus = [nilai for nilai in nilaiMhs if nilai >= 65]
print(mhsLulus)
```

Figure 78 Penyaringan Nilai Mahasiswa dengan Comprehension

Terlihat jumlah barisnya menjadi lebih sedikit dan tentunya hal ini sangat diharapkan ketika dilakukan penerapan secara langsung pada kode. Namun, apakah hasilnya juga sama? Atau tidak sama? Supaya kita tidak pernasaran, perhatikan hasil berikut :

```
[10, 0, -20, 60, 100, 80, 65, 70, 50, 90]
[100, 80, 65, 70, 90]
```

Figure 79 Hasil Penyaringan Nilai dengan Comprehension

Ternyata sama. Hal ini tentunya sangat membantu kita sebagai programmer dalam membuat program. Terus, untuk kondisinya apakah hanya bisa menggunakan if saja? Atau bisa menggunakan else? Perhatikan kembali kode berikut :

```
suhu = [10, -10, 45, 30, 28, 27, 36, 38, 41, 2]
print(suhu)

# Suhu yang ada di Indonesia berada pada rentang 28 sampai 36
# Apabila suhu dibawah dari 28 akan diubah menjadi 0
# Apabila suhu diatas dari 36 akan diubah menjadi 40
# Apabila suhu sesuai range tidak ada perubahan
suhuIndonesia = [x if x >= 28 and x <= 36 else 0 if x < 28 else 40 for x in suhu]
print(suhuIndonesia)
```

Figure 80 Comprehension List dengan If Else

Dari kode tersebut tentunya sedikit rumit untuk dipahami pada awalnya. Hanya saja ketika kita baca per bagianya, maka kita akan dengan jelas tahu apa yang akan dilakukan dan hal apa saja yang akan terjadi jika tidak terpenuhi. Untuk hasil tampilan dari kode tersebut sebagai berikut :

```
[10, -10, 45, 30, 28, 27, 36, 38, 41, 2]
[0, 0, 40, 30, 28, 0, 36, 40, 40, 0]
```

Figure 81 Hasil Comprehension List dengan If Else

Dari hasil tersebut tentunya sangat sesuai dengan kriteria yang telah kita buat sebelumnya. Tentunya ini merupakan cara penyederhanaan filter yang ada dengan menggunakan 1 baris kode.

Setelah kita pelajari kedua pelajaran tersebut, mari kita kembali ke studi kasus kita yang di awal!

SOLUTION

Dari studi kasus kita tersebut terlihat ada penyimpanan data sementara yang dibutuhkan. Hal ini bertujuan supaya lebih untuk masuk ke dalam proses pengurutan. Setelah itu, ada juga format data yang harus ada berupa kode buku, nama buku dan jenis buku. Mungkin format ini kita bisa ambil langsung dari objek buku kita yang ada.

Nah supaya lebih jelas, yuk kita bahas 1 per 1.

INSTRUCTION

1. Tahap awal untuk menyelesaikan masalah tersebut kita membuat kelas BUKU sesuai dengan ketentuan tersebut. Jadi, untuk perintah kita hanya perlu mengetikkan :

```
class BUKU:
```

2. Untuk penyimpanan buku kita menggunakan list perpus :

```
Perpus = []
```

3. Kita buat prosedur untuk memasukkan buku ke dalam list perpus, sehingga menjadi kode seperti berikut :

```
def inputBuku(self, kodeBuku, namaBuku, jenisBuku):  
    tmp = {"kode" : kodeBuku, "nama" : namaBuku, "jenis" : jenisBuku}  
    self.perpus.append(tmp)
```

NB : Untuk format buku, saya memanfaatkan list supaya lebih mudah untuk mengakses elemen yang ada pada buku

4. Setelah itu, untuk melakukan pencetakan buku, kita perlu membuat prosedur juga untuk mencetak buku seperti berikut :

```
def cetakBuku(self):  
    for i in range(len(self.perpus)):  
        print(f"Buku ke-{i+1}:")  
        print(f"Kode Buku = {self.perpus[i]['kode']}")  
        print(f"Nama Buku = {self.perpus[i]['nama']}")  
        print(f"Jenis Buku = {self.perpus[i]['jenis']}")  
        print()
```

NB : Untuk mencetak buku, karena awalnya saya menggunakan list, maka saya menggunakan teknik for i in range untuk mengakses indeks dari buku. Setelah itu, untuk format dari buku saya menggunakan keys yang ada pada dictionary buku.

5. Setelah semua proses selesai, maka selanjutnya kita melakukan proses utama yaitu dengan mengurutkan buku tersebut berdasarkan kode buku, sehingga untuk kode menjadi tampilan berikut ini :

```
def urutBuku(self):  
    self.perpus = sorted(self.perpus, key=lambda d: d['kode'])
```

NB : Untuk kode tersebut sedikit saya menggunakan teknik lambda yang ada pada Python. Teknik ini kita gunakan untuk mengakses keys yang ada pada dictionary. Karena, dictionary tersebut berada pada list, sehingga saya menggunakan lambda untuk mengakses keynya.

6. Setelah semua proses kelas telah selesai, saya memasukkan beberapa data sebagai bahan untuk pengecekan apakah kode yang telah dibuat sudah sesuai atau belum dengan contoh kode sebagai berikut :

```
hlib = BUKU()
hlib.inputBuku("P023", "Pemrograman C", "Pembelajaran")
hlib.inputBuku("K030", "Doraemon", "Komik")
hlib.inputBuku("D067", "Asal Mula Air Hujan", "Dongeng")
hlib.inputBuku("P001", "Pemrograman Python", "Pembelajaran")

print("Sebelum diurutkan")
hlib.cetakBuku()

print("\n\nSetelah diurutkan")
hlib.urutBuku()
hlib.cetakBuku()
```

7. Setelah saja jalankan proses yang ada, maka muncul tampilan berikut :

```
Sebelum diurutkan
Buku ke-1:
Kode Buku = P023
Nama Buku = Pemrograman C
Jenis Buku = Pembelajaran

Buku ke-2:
Kode Buku = K030
Nama Buku = Doraemon
Jenis Buku = Komik

Buku ke-3:
Kode Buku = D067
Nama Buku = Asal Mula Air Hujan
Jenis Buku = Dongeng

Buku ke-4:
Kode Buku = P001
Nama Buku = Pemrograman Python
Jenis Buku = Pembelajaran
```

Figure 82 Hasil Studi Kasus (1)

Setelah diurutkan

Buku ke-1:
Kode Buku = D067
Nama Buku = Asal Mula Air Hujan
Jenis Buku = Dongeng

Buku ke-2:
Kode Buku = K030
Nama Buku = Doraemon
Jenis Buku = Komik

Buku ke-3:
Kode Buku = P001
Nama Buku = Pemrograman Python
Jenis Buku = Pembelajaran

Buku ke-4:
Kode Buku = P023
Nama Buku = Pemrograman C
Jenis Buku = Pembelajaran

Figure 83 Hasil Studi Kasus (2)

EXERCISE

EXERCISE OBJECTIVES

Pada latihan berikut ini mahasiswa diharapkan dapat melakukan :

- Melakukan analisis masalah terhadap studi kasus yang diberikan.
- Mencari solusi atas masalah yang diberikan.
- Menerjemahkan masalah tersebut ke dalam bentuk kode pemrograman.

TASK 1:

Pada kampus Mikroskil tentunya sangat banyak sekali mahasiswa dalam 1 kelas. Hal ini tentunya membuat dosen kesulitan untuk mengetahui siapa saja yang ada di dalam kelas tersebut. Dimana kita ketahui bahwa data mahasiswa tersebut terdiri dari NIM, Nama dan NoHP. Dan supaya dosen lebih mudah untuk mencarinya, maka data mahasiswa tersebut perlu diurutkan **berdasarkan NIM dari mahasiswa tersebut**. Dapatkah kalian membantu dosen ini untuk melakukan absensi untuk kelasnya?

Ketentuan yang dilakukan pada task-01 ini adalah :

- a. Semua proses dilakukan di dalam kelas.
- b. Nama kelas tersebut diharuskan menggunakan nama kalian.
- c. Untuk tindakan yang dilakukan dikelas tersebut terdapat 2, yaitu pemasukan data dan absensi.
- d. Untuk jenis perulangan yang digunakan pada Task-01 ini adalah dengan menggunakan for.
- e. Untuk contoh data yang digunakan untuk melakukan pencarian ada 5 data. (Silahkan pastikan tidak ada data yang sama)

- f. Untuk hasil pencarian tampilkan :
 - o 1 data yang ada di dalam kelas Mikroskil.
 - o 2 data yang tidak ada di dalam kelas Mikroskil.
- g. Untuk pesan keluaran bisa disesuaikan dengan kebutuhan kalian sendiri.
- h. Dikarenakan ada contoh data, maka perlu ada tampilan hasil proses pencarian data (data ditemukan atau tidak) dalam bentuk screenshot.
- i. Kode bisa digunakan kode sebelumnya dan untuk teknik pengurutan silahkan gunakan sesuai dengan yang kalian inginkan.

TASK 2:

Pada sebuah website E-Commerce terdapat berbagai jenis barang yang dijual, diantaranya hand sanitizer, ban mobil, buah-buahan serta botol minuman bayi. Barang tersebut memiliki atribut berupa kode, nama dan jumlah barang. Tentunya barang yang dijual tersebut tidak lebih dari 1 barang sehingga penjual harus tahu kira-kira barang apa saja yang mau habis. Hal ini tentunya perlu dilakukan pencarian dari data yang ada. Dan untuk memudahkan penjual tersebut, maka **urutkan data berdasarkan jenis barangnya**. Dapatkan kalian membantu pedagang tersebut?

Ketentuan yang dilakukan pada task-02 ini adalah :

- a. Semua proses dilakukan di dalam kelas.
- b. Nama kelas tersebut diharuskan menggunakan nama kalian.
- c. Untuk tindakan yang dilakukan dikelas tersebut terdapat 2, yaitu pemasukan data dan pencarian.
- d. Untuk jenis perulangan yang digunakan pada Task-02 ini adalah dengan menggunakan iter.
- e. Untuk contoh data yang digunakan untuk melakukan pencarian ada 5 data. (Silahkan pastikan tidak ada data yang sama)
- f. Dari data-data yang ada, tentunya harus dicari kira-kira stok mana yang sudah mau habis.
- g. Untuk pesan keluaran bisa disesuaikan dengan kebutuhan kalian sendiri.
- h. Dikarenakan ada contoh data, maka perlu ada tampilan hasil proses pencarian data (kira-kira bagaimana pesan keluaran ketika diketahui barang tersebut sudah mau habis) dalam bentuk screenshot.
- j. Kode bisa digunakan kode sebelumnya dan untuk teknik pengurutan silahkan gunakan sesuai dengan yang kalian inginkan.

UNIT 4

THE ITERATOR PATTERN (3)

UNIT OVERVIEW

Pada iterator pattern kali ini kita akan belajar bagaimana cara yang bisa kita lakukan untuk menghemat memori yang ada pada aplikasi yang kita buat. Hal ini tentunya dapat meningkatkan fungsi penyimpanan sementara yang ada, dan dapat membuat sistem menjadi lebih ringan. Hal ini tentunya juga bisa kita terapkan pada pengecekan kode atau data yang ada pada sistem berjalan. Sehingga, proses penginputan data dapat berjalan dengan baik.

UNIT OBJECTIVES

Adapun capaian yang akan kita dapatkan pada modul ini, yaitu :

- Cara membuat pengujian fungsi dengan generator.
- Cara membuat pengujian fungsi dengan coroutine

UNIT CONTENTS

Lesson 5: Pengecekan dengan Generator	62-
66	
Lesson 6: Pengecekan dengan Coroutine	66-
67	

PRE LAB

Sebelum masuk ke dalam lab pertama ini, terlebih dahulu kita coba jawab beberapa pertanyaan ini berdasarkan hasil penjelasan pada modul teori!

QUESTION

1. Apa fungsi dari penggunaan Comprehension?
2. Fungsi dari dictionary hampir sama dengan format file :
3. Dari daftar nama-nama proses berikut, manakah yang membutuhkan tempat penyimpanan sementara :
 - Mencetak nama mahasiswa
 - Mencetak seluruh nama mahasiswa
 - Melakukan penjumlahah 2 buah angka
 - Melakukan penjumlahah 10 buah angka

CONTENT LESSON

CASE STUDY / PROJECT

Pada perpustakaan “Home Learning is Best”, ingin melakukan perubahan terhadap proses kerja yang dilakukan. Hal ini dikarenakan, zaman semakin berkembang dan kebutuhan terhadap data sangat besar, sehingga setiap proses barang yang masuk perlu dilakukan pengecekan satu per satu. Data yang dicek, dimulai dari kode, nama dan jenis buku. Hal ini bertujuan supaya data yang dimasukkan tidaklah double dan sesuai dengan data yang ada.

Dapatkankah kamu kembali membantu melakukan pengecekan pada perpustakaan kita ini?

IDENTIFICATION CONCEPT OF PROBLEM / PROJECT

Setelah kita baca kembali masalah yang ada di dalam studi kasus tersebut bahwa yang menjadi permasalahan adalah data yang banyak. Karena, data yang banyak diperlukan proses pengecekan data yang masuk untuk memastikan tidak ada data yang ganda. Oleh karena itu, diperlukannya proses pengecekan ke dalam sistem untuk memastikan data yang dimasukkan bukanlah data yang sama dengan data yang ada di dalam sistem.

Bagaimana cara kita mengeceknya? Sebelum kita mengecek mari kita pelajari beberapa pelajaran berikut!

LESSON 5: PENGECEKAN DENGAN GENERATOR

LESSON 5: OVERVIEW

Generator merupakan sebuah fungsi yang dapat digunakan untuk mengembalikan objek yang berulang. Pengembalian objek tersebut tidak lagi dengan menggunakan statement return, melainkan dengan menggunakan yield. Hal ini tertentunya memiliki perbedaan. Dimana ketika kita menggunakan return, kita hanya mengembalikan sebuah nilai untuk 1 kali perkerjaan dalam 1 waktu. Sedangkan, dengan menggunakan yield kita bisa menghasilkan sekumpulan data yang berbeda dalam 1 waktu, meskipun prosesnya tetap harus dilakukan secara berulang untuk mendapatkan hasil sekumpulan data yang kita inginkan.

Sebagai contoh perhatikan kode berikut :

```
def fungsiAB(a, b):
    return a + b
    return a - b

def fungsiAB1(a, b):
    yield a + b
    yield a - b
```

Figure 84 Perbedaan Fungsi dengan Generator

Berdasarkan dari kode tersebut, terlihat sedikit berbeda pada bagian fungsiAB yang pertama dimana, ketika kita melakukan melakukan return pertama kali masih kelihatan dengan jelas hasilnya. Sedangkan, ketika kita kembali menggunakan return terlihat sedikit gelap hasilnya. Mengapa demikian? (Proses ini akan terlihat lebih jelas ketika dijalankan nanti).

Berbeda dengan fungsiAB1 yang kedua dimana, proses yield dilakukan dua kali kelihatan sama persis dan tidak ada gelapnya. Namun, apakah hasilnya akan sama? Kita lihat dari tampilan hasil berikut :

```
>>> a = fungsiAB(2,3)
>>> print(a)
5
>>> print(a)
5
>>> b = fungsiAB1(2,3)
>>> print(b)
<generator object fungsiAB1 at 0x000001AD2AF33D00>
>>> print(next(b))
5
>>> print(next(b))
-1
```

Figure 85 Hasil Perbedaan Generator dan Fungsi

Dari hasil tersebut terlihat dengan jelas untuk fungsi hanya memunculkan hasil yang pertama saja. Sedangkan, untuk hasil berikutnya tidak muncul. Hal ini dikarenakan proses yang ada fungsi hanya bisa mengembalikan sebuah nilai untuk 1 kali proses. Sedangkan untuk generator kita bisa melakukan beberapa cetak dan menghasilkan nilai yang berbeda-beda. Hal ini dikarenakan, yield yang kita masukkan lebih dari 1, sehingga memungkinkan kita untuk menghasilkan beberapa nilai yang berbeda-

beda tergantung dari jumlah yield yang digunakan. Dengan begitu, kita tidak perlu repot-repot untuk membuat fungsi lain untuk melakukan dua proses yang hampir sama.

Pada penerapannya kita juga bisa melakukan pengecekan terhadap perulangan yang kita lakukan. Dimana ketika terjadi keanehan pada proses kita bisa langsung melakukan pengecekan dan segera memperbaiki keanehan tersebut. Sebagai contoh perhatikan kode berikut :

```
import math
...
x1,2 = [(-b) +/- akar(b*b - 4 * a * c)] / (2*a)
...
def rumusABC(a, b, c):
    yield b * b - 4 * a * c #Mengeluarkan nilai hasil determinan
    yield 2 * a #Mengeluarkan nilai pembagi
    yield ((-b) + math.sqrt(b*b - 4 * a * c)) / (2*a) #Menghasilkan nilai x1
    yield ((-b) - math.sqrt(b*b - 4 * a * c)) / (2*a) #Menghasilkan nilai x2
```

Figure 86 Penerapan Generator pada Rumus ABC

Pada kode tersebut dilakukan keluaran untuk setiap bagian yang ada pada rumus ABC. Dimana seperti yang kita ketahui, rumus ABC bisa ditulis sebagai berikut :

The diagram shows the quadratic formula $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ enclosed in a red-bordered box. At the top center of the box is the text "Rumus ABC". Below the formula, at the bottom center, is the text "By : Rumus.Co.Id".

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Figure 87 Rumus ABC

Dari rumus tersebut kita lihat dari tiap prosesnya harus dipastikan terlebih dahulu. Dimana, ketika kita ingin melakukan pengakaran pastikan nilai dari akar tersebut merupakan nilai yang positif bukanlah nilai negatif. Karena, apabila memberikan hasil berupa negatif, maka akan menghasilkan nilai yang tidak terhingga (tidak memiliki nilai pasti). Yang kedua, kita juga harus memastikan bagian pembagi tidak menghasilkan nilai 0. Apabila menghasilkan nilai 0, maka akan memberikan hasil berupa tidak terhingga juga (tidak memiliki nilai pasti).

Setelah kita memastikan kedua hal tersebut, maka kembali kita jalankan kode tersebut dengan menghasilkan tampilan berikut :

```
>>> a = rumusABC(2, 6, -56)
>>> print(a)
<generator object rumusABC at 0x0000023A63F43D00>
>>> print(f"Hasil Determinan: {next(a)}")
Hasil Determinan: 484
>>> print(f"Nilai Pembagi: {next(a)}")
Nilai Pembagi: 4
>>> print(f"Nilai X1: {next(a)}")
Nilai X1: 4.0
>>> print(f"Nilai X2: {next(a)}")
Nilai X2: -7.0
```

Figure 88 Hasil Rumus ABC dengan Generator (Contoh Benar)

Dari hasil tersebut kita bisa melihat setiap proses yang dilakukan pada perhitungan tersebut, mulai dari penentuan determinan, penentuan nilai pembagi, penentuan nilai x1 dan nilai x2. Hal ini tentunya dapat membuat kita menjadi lebih mengerti terhadap proses perhitungan yang dilakukan. Bagaimana dengan contoh berikut :

```
>>> a = rumusABC(0, 3, 7)
>>> print(a)
<generator object rumusABC at 0x0000029FF37D3D00>
>>> print(f"Hasil Determinan: {next(a)}")
Hasil Determinan: 9
>>> print(f"Nilai Pembagi: {next(a)}")
Nilai Pembagi: 0
>>> print(f"Nilai X1: {next(a)}")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    File "Contoh2.py", line 8, in rumusABC
      yield ((-b) + math.sqrt(b*b - 4 * a * c)) / (2*a) #Menghasilkan nilai x1
ZeroDivisionError: float division by zero
>>> print(f"Nilai X2: {next(a)}")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

Figure 89 Hasil Rumus ABC dengan Generator (Contoh Salah)

Dari yang kita lihat ternyata pada saat proses dijalankan terjadi kesalahan untuk mendapatkan nilai x1 dan x2. Mengapa demikian? Coba kita perhatikan pada bagian nilai pembagi. Berapa nilai yang diberikan? Nilai pembagi = 0. Hal inilah yang membuat program yang kita jalankan mengalami kendala berupa nilai yang tidak sesuai atau “float division by zero” yang memberikan arti nilai float yang

dibagi dengan angka 0. Tentunya hal ini tidak bisa kita lakukan. Karena, berapapun akan yang dihasilkan apabila dibagian dengan angka 0, akan memberikan hasil berupa tidak terhingga (tidak bisa dipastikan). Oleh karena itu, dengan adanya generator ini kita bisa melakukan pengecekan asal muasalnya kode yang kita buat tersebut mengalami kesalahan. Namun, untuk penggunaan generator ini akan kembali kita temui pada saat nanti kita mempelajari proses testing pada OOP yang akan diberikan pada pertemuan-13.

Setelah kita mempelajari pengecekan dengan menggunakan Generator, berikutnya kita juga akan mempelajari pengecekan juga dengan menggunakan coroutine. Gimana prosesnya? Dan apakah sama saja dengan proses yang ada generator? Supaya tidak berpanjang lebar lagi, mari kita pelajari pada pelajaran berikutnya!

LESSON 6: PENGECEKAN DENGAN COROUTINE

LESSON 6: OVERVIEW

Pada proses pengecekan yang ada Python ternyata ada 1 lagi cara yang bisa kita gunakan, yaitu coroutine. Dimana proses pada coroutine hampir sama dengan yang ada pada generator sama-sama menggunakan statement yield. Hanya saja ada terdapat perbedaan yang mendasar pada kedua proses pengecekan, yaitu :

- Pada generator kita melakukan pengecekan dengan memberikan keluaran dari proses yang dilakukan.
- Sedangkan pada corouting kita melakukan pengecekan pada masukkan yang diberikan pada proses yang dilakukan.

Jadi, terlihat dengan jelas bahwa generator untuk keluaran, sedangkan coroutine untuk masukkan. Adapun contoh penerapan pada coroutine sebagai berikut :

```
def stokBarang():
    jlh = 0
    while True:
        brg = yield jlh
        jlh += brg
```

Figure 90 Contoh Coroutine

Pada kode tersebut terlihat perbedaan peletakkan statement yield yang berada pada posisi inputan yang diberikan kepada variabel brg. Terus, gimana caranya kita memberikan nilai kepada variabel tersebut?

Jawabannya adalah dengan memanfaatkan send yang ada pada coroutine seperti tampilan berikut :

```
>>> a = stokBarang()
>>> print(f"Jumlah stok awal = {next(a)} barang")
Jumlah stok awal = 0 barang
>>> print(f"Jumlah stok setelah ditambah 3 barang = {a.send(3)} barang")
Jumlah stok setelah ditambah 3 barang = 3 barang
>>> print(f"Jumlah stok setelah ditambah 10 barang = {a.send(10)} barang")
Jumlah stok setelah ditambah 10 barang = 13 barang
>>> print(f"Jumlah stok setelah dikurang 5 barang = {a.send(-5)} barang")
Jumlah stok setelah dikurang 5 barang = 8 barang
```

Figure 91 Hasil Coroutine

Dari hasil tersebut terlihat bahwa proses pembaruan jumlah barang yang ada dapat dilakukan. Apakah melakukan penambahan maupun pengurangan. Hal ini bertujuan supaya kita tetap bisa mengetahui status dari jumlah barang yang tersimpan pada proses yang dilakukan. Tentunya hal ini dapat membantu kita dalam melakukan pengecekan terhadap pemasukan data yang dilakukan.

Nah setelah kita belajar mengenai cara pengecekan, mari kita kembali ke studi kasus awal kita!

SOLUTION

Dari studi kasus kita terlihat bahwa kita ingin melakukan pengecekan terhadap pemasukan data yang diberikan. Oleh karena itu, cara pengecekan yang bisa kita terapkan pada studi kasus tersebut adalah dengan memanfaatkan proses coroutine. Hal ini bertujuan setiap ada pemasukan data, data yang dimasukkan terlebih dahulu dicek, dan dipastikan memang masih belum ada di dalam sistem.

Nah supaya lebih jelas, yuk kita modifikasi sedikit fungsi masukkan yang telah kita buat sebelumnya!

INSTRUCTION

1. Fungsi pemasukan (inputan) sebelum ada coroutine :

```
def inputBuku(self, kodeBuku, namaBuku, jenisBuku):
    tmp = {"kode" : kodeBuku, "nama" : namaBuku, "jenis" : jenisBuku}
    self.perpus.append(tmp)
```

Pada kode tersebut, terlihat proses yang dilakukan masih merupakan proses sederhana dan belum ada pengecekan apakah barang yang dimasukkan sudah ada sebelumnya atau belum. Sehingga, kita perlu melakukannya dengan menggunakan coroutine menjadi seperti berikut :

```
def inputBuku(self):
    kd = ""
    nm = ""
    jns = ""
    cek = True
    while cek:
        kodeBuku = yield kd
        if(self.perpus < 1):
            cek = False
        else:
            cek = False
            for i in range(len(self.perpus)):
                if(self.perpus[i]['kode'] == kodeBuku):
                    cek = True
```

```

if(cek):
    print("Kode Buku sudah terdaftar ... !!!")
    print("Masukkan kode buku lagi ... !!!")
namaBuku = yield nm
jenisBuku = yield jns
tmp = {"kode" : kodeBuku, "nama" : namaBuku, "jenis" : jenisBuku}
self.perpus.append(tmp)

```

Dari kode tersebut terlihat adanya pengecekan yang dilakukan pada saat penginputan buku yang dilakukan. Hal ini bertujuan supaya kode buku yang tersimpan tidak ada kode yang sama.

- Untuk tampilan hasilnya sebagai berikut :

Hasil inputan awal :

```

>>> a = BUKU()
>>> b = a.inputBuku()
>>> print(b)
<generator object BUKU.inputBuku at 0x0000026D3E0E1D58>
>>> next(b)
'Kode kosong'
>>> b.send("IF001")
'IF001'
>>> b.send("Pemrograman Python")
'Pemrograman Python'
>>> b.send("Pemrograman")
>>> StopIteration

```

Figure 92 Hasil Studi Kasus (1)

Data tersimpan :

```

>>> a.cetakBuku()
Buku ke-1:
Kode Buku = IF001
Nama Buku = Pemrograman Python
Jenis Buku = Pemrograman

```

Figure 93 Hasil Studi Kasus (2)

Hasil inputan kedua :

```
>>> b = a.inputBuku()
>>> print(b)
<generator object BUKU.inputBuku at 0x000001C2E5448258>
>>> next(b)
'Kode kosong'
>>> b.send("IF001")
Kode Buku sudah terdaftar ... !!!
Masukkan kode buku lagi ... !!!
'Kode kosong'
>>> b.send("IF002")
'IF002'
>>> b.send("Bahasa C")
'Bahasa C'
>>> b.send("Pemrograman")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

Figure 94 Hasil Studi Kasus (3)

Data tersimpan :

```
>>> a.cetakBuku()
Buku ke-1:
Kode Buku = IF001
Nama Buku = Pemrograman Python
Jenis Buku = Pemrograman

Buku ke-2:
Kode Buku = IF002
Nama Buku = Bahasa C
Jenis Buku = Pemrograman
```

Figure 95 Hasil Studi Kasus (4)

EXERCISE

EXERCISE OBJECTIVES

Pada latihan berikut ini mahasiswa diharapkan dapat melakukan :

- Melakukan analisis masalah terhadap studi kasus yang diberikan.
- Mencari solusi atas masalah yang diberikan.
- Menerjemahkan masalah tersebut ke dalam bentuk kode pemrograman.

TASK 1:

Pada kampus Mikroskil tentunya sangat banyak sekali mahasiswa dalam 1 kelas. Hal ini tentunya membuat dosen kesulitan untuk mengetahui siapa saja yang ada di dalam kelas tersebut. Dimana kita ketahui bahwa data mahasiswa tersebut terdiri dari NIM, Nama dan NoHP. Dan supaya dosen lebih mudah untuk mencarinya, maka data mahasiswa tersebut perlu diurutkan **berdasarkan NIM dari mahasiswa tersebut**. Setelah data tersebut dimasukkan, maka akan muncul **tampilan berupa jumlah mahasiswa** yang sudah di dalam kelas. Dapatkan kalian membantu dosen ini untuk melakukan absensi untuk kelasnya?

Ketentuan yang dilakukan pada task-01 ini adalah :

- a. Semua proses dilakukan di dalam kelas.
- b. Nama kelas tersebut diharuskan menggunakan nama kalian.
- c. Untuk tindakan yang dilakukan dikelas tersebut terdapat 2, yaitu pemasukan data dan absensi.
- d. Untuk jenis perulangan yang digunakan pada Task-01 ini adalah dengan menggunakan for.
- e. Untuk contoh data yang digunakan untuk melakukan pencarian ada 5 data. (Silahkan pastikan tidak ada data yang sama)
- f. Untuk hasil pencarian tampilkan :
 - o 1 data yang ada di dalam kelas Mikroskil.
 - o 2 data yang tidak ada di dalam kelas Mikroskil.
- g. Untuk pesan keluaran bisa disesuaikan dengan kebutuhan kalian sendiri.
- h. Dikarenakan ada contoh data, maka perlu ada tampilan hasil proses pencarian data (data ditemukan atau tidak) dalam bentuk screenshot.
- i. Pada proses ini diperlukan cara untuk melakukan pengecekan. Silahkan kalian pikirkan sendiri apakah harus menggunakan generator atau menggunakan coroutine.
- j. Kode bisa digunakan kode sebelumnya dan untuk teknik pengurutan silahkan gunakan sesuai dengan yang kalian inginkan.

TASK 2:

Pada sebuah website E-Commerce terdapat berbagai jenis barang yang dijual, diantaranya hand sanitizer, ban mobil, buah-buahan serta botol minuman bayi. Barang tersebut memiliki atribut berupa kode, nama dan jumlah barang. Tentunya barang yang dijual tersebut tidak lebih dari 1 barang sehingga penjual harus tahu kira-kira barang apa saja yang mau habis. Hal ini tentunya perlu dilakukan pencarian dari data yang ada. Dan untuk memudahkan penjual tersebut, maka **urutkan data berdasarkan jenis barangnya**. Penjual tersebut ingin mengetahui **kira-kira stok secara keseluruhan** yang ada di dalam gudangnya sudah berapa. Dapatkan kalian membantu pedagang tersebut?

Ketentuan yang dilakukan pada task-02 ini adalah :

- a. Semua proses dilakukan di dalam kelas.
- b. Nama kelas tersebut diharuskan menggunakan nama kalian.
- c. Untuk tindakan yang dilakukan dikelas tersebut terdapat 2, yaitu pemasukan data dan pencarian.
- d. Untuk jenis perulangan yang digunakan pada Task-02 ini adalah dengan menggunakan iter.
- e. Untuk contoh data yang digunakan untuk melakukan pencarian ada 5 data. (Silahkan pastikan tidak ada data yang sama)
- f. Dari data-data yang ada, tentunya harus dicari kira-kira stok mana yang sudah mau habis.
- g. Untuk pesan keluaran bisa disesuaikan dengan kebutuhan kalian sendiri.
- h. Dikarenakan ada contoh data, maka perlu ada tampilan hasil proses pencarian data (kira-kira bagaimana pesan keluaran ketika diketahui barang tersebut sudah mau habis) dalam bentuk screenshot.

- i. Pada proses ini diperlu cara untuk melakukan pengecekan. Silahkan kalian pikirkan sendiri apakah harus menggunakan generator atau menggunakan coroutine.
- k. Kode bisa digunakan kode sebelumnya dan untuk teknik pengurutan silahkan gunakan sesuai dengan yang kalian inginkan.

UNIT 5

DESIGN PATTERN 1 (1)

UNIT OVERVIEW

Design pattern merupakan sebuah blueprint (framework/kerangka) yang sudah ada sebelumnya dan biasanya digunakan sebagai salah satu pemecahan permasalahan yang ada di dalam pemrograman. Hal ini tentunya dapat membantu kita untuk lebih memudahkan dalam pembuatan program. Karena, sering kali program yang kita buat tidak sesuai dengan apa yang kita inginkan atau program yang kita buat memiliki jumlah baris yang banyak. Hal ini tentunya dapat membuat program yang kita buat menjadi kurang efektif.

UNIT OBJECTIVES

Adapun capaian yang akan kita dapatkan pada modul ini, yaitu :

- Cara menyelesaikan masalah dengan Decorator Pattern
- Cara menyelesaikan masalah dengan Observer Pattern
- Cara menyelesaikan masalah dengan Strategy Pattern

UNIT CONTENTS

Lesson 1: The Decorator Pattern	73-
	75
Lesson 2: The Observer Pattern	75-
	78
Lesson 3: The Strategy Pattern	78-
	80

PRE LAB

Sebelum masuk ke dalam lab pertama ini, terlebih dahulu kita coba jawab beberapa pertanyaan ini berdasarkan hasil penjelasan pada modul teori!

QUESTION

1. Apa yang membedakan antara generator dengan coroutine?
2. Apa yang membuat generator bisa dibilang sebagai penghemat memori?
3. Dari daftar berikut, manakah yang sesuai dimasukkan ke dalam Coroutine atau Generator :
 - Pembuatan daftar nama pegawai satu per satu.
 - Memasukkan data 1 per 1.

CONTENT LESSON

CASE STUDY / PROJECT

Pada perpustakaan “Home Learning is Best” terjadi perubahan manajemen. Hal ini dikarenakan manajemen sebelumnya tidak begitu baik dalam melakukan pengelolaan, sehingga diganti. Pada manajemen kali ini, diharapkan semua proses sudah dilakukan secara otomatis. Mulai dari pengecekan keterbaruan buku, sampai pengecekan jumlah buku yang tersedia.

Kali ini ada perubahan manajemen nih. Sehingga, ada perubahan proses yang harus kita pahami kembali. Oleh karena itu, yuk kita bantu perpustakaan kita ini !

IDENTIFICATION CONCEPT OF PROBLEM / PROJECT

Setelah kita baca kembali studi kasus yang diberikan tersebut, kita perlu membuat sebuah fungsi untuk dapat melakukan tindakan berupa pengecekan keterbaruan buku serta pengecekan stok buku. Tidak hanya itu saja, pada proses penggerjaan ini kita juga harus memperhatikan kira-kira solusi apa yang harus kita tawarkan supaya proses pembuatan kode yang kita buat nanti tidak terlalu melebar.

Sebelum kita menjelaskan solusi dari permasalahan tersebut, mari kita simak dulu beberapa penjelasan untuk membantu kita dalam menyelesaikan permasalahan yang terdapat dalam studi kasus tersebut !

LESSON 1: THE DECORATOR PATTERN

LESSON 1: OVERVIEW

The decorator pattern merupakan sebuah pattern (pola) yang dapat digunakan untuk menambahkan fungsionalitas dari sebuah tindakan tanpa perlu merubah struktur dari kode tersebut.

Berikut contoh perhatikan kode berikut :

```
class WrittenText:

    """Represents a Written text"""

    def __init__(self, text):
        self._text = text

    def render(self):
        return self._text

class UnderlineWrapper(WrittenText):

    """Wraps a tag in <u>"""

    def __init__(self, wrapped):
        self._wrapped = wrapped

    def render(self):
        return "<u>{}</u>".format(self._wrapped.render())
```

Figure 96 Contoh Penerapan Decorator Pattern (1)

Pada kode tersebut bertujuan supaya kita bisa memberikan bentuk format tag “u” yang ada pada HTML untuk membuat sebuah teks menjadi underline (garis bawah). Tentunya ini bagian yang sangat penting kita ingin membuat dalam bentuk tag “u”. Adapun hasil dari kode tersebut seperti berikut :

```
>>> awal = WrittenText("Hai Apa Kabar?")
>>> print(f"Awal = {awal.render()}")
Awal = Hai Apa Kabar?
>>> akhir = UnderlineWrapper(awal)
>>> print(f"Akhir = {akhir.render()}")
Akhir = <u>Hai Apa Kabar?</u>
```

Figure 97 Hasil Penerapan Decorator Pattern (1)

Dari hasil tersebut terlihat kita bisa menambahkan tag “u” ke dalam teks yang kita buat. Hal ini tentunya dapat membuat teks kita menjadi lebih baik dan membuat dengan gaya-gaya yang baru. Namun, bagaimana kalau saya juga ingin menambahkan tag “b” (bold) ke dalam teks tersebut? Apakah saya perlu mengubah kode saya sebelumnya? Jawabannya adalah tidak. Kita tidak perlu menugbah kode kita sebelumnya. Kita hanya perlu menambahkan fungsi baru saja.

Inilah yang dinamakan keunggulan dari penggunaan Decorator Pattern. Perhatikan potongan kode tambahan berikut :

```
class BoldWrapper(WrittenText):  
  
    """Wraps a tag in <b>"""  
  
    def __init__(self, wrapped):  
        self._wrapped = wrapped  
  
    def render(self):  
        return "<b>{}</b>".format(self._wrapped.render())
```

Figure 98 Contoh Penerapan Decorator Pattern (2)

Nah kita tidak perlu kembali ke kode sebelumnya, kita hanya perlu menambahkan fungsi baru saja. Terus, apakah hasilnya sesuai dengan yang kita harapkan? Silahkan lihat contoh hasil berikut :

```
>>> awal = WrittenText("Hai Apa Kabar?")  
>>> print(f"Awal = {awal.render()}")  
Awal = Hai Apa Kabar?  
>>> akhir = BoldWrapper(UnderlineWrapper(awal))  
>>> print(f"Akhir = {akhir.render()}")  
Akhir = <b><u>Hai Apa Kabar?</u></b>
```

Figure 99 Hasil Penerapan Decorator Pattern (2)

Gimana hasilnya? Sesuai dengan apa yang kita inginkan bukan? Nah bagaimana kalau saya ingin menambahkan tag “I” (italic) ke dalam teks saya tersebut? Silahkan kalian explorer ya !

Sumber kode : [GeeksforGeeks](#).

Untuk design pattern ternyata bukan hanya ada Decorator Pattern saja, tapi ada juga observer pattern. Bagaimana cara penerapannya? Apakah sama dengan Decorator pattern? Mari kita lihat pada pelajaran berikutnya!

LESSON 2: THE OBSERVER PATTERN

LESSON 2: OVERVIEW

The observer pattern merupakan pola yang digunakan untuk melakukan pemantauan terhadap kejadian, situs atau penangan terhadap perubahan peristiwa. Hal ini tentunya dapat dilakukan pada proses keadaan alam yang terjadi pada saat ini. Kita tidak tahu dan tidak bisa memastikan apakah pagi hari akan terjadi hujan atau berubah menjadi siang ataupun sore. Karena, kita mungkin sulit untuk dapat langsung melakukan prediksi terhadap cuaca.

Perhatikan contoh kode berikut :

```
class Subject:

    """Represents what is being observed"""

    def __init__(self):
        """create an empty observer list"""

        self._observers = []

    def notify(self, modifier = None):
        """Alert the observers"""

        for observer in self._observers:
            if modifier != observer:
                observer.update(self)

    def attach(self, observer):
        """If the observer is not in the list,
        append it into the list"""

        if observer not in self._observers:
            self._observers.append(observer)

    def detach(self, observer):
        """Remove the observer from the observer list"""

        try:
            self._observers.remove(observer)
        except ValueError:
            pass
```

Figure 100 Contoh Penerapan Observer Pattern (1)

Sebelum kita memulai, terlebih dahulu kita desain kelas awal yang akan menjadi bagian kelas utama. Pada kelas ini kita tentunya membuat notify yang berguna nanti sebagai pemberitahuan, attach untuk memasukkan observer ke dalam bagian, dan detach untuk mengeluarkan observer yang tidak ada kaitannya. Seperti contoh kode berikut :

```
class Data(Subject):

    """monitor the object"""

    def __init__(self, name = ''):
        Subject.__init__(self)
        self.name = name
        self._data = 0

    @property
    def data(self):
        return self._data

    @data.setter
    def data(self, value):
        self._data = value
        self.notify()
```

Figure 101 Contoh Penerapan Observer Pattern (2)

Selanjutnya kita menambahkan kelas khusus yang bertanggung jawab terhadap data yang akan kita proses nantinya. Jadi, kita sudah siapkan kelas bernama Data. Pada kelas ini, kita bisa menambahkan @properties yang bertujuan menghasilkan sebuah nilai data, ketika pengguna ingin memberitahu nilainya, sedangkan kita bisa menggunakan @data.setter yang berfungsi sebagai proses yang memberikan tindakan ketika terjadi penginputan nilai ke dalam data.

Nah setelah semuanya selesai, maka kita tidak boleh lupa adalah membuat obsevernya. Karena, tujuan dari kita membuat ini adalah untuk membuat observernya. Untuk observernya bisa dibuat dalam bentuk berikut :

```
class HexViewer:

    """updates the Hexviewer"""

    def update(self, subject):
        print('HexViewer: Subject {} has data 0x{:x}'.format(subject.name, subject.data))
```

Figure 102 Contoh Penerapan Observer Pattern (3)

Nah tujuan dari obsever ini nantinya adalah melakukan konversi terhadap angka ke dalam bentuk heksa desimal. Dimana nilai heksa desimal merupakan nilai yang dimulai dari 0 sampai dengan 9 dan dilanjutkan A sampai dengan F. Terus, kalau dijalankan gimana hasilnya ya? Perhatikan hasil kode berikut :

```
>>> data1 = Data("Data Pertama")
>>> data2 = Data("Data Kedua")
>>> view = HexViewer()
>>> data1.attach(view)
>>> data2.attach(view)
>>> data1.data = 100
HexViewer: Subject Data Pertama has data 0x64
>>> data2.data = 200
HexViewer: Subject Data Kedua has data 0xc8
```

Figure 103 Hasil Penerapan Observer Pattern (1)

Dari hasil tersebut terlihat kita bisa membuat observer yang bertujuan untuk mengubah angka desimal menjadi angka heksa desimal. Terus, bagaimana kalau kita ingin menambahkan observer baru yang menampilkan angka desimal kembali? Apakah kita harus mengubah kode juga atau gimana? Jawabannya simple, yaitu tidak perlu. Karena, inilah tujuan kita belajar observer pattern. Jadi, kita hanya perlu menambahkan kode dibawah ini :

```
class DecimalViewer:

    """updates the Decimal viewer"""

    def update(self, subject):
        print('DecimalViewer: Subject % s has data % d' % (subject.name, subject.data))
```

Figure 104 Contoh Penerapan Observer Pattern (4)

Pada kode tersebut, kita perlu lagi memikirkan untuk merubah kode. Kita hanya perlu menambahkan kodennya saja. Terus, gimana hasilnya? Silahkan lihat hasil kode berikut :

```
>>> data1 = Data("Data Pertama")
>>> data2 = Data("Data Kedua")
>>> view1 = HexViewer()
>>> view2 = DecimalViewer()
>>> data1.attach(view1)
>>> data1.attach(view2)
>>> data2.attach(view1)
>>> data2.attach(view2)
>>> data1.data = 100
HexViewer: Subject Data Pertama has data 0x64
DecimalViewer: Subject Data Pertama has data 100
>>> data2.data = 200
HexViewer: Subject Data Kedua has data 0xc8
DecimalViewer: Subject Data Kedua has data 200
```

Figure 105 Hasil Penerapan Observer Pattern (2)

Gimana hasilnya? Sesuai dengan harapan bukan? Dan kita update bukan hanya pada 1 observer saja tetapi juga kepada observer yang kedua. Hal ini tentunya dapat meminimalkan waktu yang diperlukan untuk mencetak kedua observer tersebut. Terus, bagaimana kalau dengan oktal? Silahkan kalian explorer sendiri.

Sumber kode : [GeeksforGeeks](#).

Gimana? Tentunya kalian lebih mudah untuk membuat program ketimbang hanya takut Ketika kode yang dibuat apakah akan menjadi banyak dan susah untuk dibuat tidak ya? Mari kita belajar pada pelajaran selanjutnya, yaitu mengenai Strategy Pattern.

LESSON 3: THE STRATEGY PATTERN

LESSON 3: OVERVIEW

The strategy pattern merupakan pola yang digunakan untuk membuat dua kode algoritma yang berbeda. Hal ini bertujuan supaya kita tidak perlu lagi harus membuat kelas lain untuk kode algoritma yang lain. Karena, biasanya dalam penelitian kita tidak hanya menggunakan 1 buah metode saja, tapi bisa lebih dari 1. Nah gimana solusi untuk mengatasinya?

Perhatikan contoh kode berikut :

```
"""A separate class for Item"""
class Item:

    """Constructor function wth price and discount"""

    def __init__(self, price, discount_strategy = None):

        """take price and discount strategy"""

        self.price = price
        self.discount_strategy = discount_strategy

    """A separate function for price after discount"""

    def price_after_discount(self):

        if self.discount_strategy:
            discount = self.discount_strategy(self)
        else:
            discount = 0

        return self.price - discount

    def __repr__(self):

        statement = "Price: {}, price after discount: {}"
        return statement.format(self.price, self.price_after_discount())
```

Figure 106 Contoh Penerapan Strategy Pattern (1)

Pada contoh kode tersebut terlihat kita awal membuat sebuah fungsi `price_after_discount` yang merupakan fungsi yang akan berpengaruh terhadap `strategy` yang akan kita buat nantinya. Dan setelah itu, dikode tersebut juga terlihat ada fungsi berupa `repr` (representasi) yang berfungsi untuk mengganti pesan yang ada pada objek ketika objek tersebut dibuat. Hal ini bertujuan supaya yang keluar nantinya, bukan lokasi dari dimana objek tersebut disimpan tapi lebih kepada bagaimana pesan yang ingin disampaikan. Setelah kita sudah buat kelas utamanya, selanjutnya kita buat kelas untuk `strategy` diskonnya seperti tampilan berikut :

```
"""function dedicated to On Sale Discount"""
def on_sale_discount(order):

    return order.price * 0.25 + 20

"""function dedicated to 20 % discount"""
def twenty_percent_discount(order):

    return order.price * 0.20
```

Figure 107 Contoh Penerapan Strategy Pattern (2)

Pada kode tersebut kita membuat dua buah strategy yang berupa diskon ketika terjadinya pembelian pada saat SALE (yang biasa disukai kebanyak perempuan) dan pada saat tidak ada SALE, atau diskon biasa. Hal ini tentunya dapat membantu kita untuk membuat lebih banyak strategy diskon dan tidak perlu mengubah kodennya. Namun, gimana hasilnya? Perhatikan contoh hasil kode berikut :

```
>>> hargaLaptop = 30000000
>>> print(Item(hargaLaptop))
Price: 30000000, price after discount: 30000000
>>> print(Item(hargaLaptop, discount_strategy = twenty_percent_discount))
Price: 30000000, price after discount: 24000000.0
>>> print(Item(hargaLaptop, discount_strategy = on_sale_discount))
Price: 30000000, price after discount: 22499980.0
```

Figure 108 Hasil Penerapan Strategy Pattern

Dari hasil tersebut terlihat dengan jelas penerapan strategy yang kita lakukan. Nah bagaimana kalau semisalnya si penjual merasa rugi dan terakhir dia ganti pada nilai diskonnya + ada biaya tambahan? Apakah harus mengubah kode seluruhnya? Tentunya tidak. Tapi, gimana caranya? Silahkan kalian explorer lebih lanjut.

Sumber : [GeeksforGeeks](#).

Nah kita sudah mengenai ada 3 jenis pattern yang kita buat. Nah bagaimana kita menyelesaikan permasalahan studi kasus kita? Yuk kita balik ke studi kasus!

SOLUTION

Setelah kita pelajari langkah-langkah yang ada diatas, nah kita terlebih dahulu harus tahu dulu mengenai jenis design pattern yang ingin kita terapkan untuk studi kasus kita? Dan dari 3 jenis yang telah dijelaskan sebelumnya, jenis yang kita pakai adalah jenis “Observer Pattern”. Mengapa ini? Karena, pemberitahuan setiap data yang ingin diproses setiap adanya terjadi perubahan, yaitu pengecekan keterbaruan dan pengecekan jumlah buku yang tersedia.

Nah gimana cara penerapannya ke dalam program? Yuk kita bahas 1 per 1!

INSTRUCTION

1. Tahap awal untuk menyelesaikan masalah tersebut kita harus membuat kelas utamanya terlebih dahulu :

```
class BUKU:
    def __init__(self, nama, tahun, jlh):
        self._nama = nama
        self._tahun = tahun
        self._jlh = jlh

    def cetakBuku(self):
        hasil = "Nama Buku = {}\n".format(self._nama)
        hasil += "Tahun Buku = {}\n".format(self._tahun)
        hasil += "Jumlah Buku = {}\n".format(self._jlh)
        return hasil
```

2. Kita tambahkan proses pengecekan tahun pada proses yang ada dengan menambahkan kode :

```
class cekTahun(BUKU):
    def __init__(self, proses):
        self._proses = proses
    def cetakBuku(self):
        thn = 2021
        hasil = self._proses.cetak()
        hasil += "Buku termasuk buku {}\\n".format("Baru" if (thn - self._tahun) > 5 else "Lama")
        return hasil
```

3. Kita tambahkan juga proses pengecekan jumlah buku dengan menambahkan kode :

```
class cekStok(BUKU):
    def __init__(self, proses):
        self._proses = proses
    def cetakBuku(self):
        hasil = self._proses.cetak()
        hasil += "Perlu {}\\n".format("Mencari Buku" if self._jh < 3 else "Mencari Pembaca")
        return hasil
```

4. Setelah semua sudah selesai, kita tambahkan kode dibawah ini sebagai fungsi main :

```
if __name__ == '__main__':
    buku1 = BUKU("Pemrograman Python", 2019, 3)
    denganCekTahun = cekTahun(buku1)
    denganCekJumlah = cekStok(buku1)
    denganCekSemua = cekStok(cekTahun(buku1))

    print("Awal :\\n", buku1.cetakBuku())
    print()
    print("Dengan Cek Tahun:\\n", denganCekTahun.cetakBuku())
    print()
    print("Dengan Cek Jumlah:\\n", denganCekJumlah.cetakBuku())
```

5. Hasil dari kode tersebut adalah :

```
Awal :
  Nama Buku = Pemrograman Python
  Tahun Buku = 2019
  Jumlah Buku = 3

  Dengan Cek Tahun:
  Nama Buku = Pemrograman Python
  Tahun Buku = 2019
  Jumlah Buku = 3
  Buku termasuk buku Lama

  Dengan Cek Jumlah:
  Nama Buku = Pemrograman Python
  Tahun Buku = 2019
  Jumlah Buku = 3
  Perlu Mencari Pembaca
```

Figure 109 Hasil Studi Kasus

EXERCISE

EXERCISE OBJECTIVES

Pada latihan berikut ini mahasiswa diharapkan dapat melakukan :

- Melakukan analisis masalah terhadap studi kasus yang diberikan.
- Mencari solusi atas masalah yang diberikan.
- Menerjemahkan masalah tersebut ke dalam bentuk kode pemrograman.

TASK 1:

Pada sebuah kelas di Kampus Mikroskil terdapat mahasiswa dan dosen pada kelas tersebut. Mahasiswa tersebut tentunya memiliki data berupa (*bagian ini kalian cari tahu sendiri*). Sedangkan dosen memiliki data berupa NIP, Nama, jabatan, jenis kelamin, dan no HP. Pada suatu acara, seorang mahasiswa ditujukan untuk melakukan rekap absensi dari acara tersebut. Absensi tersebut nantinya harus bisa dicetak serta bisa dihitung kira-kira berapa banyak jumlah pengunjung pada acara tersebut. Bisakah kalian membantu mahasiswa tersebut?

Ketentuan yang dilakukan pada task-01 ini adalah :

- a. Semua proses dilakukan di dalam kelas.
- b. Pada proses terdapat 1 kelas utama, dan kelas absensi serta kelas main.
- c. Nama kelas utama tersebut diharuskan menggunakan nama kalian.
- d. Untuk pesan keluaran bisa disesuaikan dengan kebutuhan kalian sendiri.
- e. Untuk penentuan jenis design pattern yang digunakan disesuaikan kembali berdasarkan pemahaman kalian terhadap kasus.

UNIT 5

DESIGN PATTERN 1 (2)

UNIT OVERVIEW

Pada pembelajaran kali ini kita masih berfokus kepada bagaimana cara untuk memaksimalkan kode yang kita buat dengan tidak melakukan pola penulisan kode yang berulang. Hal ini tentunya bertujuan supaya kode yang kita buat tidak menjadi lebih baik dan sesuai dengan tujuan utama.

UNIT OBJECTIVES

Adapun capaian yang akan kita dapatkan pada modul ini, yaitu :

- Cara memahami penerapan singleton pattern
- Cara memahami penerapan template pattern

UNIT CONTENTS

Lesson 4: The Singleton Pattern	84-
87	
Lesson 5: The Template Pattern.....	87-
89	

PRE LAB

Sebelum masuk ke dalam lab pertama ini, terlebih dahulu kita coba jawab beberapa pertanyaan ini berdasarkan hasil penjelasan pada modul teori!

QUESTION

1. Dari ketiga jenis yang telah kita pelajari sebelumnya, menurut kalian apa tujuan kita menerapkan strategi pattern?
2. Apa yang dimaksud dengan decorator pattern?
3. Dari permasalahan yang ada dibawah ini, kira pattern mana saja yang sesuai dengan permasalahan tersebut :
 - Pembuatan skripsi dengan menggunakan dua metode
 - Pembayaran pada kasir
 - Melakukan pengecekan beberapa jam dunia

CONTENT LESSON

CASE STUDY / PROJECT

Pada perpustakaan “Home Learning is Best” seperti yang kita ketahui terjadinya perubahan manajemen yang membuat penambahan fitur untuk bagian pengecekan keterbaruan buku, dan pengecekan jumlah buku yang tersedia. Dari hasil pengecekan jumlah buku, pihak pemilik perpustakaan maunya adalah kita mesti mencetak terlebih dahulu tahun terbit buku baru melakukan pengecekan keterbaruan buku. Sedangkan, pihak manajemen menginginkan bahwa kita harus mengecek keterbaruan buku, baru mencetak tahun terbit buku.

Nah gimana donk? Siapa yang harus kita ikuti? Apakah kita ikuti keduanya atau salah satu saja? Sebelum kita jawab, yuk kita cari tahu terlebih dahulu akar dari permasalahannya.

IDENTIFICATION CONCEPT OF PROBLEM / PROJECT

Setelah kita baca kembali studi kasus tersebut, ternyata permasalahannya karena adanya perbedaan pendapat dari pemilik perusahaan dengan pihak manajemen. Jadi gimana? Apakah kita lakukan keduanya? Jika iya, memangnya ada cara untuk melakukan kedua proses tersebut?

Yuk kita pelajari terlebih dahulu proses-proses yang ada!

LESSON 4: THE SINGLETON PATTERN

LESSON 4: OVERVIEW

The singleton pattern merupakan sebuah pola yang memungkinkan kita untuk mendeklarasikan sebuah objek secara global untuk sub-class yang ada. Hal ini bertujuan supaya proses yang dilakukan saling berhubungan dan tidak perlu membuat tempat penyimpanan baru untuk 1 buah hal yang sama.

Sebagai contoh, ketika kita mengakses sebuah website ada 1 cara yang dapat membuat kita tetap bisa login meskipun kita berpindah halaman web, yaitu dengan menggunakan Cookies (cache). Hal ini tentunya dapat membuat kita tidak perlu lagi login untuk menuju ke halaman tertentu.

Sebagai contoh perhatikan kode berikut :

```
class cookies:  
    _COOKIES = []  
    def __init__(self):  
        self.__dict__ = self._COOKIES
```

Figure 110 Contoh Singleton Pattern (1)

Pada kode tersebut tentunya kita terlebih dahulu membuat sebuah kelas yang nantinya akan digunakan sebagai tempat penyimpanan cookies (cache) yang di website. Setelah itu, kita buat websitenya seperti kode berikut :

```
class websiteMIKA(cookies):  
    def __init__(self, **data):  
        cookies.__init__(self)  
        self._COOKIES.update(data)  
    def cetakCookies(self):  
        tmp = self._COOKIES  
        for name, data in tmp.items():  
            print(name, ":", data)  
    def cekData(self, data):  
        try:  
            return self._COOKIES[data]  
        except Exception:  
            return False
```

Figure 111 Contoh Singleton Pattern (2)

Seperti yang terlihat pada kelas tersebut terdapat 3 buah method yang terdiri dari method inisialisasi untuk proses awal, cetakCookies untuk mencetak isi dari cookies yang ada serta mengecek data yang nantinya akan kita gunakan sebagai pengecekan apakah pengguna sebelumnya sudah melakukan login atau belum.

Dan untuk menjalankan kode tersebut, kita ketikkan kelas main berikut :

```
if __name__ == "__main__":
    login = websiteMIKA(login=True, uid="apriyanto.halim")
    print("Cookies Awal")
    login.cetakCookies()

    if(login.cekData("login")):
        home = websiteMIKA(halaman="home")
        print("\nCookies Berikutnya")
        home.cetakCookies()

    if(login.cekData("login")):
        profile = websiteMIKA(halaman="profile")
        print("\nCookies Berikutnya")
        profile.cetakCookies()
```

Figure 112 Contoh Singleton Pattern (3)

Dari kelas main tersebut terlihat pada saat awal, ada seorang pengguna yang ingin melakukan login ke dalam website MIKA. Pengguna tersebut tentunya memberikan dua buah data berupa status login dan usernamenya. Setelah itu, pengguna tersebut ingin mengunjungi halaman home, tapi terlebih dahulu dilakukan pengecekan, apakah sudah pernah login sebelumnya atau belum. Dan dilanjutkan untuk masuk ke dalam halaman profile, sehingga untuk hasil kode tersebut sebagai berikut :

```
Cookies Awal
login : True
uid : apriyanto.halim

Cookies Berikutnya
login : True
uid : apriyanto.halim
halaman : home

Cookies Berikutnya
login : True
uid : apriyanto.halim
halaman : profile
```

Figure 113 Hasil Singleton Pattern

Gimana tentunya hal ini dapat membantu kalian yang ingin membuat website dengan memanfaat cookies yang ada untuk melakukan pengecekan login terhadap pengguna. Namun, bagaimana jika tahapan yang dilakukan berbeda-beda? Seperti ada website yang Ketika proses pendaftaran selesai langsung menuju ke halaman home, tapi ada juga website yang setelah selesai melakukan pendaftaran diarahkan ke halaman login? Dapatkah hal tersebut ditangani dengan menggunakan singleton pattern? Jawabannya adalah tidak. Namun, ada pattern lainnya yang dapat kita gunakan. Yuk kita pelajari pelajaran berikutnya!

LESSON 5: THE TEMPLATE PATTERN

LESSON 5: OVERVIEW

The template pattern merupakan kerangka algoritma yang ada superclass. Namun, untuk subclassnya dapat melakukan perubahan terhadap langkah-langkah yang dilakukan. Sebagai contoh, saya ada sebuah tindakan. Dimana tindakan yang saya lakukan mencari angka pada sekumpulan angka yang ada. Adapun langkah-langkah yang saya lakukan adalah :

1. Melakukan proses pengurutan. Dimana seperti yang kita ketahui ada beberapa cara yang bisa kita lakukan untuk melakukan sorting, yaitu dengan fungsi list yang ada, menggunakan teknik algoritma bubble sort, maupun dengan menggunakan Insertion Sort.
2. Setelah kita melakukan proses pengurutan, baru kita melakukan pencarian data tersebut.

Tujuan dari proses ini adalah menguji kira-kira algoritma mana yang lebih cepat dibandingkan dengan algoritma lainnya dari segi waktu. Oleh karena itu, mari kita buat terlebih dahulu kelas utamanya sebagai berikut :

```
import time
class pencarian:
    def __init__(self, arr, data):
        self.arr = arr
        self.data = data
    def pengurutan(self):
        return self.arr
    def pencarian (self):
        for i in range(len(self.arr)):
            if(self.arr[i] == self.data):
                return i
        return -1
    def cetakArr(self):
        return self.arr
    def main(self):
        hsl = self.pengurutan()
        ind = self.pencarian()
        return ind
```

Figure 114 Contoh Template Pattern (1)

Pada kode tersebut terlebih dahulu saya kenali beberapa tindakan, diantaranya inisialisasi nilai awal, untuk memasukkan data ke dalam kelas. Selanjutnya ada fungsi awal pengurutan, pencarian, pencetakan data dan fungsi main (fungsi utama) yang saya gunakan. Saya juga ada melakukan import time, yang nanti saya gunakan sebagai perhitungan waktu mulai dan selesai proses untuk masing-masing pengurutan yang dilakukan. Setelah itu selesai, selanjutnya saya membuat kelas untuk melakukan fungsi sort pada list python dengan tampilan berikut :

```
class fungsiSort(pencarian):
    def pengurutan(self):
        return self.arr.sort()
```

Figure 115 Contoh Template Pattern (2)

Kelihatan simple bukan. Karena, kita hanya perlu memanggil fungsi sort yang merupakan fungsi default yang ada pada Python list. Setelah itu kita buat kode untuk BubbleSort seperti tampilan kode berikut :

```
class bubbleSort(pencarian):
    def pengurutan(self):
        for i in range(len(self.arr) - 1):
            for j in range(i+1, len(self.arr)):
                if(self.arr[i] > self.arr[j]):
                    tmp = self.arr[i]
                    self.arr[i] = self.arr[j]
                    self.arr[j] = tmp
        return self.arr
```

Figure 116 Contoh Template Pattern (3)

Seperti yang kita ketahui bahwa proses pengurutan menggunakan bubblesort terlihat sesederhana tersebut. Dan terakhir saya tambahkan kode pengurutan dengan menggunakan InsertionSort seperti tampilan berikut :

```
class insertionSort(pencarian):
    def pengurutan(self):
        for i in range(1, len(self.arr)):
            key_item = self.arr[i]
            j = i - 1
            while j >= 0 and self.arr[j] > key_item:
                self.arr[j + 1] = self.arr[j]
                j -= 1
            self.arr[j + 1] = key_item
        return self.arr
```

Figure 117 Contoh Template Pattern (4)

Untuk kode tersebut saya ambil dari website realpython.com yang merupakan website yang berisi kode-kode program dalam bahasa pemrograman Python. Setelah semuanya selesai, selanjutnya yuk kita buat kelas mainnya seperti tampilan berikut :

```
if __name__ == "__main__":
    data = [10, 11, 6, 19, -3, 5, 9, 100, 50, 49, 101, 200, 67, 891, 892, 1000, 60, -37]
    cari = 11

    #Fungsi Sort
    start_time = time.time()
    algo1 = fungsiSort(data, cari)
    print(f"Angka {cari} ada diurutan ke-{algo1.main()}")
    print(f"Hasil pengurutan = {algo1.cetakArr()}")
    print("Hasil Fungsi sort = %s seconds\n" % (time.time() - start_time))

    #Fungsi BubbleSort
    start_time = time.time()
    algo2 = bubbleSort(data, cari)
    print(f"Angka {cari} ada diurutkan ke-{algo2.main()}")
    print(f"Hasil pengurutan = {algo2.cetakArr()}")
    print("Hasil Fungsi BubbleSort = %s seconds\n" % (time.time() - start_time))

    #Fungsi InsertionSort
    start_time = time.time()
    algo3 = insertionSort(data, cari)
    print(f"Angka {cari} ada diurutkan ke-{algo3.main()}")
    print(f"Hasil pengurutan = {algo3.cetakArr()}")
    print("Hasil Fungsi InsertionSort = %s seconds\n" % (time.time() - start_time))
```

Figure 118 Contoh Template Pattern (5)

Dari kode tersebut terlihat saya ada melakukan inisialisasi array awal yang memiliki angka acak, yang nantinya angka acak ini akan saya gunakan untuk mengurutkan data tersebut. Nah setelah selesai, kira-kira gimana ya hasilnya? Untuk hasilnya dapat dilihat pada tampilan berikut :

```
Angka 11 ada diurutan ke-6
Hasil pengurutan = [-37, -3, 5, 6, 9, 10, 11, 19, 49, 50, 60, 67, 100, 101, 200, 891, 892, 1000]
Hasil Fungsi sort = 0.0 seconds

Angka 11 ada diurutkan ke-6
Hasil pengurutan = [-37, -3, 5, 6, 9, 10, 11, 19, 49, 50, 60, 67, 100, 101, 200, 891, 892, 1000]
Hasil Fungsi BubbleSort = 0.0 seconds

Angka 11 ada diurutkan ke-6
Hasil pengurutan = [-37, -3, 5, 6, 9, 10, 11, 19, 49, 50, 60, 67, 100, 101, 200, 891, 892, 1000]
Hasil Fungsi InsertionSort = 0.0 seconds
```

Figure 119 Hasil Template Pattern

Gimana? Pastinya kalian akan lebih mudah untuk memahami penggunaan template pattern ini. Jangan lupa untuk melakukan explorer ya.

Mari kita kembali ke studi kasus awal kita!

SOLUTION

Setelah kita pelajari beberapa pelajaran yang sudah ada, mari kita kembali ke studi kasus kita. Dimana distudi kasus kita si pihak manajemen menginginkan untuk mencetak keterbaruan buku baru tahun, sedangkan pemilih ingin mencetak tahun baru keterbaruan buku. Sebelum kita mengeksekusinya, kita harus tahu dulu ini termasuk ke dalam design pattern yang mana. Dan jawaban yang sesuai adalah termasuk ke bagian Template Pattern. Karena, sudah ada template awal, kita tinggal sesuaikan kembali. Oleh karena itu, mari kita rombak sedikit kode kita sebelumnya supaya sesuai dengan yang diharapkan!

INSTRUCTION

1. Kita ubah sedikit kode untuk cekTahun sebelumnya menjadi seperti berikut :

```
class cekTahun(BUKU):  
    def __init__(self, proses):  
        self._proses = proses  
    def cetakBuku(self):  
        pass
```

2. Setelah itu, kita sesuaikan permintaan dari pemilik :

```
class cekTahunPemilik(cekTahun):  
    def cetakBuku(self):  
        thn = 2021  
        hasil = self._proses.cetakBuku()  
        hasil += "Buku terbit pada tahun {} ".format(self._proses.tahun)  
        hasil += " sehingga, termasuk buku {} \n".format("Baru" if (thn - self._proses.tahun) > 5 else "Lama")  
        return hasil
```

3. Setelah itu, kita sesuaikan permintaan dari manajemen :

```
class cekTahunManajemen(cekTahun):  
    def cetakBuku(self):  
        thn = 2021  
        hasil = self._proses.cetakBuku()  
        hasil += "Buku termasuk buku {} ".format("Baru" if (thn - self._proses.tahun) > 5 else "Lama")  
        hasil += " dimana buku terbit pada tahun {} \n".format(self._proses.tahun)  
        return hasil
```

4. Setelah sudah semua, berikut kita ubah sedikit pada bagian main :

```
if __name__ == '__main__':  
    buku1 = BUKU("Pemrograman Python", 2019, 3)  
    denganPemilik = cekTahunPemilik(buku1)  
    denganManajemen = cekTahunManajemen(buku1)  
    denganCekJumlah = cekStok(buku1)  
    denganCekSemua = cekStok(cekTahun(buku1))  
  
    print("Awal :\n", buku1.cetakBuku())  
    print()  
    print("Dengan Cek Tahun Pemilik:\n", denganPemilik.cetakBuku())  
    print()
```

```
print("Dengan Cek Tahun Manajemen:\n", denganManajemen.cetakBuku())
print()
print("Dengan Cek Jumlah:\n", denganCekJumlah.cetakBuku())
```

5. Setelah semuanya sudah selesai, berikut hasilnya :

Awal :
Nama Buku = Pemrograman Python
Tahun Buku = 2019
Jumlah Buku = 3

Dengan Cek Tahun Pemilik:
Nama Buku = Pemrograman Python
Tahun Buku = 2019
Jumlah Buku = 3
Buku terbit pada tahun 2019 sehingga, termasuk buku Lama

Dengan Cek Tahun Manajemen:
Nama Buku = Pemrograman Python
Tahun Buku = 2019
Jumlah Buku = 3
Buku termasuk buku Lama dimana buku terbit pada tahun 2019

Dengan Cek Jumlah:
Nama Buku = Pemrograman Python
Tahun Buku = 2019
Jumlah Buku = 3
Perlu Mencari Pembaca

Figure 120 Hasil Studi Kasus

EXERCISE

EXERCISE OBJECTIVES

Pada latihan berikut ini mahasiswa diharapkan dapat melakukan :

- Melakukan analisis masalah terhadap studi kasus yang diberikan.
- Mencari solusi atas masalah yang diberikan.
- Menerjemahkan masalah tersebut ke dalam bentuk kode pemrograman.

TASK 1:

Pada sebuah kelas di Kampus Mikroskil terdapat mahasiswa dan dosen pada kelas tersebut. Mahasiswa tersebut tentunya memiliki data berupa (*bagian ini kalian cari tahu sendiri*). Sedangkan dosen memiliki data berupa NIP, Nama, jabatan, jenis kelamin, dan no HP. Pada suatu acara, seorang mahasiswa ditujukan untuk melakukan rekap absensi dari acara tersebut. Absensi tersebut nantinya harus bisa dicetak serta bisa dihitung kira-kira berapa banyak jumlah pengunjung pada acara tersebut. Data absensi tersebut diharapkan juga disimpan semua ke dalam sebuah tempat penyimpanan sementara, sehingga nanti bisa memudahkan pada saat proses pencarian maupun proses pengecekan data. Bisakah kalian membantu mahasiswa tersebut?

Ketentuan yang dilakukan pada task-01 ini adalah :

- a. Semua proses dilakukan di dalam kelas.
- b. Pada proses terdapat 1 kelas utama, dan kelas absensi serta kelas main.
- c. Nama kelas utama tersebut diharuskan menggunakan nama kalian.
- d. Untuk pesan keluaran bisa disesuaikan dengan kebutuhan kalian sendiri.
- e. Untuk penentuan jenis design pattern yang digunakan disesuaikan kembali berdasarkan pemahaman kalian terhadap kasus.
- f. Bisa menggunakan kode sebelumnya.

UNIT 6

DESIGN PATTERN 2 (1)

UNIT OVERVIEW

Pada pembelajaran kali ini, masih dengan topik yang sama mengenai design pattern. Dimana tujuan utama adalah mengurangi beban kerja dari kode dan bagaimana cara kita bisa melakukan perancangan terhadap kode yang kita buat, sesuai dengan apa yang kita harapkan.

UNIT OBJECTIVES

Adapun capaian yang akan kita dapatkan pada modul ini, yaitu :

- Memahami penggunaan Adapter Pattern
- Memahami penggunaan Façade Pattern
- Memahami penggunaan Command Pattern

UNIT CONTENTS

Lesson 1: The Adapter Pattern.....	94-
96	
Lesson 2: The Façade Pattern.....	96-
98	
Lesson 3: The Command Pattern	98-
100	

PRE LAB

Sebelum masuk ke dalam lab pertama ini, terlebih dahulu kita coba jawab beberapa pertanyaan ini berdasarkan hasil penjelasan pada modul teori!

QUESTION

1. Menurut kalian, alasan apa yang mengharuskan kita menggunakan singleton pattern?
2. Sebutkan salah satu contoh penerapan dari template pattern?
3. Dari daftar proses, manakah yang bisa diselesaikan dengan menggunakan template pattern yang telah kita pelajari pada pertemuan-10 :
 - Proses pembuatan website dengan bootstraps.
 - Proses pengecekan jumlah stok yang masuk.
 - Proses pembuatan kartu undangan.

CONTENT LESSON

CASE STUDY / PROJECT

Pada perpustakaan “Home Learning is Best” telah diselesaikan permasalahan mengenai perbedaan pendapat antara pihak manajemen dan pihak pemilik. Namun, ketika diberikan kepada pihak project manajer ada kendala mengenai si project manajer tidak mengetahui asal dari saran ini siapa-siapa. Mengenai tampilan pertama ini saran dari siapa tidak dikenal. Semua itu hanya bisa dilihat dari kode secara langsung. Jadi, project manajer mengharapkan untuk membuat sebuah cara terkait siapa pengusul dari saran tersebut. Supaya lebih mudah didalam pemrosesan, maka yang pertama tersebut merupakan proses yang diminta oleh pemilik dan yang kedua merupakan proses yang diminta oleh pihak manajemen.

Waduh? Kita kelupaan ya membuat nama pengusulnya. Namun, tidak apa-apa kita bisa melakukannya pada pertemuan kali ini dengan pattern yang ada. Namun, pattern yang gimana ya? Yuk kita cari tahu!

IDENTIFICATION CONCEPT OF PROBLEM / PROJECT

Sebelum kita lanjut ke solusi dari studi kasus tersebut, mari kita pelajari kembali terkait studi kasus tersebut. Permasalahannya lumayan simple, dimana kita hanya perlu memberikan tanda pengenal mengenai siapa pengusul desain tersebut. Namun, gimana caranya? Apakah kita merombak kode kita lagi? Kalau kita rombak, apakah akan ada yang perlu kita tambahkan atau kita kurangi?

Sebelum kita menjelaskan solusi dari permasalahan tersebut, mari kita simak dulu beberapa penjelasan untuk membantu kita dalam menyelesaikan permasalahan yang terdapat dalam studi kasus tersebut !

LESSON 1: THE ADAPTER PATTERN

LESSON 1: OVERVIEW

The adapter pattern merupakan sebuah pola yang bisa digunakan untuk menggabungkan beberapa objek yang berbeda ke dalam sebuah objek. Hal ini bertujuan supaya kita tidak perlu lagi membuat 2 tempat yang berbeda untuk menyimpan dua objek yang hampir sama. Sebagai contoh, ketika kita membuat sebuah aplikasi tidak dapat dipungkiri apabila kita menggunakan lebih dari 1 buah database. Hal ini tentunya harus kita pastikan kembali terhadap gimana cara pemanggilannya dan gimana cara akses serta prosesnya. Perhatikan contoh kelas utama berikut :

```

class database:
    def __init__(self, host, database, username, password):
        self._host = host
        self._database = database
        self._username = username
        self._password = password

    def koneksi(self):
        print("Berhasil terhubung ke server")
        print(f"Host      : { self._host }")
        print(f"Database  : { self._database }")
        print(f"Username  : { self._username }")
        print(f"Password  : { self._password }")

    def excuteQuery(self, perintah):
        print(f"Perintah yang dimasukkan adalah : {perintah}")

```

Figure 121 Contoh Kode Adapter Pattern (1)

Kelas tersebut merupakan kelas utama yang biasa kita gunakan untuk melakukan koneksi ke dalam database. Pada kelas tersebut terdapat 3 buah method utama, yaitu method init yang merupakan method yang digunakan untuk melakukan string koneksi, method koneksi yang digunakan untuk terhubung ke dalam database, serta method executeQuery yang berfungsi untuk melaksanakan perintah query yang ada di dalam database. Nah apabila kita membuat sebuah program dan ingin mengkases kelas tersebut tapi tanpa mengubah isi di dalam kelas. Gimana ya caranya? Nah hal tersebut bisa kalian lakukan dengan menggunakan Adapter Pattern. Mari kita lihat kode adapter pattern berikut :

```

class AdapterDatabase:
    #stringKoneksi = host database username password
    def __init__(self, stringKoneksi):
        tmp = stringKoneksi.split(" ")
        self.db = database(tmp[0], tmp[1], tmp[2], tmp[3])
        self.db.koneksi()
    def selectTable(self, namaTable):
        perintah = "Select * from ()".format(namaTable)
        self.db.excuteQuery(perintah)

```

Figure 122 Contoh Kode Adapter Pattern (2)

Dari kelas adapter ini kita harus menyesuaikan dengan kebutuhan dari penggunaan dari program yang kita buat serta menghubungkan dengan database. Dari kode tersebut ada 2 method yang dibuat, yaitu

method inisialisasi yang nantinya akan digunakan untuk koneksi dan method selectTable yang akan digunakan untuk mengakses tabel yang ada di dalam database. Terus, setelah kita buat ini, gimana cara mengaksesnya? Tentunya sangat simple, kita hanya perlu mengakses kelas adaptor saja tidak perlu lagi mengakses database. Seperti tampilan berikut ini :

```
if __name__ == "__main__":
    Host = "127.0.0.1"
    Database = "pelaporanUang"
    Username = "apriyanto"
    Password = "*****"
    koneksi = "{} {} {} {}".format(Host, Database, Username, Password)
    Tabel = "Pengeluaran"
    adapter = AdapterDatabase(koneksi)
    adapter.selectTable(Tabel)
```

Figure 123 Contoh Kode Adapter Pattern (3)

Hasil dari kode tersebut dapat dilihat pada gambar dibawah ini :

```
Berhasil terhubung ke server
Host      : 127.0.0.1
Database  : pelaporanUang
Username   : apriyanto
Password   : *****
Perintah yang dimasukkan adalah : Select * from Pengeluaran
```

Figure 124 Hasil Kode Adapter Pattern

Terlihat dengan jelas kita bisa mengakses database tersebut tanpa perlu kesulitan dalam mengaksesnya. Hal ini tentunya dapat membantu kita dalam membuat kode yang ada.

Setelah kita mempelajari ini, selanjutnya kita akan mempelajari mengenai Façade Pattern. Gimana prosesnya dan gimana juga penerapannya. Mari kita masuk ke pelajaran selanjutnya ... !!!

LESSON 2: THE FAÇADE PATTERN

LESSON 2: OVERVIEW

The façade pattern merupakan sebuah pola yang memudahkan kita dalam mengakses setiap kelas atau setiap proses yang ada. Hal ini bertujuan supaya pengguna tidak perlu lagi mengetahui ketika dia memesan makanan, apa saja yang harus dilakukan supaya dia bisa mendapatkan makanannya.

Sebagai contoh perhatikan kode berikut :

```
class kasir:  
    def __init__(self, pelanggan):  
        self.pelanggan = pelanggan  
    def minta(self):  
        print("Kasir diminta proses bon untuk pelanggan {}".format(self.pelanggan))  
    def proses(self):  
        print("Kasir mencetak bon untuk pelanggan {}".format(self.pelanggan))  
class pelayan:  
    def __init__(self, pelanggan):  
        self.pelanggan = pelanggan  
    def pemesanan(self):  
        print("Pelayan mencetak pesanan pelanggan {}".format(self.pelanggan))  
    def antarPesan(self):  
        print("Pelayan mengantar pesanan pelanggan {}".format(self.pelanggan))  
    def mintaBon(self):  
        print("Pelayan meminta bon sama kasir")  
    def antarBon(self):  
        print("Pelayan mengantar bon pelanggan {}".format(self.pelanggan))
```

Figure 125 Contoh Kode Facade Pattern (1)

Dari kode tersebut ketika pelanggan datang pada sebuah restoran maka kasir dan pelayan harus melakukan semua proses tersebut. Namun, kita harus dari proses mana? Mungkin kalau sudah sering dilakukan, tidak perlu lagi melihat sudah tahu. Terus, kalau ada pegawai baru gimana? Tentunya kita harus mendaftarkan kembali, apa saja yang harus dilakukan dan dimulai dari mana. Sebagai contoh perhatikan kode berikut ini :

```
class restoran:  
    def __init__(self, pelanggan):  
        self.kasir = kasir(pelanggan)  
        self.pelayan = pelayan(pelanggan)  
    def prosesPemesanan(self):  
        self.pelayan.pemesanan()  
        self.pelayan.antarPesan()  
        self.pelayan.mintaBon()  
        self.kasir.minta()  
        self.kasir.proses()  
        self.pelayan.antarBon()
```

Figure 126 Contoh Kode Facade Pattern (2)

Pada kelas tersebut terdapat 2 method, yaitu kelas utama yang melakukan inisialisasi terhadap pemanggilan dua kelas sebelumnya yang nantinya akan digunakan pada saat proses pemesanan makanan. Setelah itu, ada juga method untuk proses makanan. Dimana method ini berisi tindakan yang terstruktur apa saja yang harus dilakukan. Sehingga, pelayan dan kasir yang baru tidak perlu

lagi menghapal setiap tindakan, hanya perlu disesuaikan dengan ketentuan yang ada. Nah gimana dengan proses utamanya? Apakah perlu diberitahukan lagi kepada kasir dan pelayan? Tentunya tidak lagi. Kita hanya perlu memberitahukan kepada pihak restoran saja seperti contoh kode berikut :

```
if __name__ == "__main__":
    pelanggan = "Budi"
    Resto_Apri = restoran(pelanggan)
    Resto_Apri.prosesPemesanan()
```

Figure 127 Contoh Kode Facade Pattern (3)

Nah gimana? Lebih mudah bukan. Kita tidak perlu lagi memanggil kelas lainnya. Tentunya hal ini dapat membantu kita untuk lebih mudah melakukan proses tanpa perlu tahu proses apa saja yang ada dibagian belakangnya. Untuk hasil kode dari atas sebagai berikut :

```
Pelayan mencetak pesanan pelanggan Budi
Pelayan mengantar pesanan pelanggan Budi
Pelayan meminta bon sama kasir
Kasir diminta proses bon untuk pelanggan Budi
Kasir mencetak bon untuk pelanggan Budi
Pelayan mengantar bon pelanggan Budi
```

Figure 128 Hasil Kode Facade Pattern

Gimana? Sesuai dengan apa yang diharapkan bukan. Tentunya hal ini diharapkan bisa membantu kalian dapat membuat kode yang lebih mudah untuk dibuat.

Setelah ini kita akan kembali membahas mengenai Command Pattern. Gimana proses dan untuk apa digunakan? Supaya tidak bertanya-tanya lagi, mari kita lanjut ke pelajaran selanjutnya!

LESSON 3: THE COMMAND PATTERN

LESSON 3: OVERVIEW

The command pattern merupakan sebuah pola yang digunakan untuk memanggil sebuah objek yang digunakan untuk merangkum semua objek yang digunakan. Objek tersebut nantinya dapat berupa nama objek, method yang dimiliki objek dan bisa juga parameter yang terdapat pada objek. Dari objek tersebut dapat mewakili beberapa proses yang ada. Sehingga, kita tidak perlu lagi membuat kata yang berbeda untuk nama proses yang sama. Sebagai contoh, ketika menyalakan lampu pastinya kita tidak perlu mengetahui bagaimana lampu tersebut bisa menyala. Kita cuman tahu, bahwa ketika saya menekan tombol ON maka lampu akan menyala.

Terus, hal ini akan kita terapkan pada kode program berikut :

```
from abc import ABC, abstractmethod

class proses(ABC):
    @abstractmethod
    def execute(): pass

class switchOn(proses):
    def execute(self):
        print("Lampu Nyala")

class switchOff(proses):
    def execute(self):
        print("Lampu Mati")
```

Figure 129 Contoh Kode Command Pattern (1)

Pada kode tersebut terlebih dahulu saya membuat sebuah abstrak kelas, yang nanti akan digunakan sebagai panduan untuk pembuatan kelas-kelas perintah. Setelah itu, dibuat 2 kelas perintah yang berupa kelas switchOn dan kelas switchOff. Dua kelas ini berisi perintah untuk menyalakan dan mematikan lampu. Namun, ada satu proses dimana mengharuskan saya untuk mengganti semua nama menjadi proses maka saya perlu menggunakan command pattern seperti yang terlihat pada kelas berikut :

```
class switch:
    def __init__(self, cmd):
        self.cmd = cmd
    def Proses(self):
        self.cmd.execute()
```

Figure 130 Contoh Kode Command Pattern (2)

Dari kode tersebut, nantinya akan melakukan 2 hal, yaitu menganti nama perintah dengan switch setelah itu mengganti nama method execute menjadi proses. Tentunya hal ini membuat kita tidak perlu lagi mengubah kode secara keseluruhan untuk kelas perintah yang telah kita buat sebelumnya.

Namun, gimana penerapannya pada main kelas? Perhatikan kode berikut :

```
if __name__ == "__main__":
    on = switchOn()
    sw = switch(on)
    sw.Proses()

    print("\n")

    off = switchOff()
    sw = switch(off)
    sw.Proses()
```

Figure 131 Contoh Kode Command Pattern (3)

Gimana? Lebih tetap sama menggunakan nama method proses bukan? Terus, apakah pada saat kita jalankan program, program tidak error? Kita lihat hasil berikut :

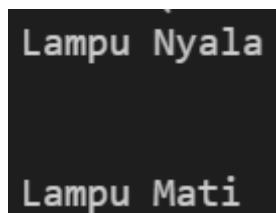


Figure 132 Hasil Kode Command Pattern

Nah gimana? Tidak ada error bukan. Inilah kegunaan dari penggunaan Command Pattern pada Design Pattern yang telah kita pelajari.

Setelah kita belajar dari ketiga pattern tersebut, mari kita selesaikan studi kasus tersebut!

SOLUTION

Setelah kita baca kembali dari studi kasus tersebut kita menemukan permasalahan yang berupa memberikan penamaan untuk setiap proses yang terjadi. Namun, sebelum kesana kita harus tahu terlebih dahulu cara apa yang sesuai dengan permasalahan tersebut. Dan berdasarkan hasil analisis permasalahan tersebut lebih sesuai apabila diselesaikan dengan menggunakan Adapter Pattern. Karena, proses yang dilakukan hampir sama, hanya perlu memberikan keterangan terkait siapa yang membuat saja. Jadi, kita bisa menggunakan Adapter Pattern untuk menyelesaikan masalah tersebut. Namun, ketika kita baca kembali, ada proses yang harus dilanjutkan yaitu membuat dua proses secara berurutan sehingga, kita perlu menggunakan Façade Pattern untuk menangani studi kasus tersebut.

Setelah kita tahu solusinya, mari kita buat kodennya !

INSTRUCTION

1. Kita gunakan kembali kode yang telah kita buat semalam. Untuk kode kelas cekTahun manajemen dan pemilik tidak perlu kita ubah, kita hanya perlu menambahkan kelas baru yang berfungsi sebagai Façade, yaitu :

```
class facadeCekTahun:  
    def __init__(self, buku):  
        self.denganPemilik = cekTahunPemilik(buku)  
        self.denganManajemen = cekTahunManajemen(buku)  
    def proses(self):  
        print("Saran dari \"Pemilik\" =")  
        print("Dengan Cek Tahun :\n", self.denganPemilik.cetakBuku())  
        print("\nSaran dari \"Manajemen\" =")  
        print("Dengan Cek Tahun :\n", self.denganManajemen.cetakBuku())  
        print()
```

2. Dikarenakan proses pencetakan cekTahun untuk manajemen dan pemilik telah kita lakukan pada Façade maka untuk kode mainnya menjadi sebagai berikut :

```
if __name__ == '__main__':  
    buku1 = BUKU("Pemrograman Python", 2019, 3)  
    cekThn = facadeCekTahun(buku1)  
    denganCekJumlah = cekStok(buku1)  
  
    print("Awal :\n", buku1.cetakBuku())  
    print()  
    cekThn.proses()  
    print("Dengan Cek Jumlah:\n", denganCekJumlah.cetakBuku())
```

3. Terus, bagaimana dengan hasilnya? Apakah masih sama? Silahkan kita lihat tampilan berikut :

```
Awal :  
Nama Buku = Pemrograman Python  
Tahun Buku = 2019  
Jumlah Buku = 3  
  
Saran dari "Pemilik" =  
Dengan Cek Tahun :  
Nama Buku = Pemrograman Python  
Tahun Buku = 2019  
Jumlah Buku = 3  
Buku terbit pada tahun 2019 sehingga, termasuk buku Lama  
  
Saran dari "Manajemen" =  
Dengan Cek Tahun :  
Nama Buku = Pemrograman Python  
Tahun Buku = 2019  
Jumlah Buku = 3  
Buku termasuk buku Lama dimana buku terbit pada tahun 2019  
  
Dengan Cek Jumlah:  
Nama Buku = Pemrograman Python  
Tahun Buku = 2019  
Jumlah Buku = 3  
Perlu Mencari Pembaca
```

Figure 133 Hasil Studi Kasus

EXERCISE

EXERCISE OBJECTIVES

Pada latihan berikut ini mahasiswa diharapkan dapat melakukan :

- Melakukan analisis masalah terhadap studi kasus yang diberikan.
- Mencari solusi atas masalah yang diberikan.
- Menerjemahkan masalah tersebut ke dalam bentuk kode pemrograman.

TASK 1:

Pada sebuah kelas dari kampus Mikroskil memiliki berbagai kelas yang berbeda-beda dengan jenis ruangan yang berbeda-beda pula. Pada kelas A merupakan kelas T1/L2 yang dimana kelas tersebut terdapat dosen yang mengajarkan mengenai perhitungan kalkulus beserta proses yang ada. Sedangkan pada kelas B merupakan kelas T3/L2 yang dimana kelas tersebut terdapat dosen yang mengajarkan pemrograman beserta proses-proses dalam pemrograman. Sedangkan pada kelas terakhir, kelas C merupakan kelas T5/L2 yang dimana kelas tersebut terdapat dosen yang mengajarkan Bahasa Inggris lengkap dengan formula-formula untuk pembuatan kata-kata. Nah ketika seorang resepsionis ingin melakukan rekap absensi pada ketiga kelas tersebut dengan keluaran berupa nama kelas serta proses yang dilakukan masing-masing, dapatkah Anda membantu resepsionis tersebut untuk membuat sebuah yang membantu dia dalam merekap absensi?

Ketentuan yang dilakukan pada task-01 ini adalah :

- a. Semua proses dilakukan menggunakan kelas, dan untuk menjelaskan kelas harus menggunakan fungsi `__main__`.
- b. Pahami kembali soal dan pilih satu design pattern dari ketiga design pattern yang dipelajari pada beberapa pelajaran sebelumnya (Adapter, Façade, Command) dan silahkan buatkan kodennya.
- c. Silahkan kalian rancang sendiri keluaran dari kelas dan gimana supaya semua proses dapat diselesaikan.
- d. Bagi kalian yang memilih untuk tidak membuat kode, kalian bisa membuat sebuah class diagram dari hasil analisis yang kalian dapatkan.

UNIT 6

DESIGN PATTERN 2 (2)

UNIT OVERVIEW

Pada pertemuan kali ini kita kembali membahas mengenai design pattern bagian terakhir. Pada pola kali ini kita akan lebih berfokus kepada bagaimana cara kita bisa menggabungkan dua buah proses secara bersamaan dan bagaimana cara untuk membuat sebuah bentuk secara terperinci dengan jelas.

UNIT OBJECTIVES

Adapun capaian yang akan kita dapatkan pada modul ini, yaitu :

- Cara memahami penggunaan Abstract Factory Pattern
- Cara memahami penggunaan Composite Pattern

UNIT CONTENTS

Lesson 4: The Abstract Factory Pattern	104-
108	
Lesson 5: The Composite Pattern	108-
110	

PRE LAB

Sebelum masuk ke dalam lab pertama ini, terlebih dahulu kita coba jawab beberapa pertanyaan ini berdasarkan hasil penjelasan pada modul teori!

QUESTION

1. Menurut kalian apa yang dimaksud dengan Command Pattern?
2. Salah satu contoh yang bisa diselesaikan dengan menggunakan Façade Pattern adalah
3. Dari beberapa proses berikut dapat diselesaikan dengan menggunakan pattern :
 - Menghubungkan dua buah objek yang memiliki ciri khas utama yang sama.
 - Membuat nama proses sesuai dengan kesesuaian proses yang ada.
 - Membuat proses yang menggunakan 1 buah tombol untuk menyalakan tv dan mengaktifkan lampu.

CONTENT LESSON

CASE STUDY / PROJECT

Pada perpustakaan “Home Learning is Best” memerlukan sebuah proses baru untuk melakukan pengecekan terhadap buku yang baru masuk. Hal ini bertujuan supaya proses pemasukkan buku sesuai dengan yang ada. Untuk melakukan proses pengecekan buku diperlukan proses lainnya berupa pencetakan kartu dan pemberian kode pada buku baru tersebut. Dimana untuk pemberian kode tersebut didasarkan pada 3 huruf awal dan tahun buku tersebut dimasukkan. Untuk pencetakan kartu nantinya, akan berisi mengenai keterangan dari buku tersebut berupa nama buku, kode buku serta jumlah buku tersebut.

Dapatkah Anda membantu kembali pegawai tersebut? Seperti biasa kita perlu mencari tahu terlebih dahulu masalah yang dihadapi itu seperti apa. Mari kita cari tahu terlebih dahulu!

IDENTIFICATION CONCEPT OF PROBLEM / PROJECT

Pada studi kasus terlihat adanya permasalahan terkait penambahan proses baru. Namun, bukan hanya itu saja. Karena, dalam proses baru tersebut diharapkan untuk menyelesaikan dua hal secara bersamaan. Dimana proses pertama adalah pemberian kode yang didasarkan pada nama dan tahun serta kita juga mencetak kartu untuk buku baru yang dimasukkan. Kira-kira adakah pattern yang kita gunakan untuk menyelesaikan dua buah proses secara bersamaan? Dan jika ada, gimana cara penerapannya ke dalam kode?

Sebelum kita mencari solusi atas permasalahan tersebut, mari kita pelajari terlebih dahulu pattern-pattern yang ada pada bagian design pattern!

LESSON 4: THE ABSTRACT FACTORY PATTERN

LESSON 4: OVERVIEW

The abstract factory pattern merupakan sebuah pola yang digunakan untuk memberikan tanggung jawab kepada sub-class yang menentukan sendiri apa yang harus dibuat. Hal ini bertujuan supaya proses yang ada pada sub-class sesuai dengan apa yang diharapkan pada umumnya. Sebagai contoh ketika saya ingin membuat sebuah browser dan aplikasi messenger maka terlebih dahulu saya harus memberitahukan kira-kira proses apa saja yang harus ada di dalam browser maupun messenger tersebut.

Seperti contoh ini :

```
from abc import ABC, abstractmethod
class Browser(ABC):
    @abstractmethod
    def create_search_toolbar(self):
        pass
    @abstractmethod
    def create_browser_window(self):
        pass
```

Figure 134 Contoh Abstract Factory Design (1)

Pada browser tersebut, saya berhadap nantinya harus ada sebuah toolbar (tombol fungsi) yang berguna untuk mencari data serta tentunya adalah halaman yang digunakan untuk melakukan browsing. Setelah itu, saya juga menjelaskan kira-kira apa saja yang harus dilakukan ketika membuat sebuah messenger seperti berikut ini :

```
class Messenger(ABC):
    @abstractmethod
    def create_messenger_window(self):
        pass
```

Figure 135 Contoh Abstract Factory Design (2)

Pada messenger tersebut saya berharap tentunya ada sebuah halaman yang dapat saya gunakan untuk membuat chatting menjadi lebih mudah. Setelah kita sudah membuat syaratnya, mari kita buat sub-class sebagai berikut ini :

```
class VanillaBrowser(Browser):
    def create_search_toolbar(self):
        print("Search Toolbar Created")
    def create_browser_window(self):
        print("Browser Window Created")

class VanillaMessenger(Messenger):
    def create_messenger_window(self):
        print("Messenger Window Created")
```

Figure 136 Contoh Abstract Factory Design (3)

Bagaimana? Sesuai bukan dengan yang ada pada persyaratan yang kita buat diawal. Hal ini tentunya memudahkan kita untuk membuat sub-class sesuai dengan yang kita harapkan. Namun, apabila saya ingin menambahkan beberapa proses tambahan seperti keamanan pada browser maupun pada messenger apakah itu diperbolehkan? Jawabanya adalah diperbolehkan.

Sebagai contoh, lihat kode berikut ini :

```
class SecureBrowser(Browser):
    def create_search_toolbar(self):
        print("Secure Browser - Search Toolbar Created")
    def create_browser_window(self):
        print("Secure Browser - Browser Window Created")
    def create_incognito_mode(self):
        print("Secure Browser - Incognito Mode Created")

class SecureMessenger(Messenger):
    def create_messenger_window(self):
        print("Secure Messenger - Messenger Window Created")
    def create_privacy_filter(self):
        print("Secure Messenger - Privacy Filter Created")
    def disappearing_messages(self):
        print("Secure Messenger - Disappearing Messages Feature Enabled")
```

Figure 137 Contoh Abstract Factory Design (4)

Pada contoh browser tersebut saya tambahkan 1 buah proses, yaitu Incognito. Dimana seperti yang kita ketahui, mode incognito sangat membantu kita dalam hal segi keamanan. Dimana segala proses dari segi data browsing atau cache yang tersimpan, akan terhapus pada saat browser tersebut kita tutup. Hal ini tentunya menambahkan keamanan bagi setiap pengguna yang menggunakannya. Terus pada bagian messenger saya menambahkan sebuah proses untuk menghilangkan pesan. Hal ini supaya pesan yang kita kirimkan dapat kita sembunyikan sementara. Hal ini bertujuan supaya jika ada pihak ketiga yang ingin mengakses pesan kita, tidak dapat melihat pesan penting yang kita kirimkan.

Nah setelah kita buat kedua proses tersebut, bagaimana cara menerapkannya ke dalam kode? Sebelum menerapkan ke dalam kode, terlebih dahulu kita perlu memberikan syarat kembali. Dimana proses nanti yang saya mau harus ada dua proses sekaligus atau fitur yaitu fitur untuk browser dan fitur untuk messenger. Hal ini bertujuan supaya kita tidak perlu lagi menjalankan dua buah tindakan secara terpisah. Sebagai contoh perhatikan kode berikut ini :

```
class AbstractFactory(ABC):
    @abstractmethod
    def create_browser(self):
        pass
    @abstractmethod
    def create_messenger(self):
        pass
```

Figure 138 Contoh Abstract Factory Design (5)

Dari kode tersebut terlihat bahwa, ketika saya menjalankan proses nanti diharuskan untuk membuat dua proses berjalan sekaligus, yaitu proses pembuatan browser dan pembuatan messenger. Nah bagaimana cara penerapannya ke dalam sub-class?

Sama seperti dengan contoh-contoh sebelumnya, seperti tampilan berikut ini :

```
class VanillaProductsFactory(AbstractFactory):
    def create_browser(self):
        return VanillaBrowser()
    def create_messenger(self):
        return VanillaMessenger()

class SecureProductsFactory(AbstractFactory):
    def create_browser(self):
        return SecureBrowser()
    def create_messenger(self):
        return SecureMessenger()
```

Figure 139 Contoh Abstract Factory Design (6)

Gimana? Mudah bukan. Kita tidak perlu lagi membuat dua proses atau mendeklarasikan class secara terpisah. Karena semua itu, bisa kita lakukan hanya dengan menggunakan 1 sub-class saja dengan jelas. Terus, gimana prosesnya pada main? Kita perhatikan kode berikut ini :

```
if __name__ == "__main__":
    for factory in (VanillaProductsFactory(), SecureProductsFactory()):
        product_a = factory.create_browser()
        product_b = factory.create_messenger()
        product_a.create_browser_window()
        product_a.create_search_toolbar()
        product_b.create_messenger_window()
```

Figure 140 Contoh Abstract Factory Design (7)

Nah gimana? Tentunya ini semua berkat adanya Abstract Factory Design. Kita tidak perlu lagi memanggil proses secara terpisah. Semuanya bisa dijalankan secara bersamaan. Terus, apakah akan terjadi error nantinya? Supaya tidak bertanya-tanya, silahkan lihat tampilan berikut ini :

```
Browser Window Created
Search Toolbar Created
Messenger Window Created
Secure Browser - Browser Window Created
Secure Browser - Search Toolbar Created
Secure Messenger - Messenger Window Created
```

Figure 141 Hasil Abstract Factory Design

Sumber : [stackabuse](#).

Nah mudah bukan? Dan tidak ada terjadi kesalahan pada kode. Terus, apakah ada pattern lain yang kita gunakan pada design pattern? Jawabannya adalah ada. Kita akan pelajari berikutnya, terkait

Composite Pattern. Gimana cara penerapannya, dan apa saja yang harus kita lakukan? Yuk kita pelajari ke pelajaran berikutnya!

LESSON 5: THE COMPOSITE PATTERN

LESSON 5: OVERVIEW

The composite pattern merupakan sebuah pola yang dapat digunakan sebagai sebuah cara untuk menjelaskan secara langsung yang terjadi pada sub-classnya. Sebagai contoh ketika kita kedatang sebuah paket, terus di dalam paket tersebut terdapat beberapa benda, diantaranya ada hp, laptop, dan uang cash. Kita disuruh untuk menghitung total harga yang ada pada paket tersebut. Mungkin cara ini bisa kita selesaikan dengan menggunakan proses looping. Namun, bayangkan, apabila di dalam kode tersebut terdapat kode yang lain? Apa yang harus kita lakukan? Tentunya kita juga harus mentalkan harga yang ada pada kotak kecil tersebut, baru kita jumlahkan dengan kotak yang besar bukan? Namun, gimana caranya kita terapkan pada kode?

Sebelum kita menjelaskan kode tersebut, terlebih dahulu kita buat sebuah ketentuan. Dimana setiap produk atau barang yang ada di dalam kotak tersebut, memiliki harga. Dan itu bisa kita buat dalam kode seperti berikut ini :

```
from abc import ABC, abstractmethod

class Item(ABC):
    @abstractmethod
    def return_price(self):
        pass
```

Figure 142 Contoh Composite Pattern (1)

Nah setelah kita beritahu bahwa setiap barang harus ada harganya, kita buat beberapa proses yang akan dilakukan pada kotak, yaitu :

```
class Box(Item):
    def __init__(self, contents):
        self.contents = contents
    def return_price(self):
        price = 0
        for item in self.contents:
            price = price + item.return_price()
        return price
```

Figure 143 Contoh Composite Pattern (2)

Pada kotak tersebut tentunya ada barang, dan barang yang ada di dalam kotak tersebut tentunya tidak hanya 1 saja, bisa lebih dari 1. Dan untuk setiap barang pastinya memiliki harga, karena sesuai dengan persyaratan yang telah kita buat pada awalnya. Sehingga, untuk menghasilkan total harga yang ada pada kotak, kita hanya perlu mentalkan setiap harga barang yang ada pada kotak tersebut.

Terus, bagaimana dengan barang-barangnya? Apakah perlu kita lakukan proses yang sama dengan kotak? Tentunya tidak perlu. Silahkan perhatikan kode berikut ini :

```
class Phone(Item):
    def __init__(self, price):
        self.price = price
    def return_price(self):
        return self.price

class Charger(Item):
    def __init__(self, price):
        self.price = price
    def return_price(self):
        return self.price

class Earphones(Item):
    def __init__(self, price):
        self.price = price
    def return_price(self):
        return self.price
```

Figure 144 Contoh Composite Pattern (3)

Disini saya hanya menggunakan 3 buah barang saja yang kita beli ketika membeli sebuah handphone. Dari masing-masing barang tersebut tentunya memiliki harganya masing-masing yang dapat kita kembalikan dengan menggunakan method return_price sesuai dengan persyaratan yang kita buat diawal. Terus, gimana proses penerapannya ke dalam proses utama? Perhatikan contoh kode berikut ini :

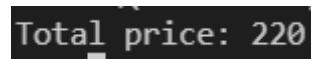
```
if __name__ == "__main__":
    phone_case_contents = []
    phone_case_contents.append(Phone(200))
    phone_case_box = Box(phone_case_contents)

    big_box_contents = []
    big_box_contents.append(phone_case_box)
    big_box_contents.append(Charger(10))
    big_box_contents.append(Earphones(10))
    big_box = Box(big_box_contents)

    print("Total price: " + str(big_box.return_price()))
```

Figure 145 Contoh Composite Pattern (4)

Pada main ini, saya buat sedikit scenario berupa ketika saya membuat sebuah handphone dengan harga 200 dollars, setelah itu pada kode handphone tersebut terdapat juga charger seharga 10 dollars dan earphones seharga 10 dollars. Namun, saya tidak mau ambil pusing. Saya hanya mau tahu berapa harga total yang perlu saya bayarkan? Tentunya apabila kita menghitung secara manual pastikan akan lama. Namun, karena prosesnya telah kita buat ke dalam composite pattern kita hanya perlu memanggil method `return_price` pada kotak besar untuk memunculkan total harga yang harus saya bayar. Sehingga, total harga yang harus saya bayar adalah :



Total price: 220

Figure 146 Hasil Composite Pattern

Sumber :

- [Stackabuse](#).
- [GeeksforGeeks](#).

Nah, tentunya saya langsung tahu kira-kira berapa harganya tanpa perlu menghitung barangnya 1 per 1. Gimana? Mudah bukan? Yuk kita kembali cari tahu solusi untuk permasalahan pada studi kasus kita ini!

SOLUTION

Setelah kita memahami kedua pattern tersebut dan memahami kembali studi kasus yang ingin kita selesaikan, kita hanya perlu menggunakan Abstract Factory Pattern. Karena, dengan adanya pattern tersebut kita tidak perlu lagi melakukan dua proses secara terpisah. Kita bisa melakukannya secara bersamaan. Namun, kita penerapannya ke dalam kode?

Yuk kita bahas 1 per 1 pada kode kita ini!

INSTRUCTION

1. Untuk kodennya saya gunakan kode yang ada sebelumnya. Sebelum kita lanjut kepada kode tersebut, terlebih dahulu saya tambahkan class yang akan digunakan sebagai abstract nantinya sebagai berikut :

```
from abc import ABC, abstractmethod
```

2. Setelah itu, kita tambahkan syarat terlebih dahulu apa yang harus dipenuhi untuk kedua kode tersebut sebagai berikut :

```
class kartuBuku(ABC):  
    @abstractmethod  
    def cetakKartu(self):  
        pass
```

3. Setelah itu kita buat sub-class sesuai dengan permintaan soal sebagai berikut :

```
class kodeBuku(kartuBuku):  
    def cetakKartu(self, buku):  
        print(f"Kode Buku = {buku.nama[0:3]} + str(buku.tahun)")  
  
class kartuBuku(kartuBuku):  
    def cetakKartu(self, buku):  
        print(f"Nama Buku = {buku.nama}")  
        print(f"Jumlah Buku = {buku.jlh}")
```

4. Setelah itu, kita kembali menambahkan beberapa persyaratan untuk kelas abstract factory design nantinya, sebagai berikut :

```
class factoryKartuBuku(ABC):
    @abstractmethod
    def cetakKode(self):
        pass
    @abstractmethod
    def cetakData(self):
        pass
```

5. Setelah kita buat syaratnya, sekarang kita buat kelas factory designnya menjadi seperti berikut ini :

```
class afdKartuBuku(factoryKartuBuku):
    def cetakKode(self):
        return kodeBuku()
    def cetakData(self):
        return kartuBuku()
```

6. Setelah kita buat itu, kita tambahkan kode di main class supaya bisa mengakses atau membuat kartu buku yang telah kita buat dengan menerapkan prinsip abstract factory design, sehingga kode pada main menjadi seperti berikut :

```
if __name__ == '__main__':
    buku1 = BUKU("Pemrograman Python", 2019, 3)
    cekThn = facadeCekTahun(buku1)
    denganCekJumlah = cekStok(buku1)

    print("Awal :\n", buku1.cetakBuku())
    print()
    cekThn.proses()
    print("Dengan Cek Jumlah:\n", denganCekJumlah.cetakBuku())

    ctkKrt = afdKartuBuku()
    kdBuku = ctkKrt.cetakKode()
    dataBuku = ctkKrt.cetakData()
    print("\nKartu Buku")
    kdBuku.cetakKartu(buku1)
    dataBuku.cetakKartu(buku1)
```

NB : Untuk kode yang diberi warna **hijau**, merupakan kode tambahan yang digunakan untuk memanggil abstract factory yang telah kita buat sebelumnya.

7. Setelah semuanya selesai, maka untuk kode akan menghasilkan tampilan berikut ini :

```
Awal :  
Nama Buku = Pemrograman Python  
Tahun Buku = 2019  
Jumlah Buku = 3  
  
Saran dari "Pemilik" =  
Dengan Cek Tahun :  
Nama Buku = Pemrograman Python  
Tahun Buku = 2019  
Jumlah Buku = 3  
Buku terbit pada tahun 2019 sehingga, termasuk buku Lama  
  
Saran dari "Manajemen" =  
Dengan Cek Tahun :  
Nama Buku = Pemrograman Python  
Tahun Buku = 2019  
Jumlah Buku = 3  
Buku termasuk buku Lama dimana buku terbit pada tahun 2019
```

Figure 147 Hasil Studi Kasus (1)

```
Dengan Cek Jumlah:  
Nama Buku = Pemrograman Python  
Tahun Buku = 2019  
Jumlah Buku = 3  
Perlu Mencari Pembaca  
  
Kartu Buku  
Kode Buku = Pem2019  
Nama Buku = Pemrograman Python  
Jumlah Buku = 3
```

Figure 148 Hasil Studi Kasus (2)

EXERCISE

EXERCISE OBJECTIVES

Pada latihan berikut ini mahasiswa diharapkan dapat melakukan :

- Melakukan analisis masalah terhadap studi kasus yang diberikan.
- Mencari solusi atas masalah yang diberikan.
- Menerjemahkan masalah tersebut ke dalam bentuk kode pemrograman.

TASK 1:

Pada kampus Mikroskil terdapat sebuah proses pendaftaran mahasiswa. Dimana proses pendaftaran tersebut dilakukan berdasarkan data tabel berikut :

Biaya	Total
Uang Pendaftaran	Rp. 200.000,00
Uang Kuliah Pertama	Rp. 1.500.000,00
Uang MPT : - Uang training - Uang penginapan - Uang konsumsi	Rp. 100.000,00 Rp. 200.000,00 Rp. 150.000,00

NB : Ini hanya sebatas asumsi saja. Bukan kenyataan.

Mungkin ketika kita lihat sementara kita langsung dapat mengetahui secara langsung. Namun, dapatkah kalian membantu bagian pendaftaran Mikroskil dengan menerapkan salah satu pola untuk mengatasi masalah tersebut?

Ketentuan yang dilakukan pada task-01 ini adalah :

- a. Semua proses dilakukan menggunakan kelas, dan untuk menjalankan kelas harus menggunakan fungsi `main`.
- b. Pahami kembali soal dan pilih satu design pattern dari ketiga design pattern yang dipelajari pada beberapa pelajaran sebelumnya (Abstract Factory Design Pattern dan Composite Pattern) dan silahkan buatkan kodennya.
- c. Silahkan kalian rancang sendiri keluaran dari kelas dan gimana supaya data tersebut bisa dijumlahkan dengan benar.
- d. Bagi kalian yang memilih untuk tidak membuat kode, kalian bisa membuat sebuah class diagram atau buat penjelasan dari hasil analisis yang kalian dapatkan.

UNIT 7

TESTING OOP

UNIT OVERVIEW

Testing OOP atau biasa dibilang sebagai pengujian dalam OOP merupakan sebuah proses yang digunakan untuk memastikan kode yang dibuat telah sesuai dengan yang ditentukan diawal. Hal ini tentunya sangatlah penting. Mengingat proses akan terus berjalan secara terus menurus dan juga proses harus bisa mengerjakan banyak hal. Oleh karena itu, proses pengujian ini menjadi sangat penting untuk dilakukan.

UNIT OBJECTIVES

Adapun capaian yang akan kita dapatkan pada modul ini, yaitu :

- Cara melakukan pengujian dengan menggunakan unittest
- Cara melakukan pengujian dengan menggunakan py.test

UNIT CONTENTS

Lesson 1: Testing dengan unittest	115-
	118
Lesson 2: Testing dengan Py.Test	118-
	119

PRE LAB

Sebelum masuk ke dalam lab pertama ini, terlebih dahulu kita coba jawab beberapa pertanyaan ini berdasarkan hasil penjelasan pada modul teori!

QUESTION

1. Menurut kalian apa yang dimaksud dengan Composite Pattern?
2. Dengan menggunakan Abstract Factory Pattern, kita bisa ...
3. Dari nama proses berikut manakah yang bisa diselesaikan dengan menggunakan Design Pattern yang telah dijelaskan pada UNIT 12 sebelumnya :
 - Proses strukturisasi kerangka jabatan.
 - Penggabungan dua buah proses menjadi 1 kesatuan dengan nama yang baru.
 - Melakukan pengecekan.

CONTENT LESSON

CASE STUDY / PROJECT

Pada perpustakaan “Home Learning is Best” ingin menambahkan sebuah fitur baru terkait dalam pencarian buku yang ada di dalam perpustakaan tersebut. Namun, karena proses yang ada di dalam aplikasi kurang lebih sudah banyak, sehingga diperlukan pengecekan kesesuaian dari fitur tersebut dengan kebutuhan yang ada pada perpustakaan. Dimana keperluan tersebut berupa, apabila data buku dimasukkan berupa nama buku, maka akan memunculkan data berupa nama buku beserta tahun bukunya.

Yuk kita bantu programmer tersebut! Namun, sebelum itu mari kita cari tahu dulu permasalahannya seperti apa dengan melakukan indentifikasi masalah.

IDENTIFICATION CONCEPT OF PROBLEM / PROJECT

Setelah kita baca kembali studi kasus terdapat masalah utama yaitu penambahan fitur pencarian. Namun, karena proses pada aplikasi yang sudah besar dan tentunya apabila kita tambahkan langsung ke dalam kode, bisa jadi nantinya dapat memberikan kode tersebut masalah. Oleh karena itu, kita perlu melakukan pengecekan terhadap kodennya untuk memastikan kode tersebut dapat berjalan. Pada dasarnya kalau kita lakukan secara langsung tidak masalah. Namun, bisa jadi nantinya akan menjadi permasalahan ketika program tersebut dijalankan secara langsung. Jadi, adakah cara yang bisa digunakan untuk melakukan dalam proses tersebut?

Jawabannya pasti ada. Tapi, kita harus paham terlebih dahulu mengenai cara yang digunakan untuk melakukan pengecekan. Oleh karena itu, mari kita pelajari terlebih dahulu cara pengecekan yang ada Python berikut!

LESSON 1: TESTING DENGAN UNITTEST

LESSON 1: OVERVIEW

Unittest merupakan sebuah library atau module yang sudah ada di dalam Python tanpa perlu kita melakukan instalasi lagi. Cara menggunakannya juga sangatlah mudah, kita hanya perlu memanggilnya dengan menggunakan **import unittest**. Namun, gimana cara penerapannya ke dalam kode?

Mari kita lihat contoh kode program kalkulator sederhana berikut :

```
class kalku:
    def tambah(a, b):
        return a + b
    def kurang(a, b):
        if(a > b):
            return a - b
        else:
            return b - a
    def kali(a, b):
        return a * b
    def bagi(a, b):
        return a / b
```

Figure 149 Contoh Program Kalkulator

Terlihat sangat sederhana bukan? Tentunya. Karena, ini hanya saya gunakan sebagai kelas untuk melakukan pengujian apakah sudah sesuai dengan gimana cara penerapan dalam unittest. Terus, gimana cara pengujian dalam unittest? Mari kita lihat contoh kode berikut :

```
import unittest
from kalkulator import kalku
class test_kalku(unittest.TestCase):
    def test_tambah(self):
        self.assertEqual(kalku.tambah(10, -30), -20)
    def test_kurangBenar(self):
        self.assertEqual(kalku.kurang(10, -30), 40)
    def test_kurangSalah(self):
        self.assertEqual(kalku.kurang(10, -30), -40)
    def test_kali(self):
        self.assertEqual(kalku.kali(10, -30), -300)
    def test_bagi(self):
        self.assertEqual(kalku.bagi(10, -20), -0.5)

if __name__ == "__main__":
    unittest.main()
```

Figure 150 Contoh Penggunaan unittest (1)

Dari kode tersebut untuk proses pengujian pada fungsi kurang, saya ada buat 2 kali proses pengujian. Dimana yang pertama untuk pengecekan yang benar, dan yang dibawah terkait proses pengecekan yang salah. Kenapa harus demikian? Karena, apabila proses tersebut saya jalankan secara bersamaan atau dalam 1 buah fungsi yang sama, maka akan dianggap hanya sekali. Jadi, apabila kalian ingin

melakukan pengecekan 5 kali, maka kalian mesti membuat fungsi pengecekan sebanyak 5 kali. Terus, gimana keluarannya? Apakah ada diberitahukan terkait kegagalan atau ketidaksesuaian pada proses? Untuk lebih jelasnya, perhatikan hasil jalan kode berikut :

```
...F.  
===== FAIL: test_kurangSalah (__main__.test_kalku)  
-----  
Traceback (most recent call last):  
  File "testunit.py", line 9, in test_kurangSalah  
    self.assertEqual(kalku.kurang(10, -30), -40)  
AssertionError: 40 != -40  
  
-----  
Ran 5 tests in 0.001s  
  
FAILED (failures=1)
```

Figure 151 Hasil Penggunaan unittest (1)

Dari hasil tersebut dapat terlihat pada baris pertama ada memberitahukan berupa "...F.". Untuk perlu diketahui terkati hasil ini dapat melihat keterangan berikut :

- Apabila hasilnya berupa “.” (dot) menandakan proses pengujian berhasil dan tidak ada kesalahan.
- Apabila hasilnya berupa “F” (False) menandakan proses pengujian tidak berhasil atau tidak sesuai dengan yang diharapkan.
- Apabila hasilnya berupa “E” (Error) menandakan proses pengujian mengalami kendala yang membuat sistem tidak bisa dilanjutkan.

Namun, itu semua dapat kalian lihat lebih jelas, dengan mengganti bagian main berikut, menjadi seperti ini :

```
if __name__ == "__main__":  
    unittest.main(verbose=2)
```

Figure 152 Contoh Penggunaan unittest (2)

Penambahan tersebut berfungsi untuk memberikan detail terhadap proses keluaran yang diberikan oleh program. Sehingga, kita tidak perlu lagi mengingat arti dari tiap symbol yang ditampilkan.

Adapun hasil keluarannya menjadi seperti berikut :

```
test_bagikan (_main_.test_kalku) ... ok
test_kali (_main_.test_kalku) ... ok
test_kurangBenar (_main_.test_kalku) ... ok
test_kurangSalah (_main_.test_kalku) ... FAIL
test_tambah (_main_.test_kalku) ... ok

=====
FAIL: test_kurangSalah (_main_.test_kalku)
-----
Traceback (most recent call last):
  File "testunit.py", line 9, in test_kurangSalah
    self.assertEqual(kalku.kurang(10, -30), -40)
AssertionError: 40 != -40

-----
Ran 5 tests in 0.002s

FAILED (failures=1)
```

Figure 153 Hasil Penggunaan unittest (2)

Gimana? Lebih detail bukan? Hal ini tentunya bertujuan supaya kita tidak salah paham dalam memahami setiap symbol yang ada. Sehingga, kita bisa mengetahui bahwa proses pada pengujian yang ke-4 memberikan keluaran yang tidak sesuai dengan apa yang direncakan pada bagian kode unittest.

Gimana? Pastinya sekarang kalian lebih memahami kode yang kalian buat. Dan kalian juga bisa berkreasi terhadap keluaran yang ingin kalian buat. Namun, selain penggunaan dari unittest adakah pengujian lain yang bisa gunakan pada Python? Jawabannya adalah ada, yaitu py.test. Gimana cara pakainya dan apakah sama juga keluaran yang diberikan? Kita perhatikan saja pelajaran berikutnya!

LESSON 2: TESTING DENGAN PY.TEST

LESSON 2: OVERVIEW

Py.Test juga merupakan salah satu alat yang bisa kita gunakan untuk melakukan pengujian terhadap kode yang kita buat. Hanya saja, kembali pada Python kita hanya bisa melakukan pengujian terhadap kesesuaian hasil bukanlah terhadap tipe datanya.

Untuk menggunakan py.test ini tidak bisa dilakukan secara langsung, kita harus melakukan instalasinya terlebih dahulu harus melakukan pemasangan. Untuk melakukan instalasi pada python kita bisa menggunakan perintah : **pip install pytest**. Setelah itu, baru kita bisa gunakan pada kode program yang kita buat. Terus, untuk proses penerapannya ke dalam kode gimana? Untuk pengecekan kodennya, saya menggunakan kode program sebelumnya, yaitu kalkulator.

Untuk proses pengecekan yang dijalankan pada pytest sebagai berikut :

```
from kalkulator import kalku

def test_tambah():
    assert kalku.tambah(10, -30) == -20
def test_kurangBenar():
    assert kalku.kurang(10, -30) == 40
def test_kurangSalah():
    assert kalku.kurang(10, -30) == -40
def test_kali():
    assert kalku.kali(10, -30) == -300
def test_bagi():
    assert kalku.bagi(10, -20) == -0.5
```

Figure 154 Pengujian dengan Py.Test

Pada kode tersebut, kita tidak perlu lagi mendeklarasikan sebuah kelas dan fungsi main. Kita hanya perlu menggunakan fungsi perbandingan untuk membandingkan apakah nilai yang kita buat sudah sesuai dengan yang kita harapkan atau belum. Terus, gimana hasilnya? Untuk hasilnya dapat kita lihat pada tampilan berikut ini :

```
collected 5 items

test_kalku.py ...F...

===== FAILURES =====
test_kurangSalah

def test_kurangSalah():
>     assert kalku.kurang(10, -30) == -40
E     assert 40 == -40
E     +  where 40 = <function kalku.kurang at 0x0000029179BD6A60>(10, -30)
E     +  where <function kalku.kurang at 0x0000029179BD6A60> = kalku.kurang

test_kalku.py:8: AssertionError
===== short test summary info =====
FAILED test_kalku.py::test_kurangSalah - assert 40 == -40
===== 1 failed, 4 passed in 0.07s =====
```

Figure 155 Hasil Pengujian Py.Test

Bisa dilihat proses hasil lebih jelas. Disana juga dilampirkan waktu yang dibutuhkan untuk melakukan pengujian. Sehingga, kita bisa mengetahui hasil dari kode kita apakah sudah sesuai atau perlu dilakukan perbaikan. Dan tentunya hal ini bisa membuat kode yang kita buat menjadi lebih jelas dan terarah.

Gimana? Tentunya lebih mudah bukan ketimbang menggunakan unittest? Namun, semua tentunya kembali lagi pada diri sendiri lebih mudah menggunakan yang unittest atau py.test. Yuk kita balik lagi mencari solusi untuk permasalahan kita!

SOLUTION

Setelah kita membaca kembali studi kasus tersebut, solusi untuk dari permasalahan tersebut adalah melakukan pengujian terhadap fitur baru tersebut. Dan untuk mempermudah dalam pengujian terdapat fitur baru tersebut kita menggunakan py.test untuk menangani. Namun, gimana penerapannya ke dalam kode?

Nah supaya lebih jelas, yuk kita bahas 1 per 1.

INSTRUCTION

1. Tahap awal kita buat terlebih dahulu kelas untuk penambahan fitur barunya menjadi seperti berikut :

```
class BUKU:  
    def __init__(self, nama, tahun, jlh):  
        self.nama = nama  
        self.tahun = tahun  
        self.jlh = jlh  
  
class fiturBaru:  
    data = []  
    def tambahBuku(self, buku):  
        self.data.append(buku)  
    def pencarianBuku(self, cari):  
        for i in range(len(self.data)):  
            if(self.data[i].nama == cari):  
                return [self.data[i].nama, self.data[i].tahun]  
        return [0, 0]
```

2. Setelah itu, kita buat untuk proses pengujian dengan menggunakan Py.Test seperti tampilan kode berikut :

```
from jawabanPert13 import BUKU, fiturBaru  
import pytest  
  
@pytest.fixture  
def data(request):  
    buku1 = BUKU("Pemrograman C", 2021, 3)  
    buku2 = BUKU("Pemrograman C#", 2022, 10)  
    fb = fiturBaru()  
    fb.tambahBuku(buku1)  
    fb.tambahBuku(buku2)  
    return fb  
  
def test_pencarianBenar(data):  
    assert data.pencarianBuku("Pemrograman C") == ["Pemrograman C", 2021]  
def test_pencarianSalah(data):  
    assert data.pencarianBuku("pemrograman c") == ["Pemrograman C", 2021]
```

NB :

- Untuk versi py.test 3 ke atas menggunakan `pytest.fixture` untuk menambahkan variabel baru ke dalam pengujian.
- Untuk versi py.test 2 menggunakan `pytest_funcarg_[nama_variable](request)` untuk menambahkan variabel baru ke dalam pengujian.

3. Setelah semua kode telah dibuat, kita jalankan yang langkah ke-2 tersebut dan memberikan hasil seperti ini :

```
collected 2 items

test_fungsi.py .F

===== FAILURES =====
____ test_pencarianSalah ____

data = <jawabanPert13.fiturBaru object at 0x0000021467489E48>

def test_pencarianSalah(data):
>     assert data.pencarianBuku("pemrograman c") == ["Pemrograman C", 2021]
E     AssertionError: assert [0, 0] == ['Pemrograman C', 2021]
E         At index 0 diff: 0 != 'Pemrograman C'
E         Use -v to get the full diff

test_fungsi.py:16: AssertionError
===== short test summary info =====
FAILED test_fungsi.py::test_pencarianSalah - AssertionError: assert [0, 0] == ['Pemrograman C', 2021]
===== 1 failed, 1 passed in 0.06s =====
```

Figure 156 Hasil Studi Kasus

EXERCISE

EXERCISE OBJECTIVES

Pada latihan berikut ini mahasiswa diharapkan dapat melakukan :

- Melakukan analisis masalah terhadap studi kasus yang diberikan.
- Mencari solusi atas masalah yang diberikan.
- Menerjemahkan masalah tersebut ke dalam bentuk kode pemrograman.

TASK 1:

Pada sebuah kelas di Kampus Mikroskil terdapat mahasiswa dan dosen pada kelas tersebut. Mahasiswa tersebut tentunya memiliki data berupa (*bagian ini kalian cari tahu sendiri*). Sedangkan dosen memiliki data berupa NIP, Nama, jabatan, jenis kelamin, dan no HP. Pada suatu acara, seorang mahasiswa ditujukan untuk melakukan rekap absensi dari acara tersebut. Absensi tersebut nantinya harus bisa dicetak serta bisa dihitung kira-kira berapa banyak jumlah pengunjung pada acara tersebut. Untuk proses perhitungan, akan dilakukan pengecekan. Apakah data sudah sesuai atau belum. Dapatkah kalian membantu programer tersebut terhadap proses perhitungan jumlah mahasiswa yang datang?

Ketentuan yang dilakukan pada task-01 ini adalah :

- a. Gunakan kode untuk task-01 pertemuan-01.
- b. Silahkan pilih penggunaan apakah menggunakan unittest atau py.test
- c. Rancanglah pengujian yang kalian buat supaya bisa memastikan fungsi perhitungan sudah sesuai atau belum.

UNIT 8

PRINSIP SOLID PADA PYTHON (1)

UNIT OVERVIEW

Prinsip SOLID merupakan sebuah cara yang digunakan dalam pemrograman yang berfokus terhadap penggunaan kode yang berulang dan pengembangan. Karena, pada umumnya semua kode yang dibuat asalkan dapat berjalan sudah bisa dikatakan dengan baik. Hanya saja pada saat ketahapan selanjutnya terkait pengujian, pengembangan dan penambahan fitur harus diperhatikan juga dalam pembuatan kode. Hal ini yang menjadi fokus terbentuknya prinsip SOLID ini.

UNIT OBJECTIVES

Adapun capaian yang akan kita dapatkan pada modul ini, yaitu :

- Cara menerapkan prinsip Single Responsibility Principle (SRP)
- Cara menerapkan prinsip Open-Closed Principle

UNIT CONTENTS

Lesson: Lesson Name	1-8
Lesson: Lesson Name	1-38
Lesson: Lesson Name	1-88

PRE LAB

Sebelum masuk ke dalam lab pertama ini, terlebih dahulu kita coba jawab beberapa pertanyaan ini berdasarkan hasil penjelasan pada modul teori!

QUESTION

1. Menurut kalian kenapa kita perlu melakukan testing dalam pengembangan aplikasi?
2. Menurut kalian, lebih mudah menggunakan py.test atau unittest?
3. Dari contoh proses berikut manakah yang termasuk ke dalam Unit Test, sub-System Test, System Test :
 - Pembuatan aplikasi perkantoran.
 - Pembuatan proses pendaftaran mahasiswa baru.
 - Proses pengecekan dua buah variabel.

CONTENT LESSON

CASE STUDY / PROJECT

Seperti yang kita ketahui, dimana pada aplikasi pada perpustakaan “Home Learning is Best” ada perbedaan pendapat antara pemilik dan project manager. Hal ini tentunya menjadi sebuah tantangan sendiri dalam pembuatan aplikasi. Namun, apabila zaman semakin berkembang dan proses pencetakan sebelumnya dianggap tidak sesuai dengan yang diharapkan. Sehingga, ada keperluan untuk mengganti dengan yang ada saat ini.

Waduh? Gimana ya? Kita harus cari tahu dulu, apakah kita harus mengubah kode secara keseluruhan? Terus, nanti kalau perkembangan berikutnya, kita juga ubah kode lagi?

IDENTIFICATION CONCEPT OF PROBLEM / PROJECT

Permasalahan yang terjadi pada studi kasus tersebut adalah proses yang sering berubah karena adanya kebutuhan dari pembuat, pemilik dan zaman. Hal ini tentunya sering terjadi pada saat ini. Dimana ketika perkembangan zaman semakin berkembang, kita tentunya harus memstikan kembali bahwa setiap proses harus bisa berjalan sesuai dengan perkembangan tersebut. Namun, gimana caranya kita bisa membuat kode yang bisa sesuai dengan perkembangan zaman tersebut?

Sebelum menjawab pertanyaan tersebut, mari kita belajar terlebih dahulu!

LESSON 1: SINGLE RESPONSIBILITY PRINCIPLE

LESSON 1: OVERVIEW

Single responsibility principle (SRP) merupakan prinsip yang berfokus ke dalam sebuah kelas yang hanya diubah apabila hanya karena 1 alasan. Ini diartikan sebagai sebuah kelas pada umumnya hanya menangani 1 masalah saja. Sebagai contoh, ketika kita membuat sebuah aplikasi untuk penyimpanan data ke dalam database terlebih dahulu kita harus melakukan pengecekan ke dalam data tersebut. Dan apabila terjadi perubahan pada tempat penyimpanannya, maka kita harus mengganti kode secara keseluruhan. Hal ini tentunya menjadi sebuah masalah apabila kode yang kita buat mempengaruhi keseluruhan pada proses. Jadi, gimana caranya kita bisa menyelesaikan masalah tersebut?

Sebagai contoh perhatikan potongan kode berikut :

```
def simpanData(nama):
    cek = True
    for kar in nama:
        if(not(ord("A") <= ord(kar) <= ord("Z") or ord("a") <= ord(kar) <= ord("z") or kar == " ")):
            cek = False
    if(cek):
        print(f"{nama} berhasil disimpan ke dalam data\n")
    else:
        print("Mohon maaf nama tidak bisa disimpan ya ... !!!\n")
```

Figure 157 Contoh SRP (1)

Pada kode tersebut sekilas menurut kita tidak ada yang salah, dan apabila dijalankan dengan menggunakan perintah berikut :

```
if __name__ == "__main__":
    simpanData("Apriyanto Halim")
    simpanData("S1@p@ y@ng t@hu")
    simpanData("AnEh sAjA sIh")
```

Figure 158 Contoh SRP (2)

Kode tersebut dapat berjalan dengan baik. Namun, permasalahan muncul, dimana ketika saya ingin membuat sebuah fungsi baru yang mengharus saya untuk melakukan cek nama terlebih dahulu, otomatis saya perlu mengetikkannya kembali, seperti contoh berikut :

```
def loginNama(nama):
    cek = True
    for kar in nama:
        if(not(ord("A") <= ord(kar) <= ord("Z") or ord("a") <= ord(kar) <= ord("z") or kar == " ")):
            cek = False
    if(cek):
        print(f"{nama} sesuai dengan ketentuan yang ada dan bisa login.\n")
    else:
        print("Mohon maaf nama tidak sesuai, sehingga Anda tidak bisa login.\n")
```

Figure 159 Contoh SRP (3)

Dari contoh tersebut terlihat bahwa kita mengulangi kembali kode yang telah kita buat sebelumnya ke dalam fungsi yang baru. Tentunya hal ini tidak sesuai dengan prinsip yang ada pada SRP. Oleh karena itu, kita harus membagi tanggungjawab untuk kode masing-masing. Seperti untuk proses pengecekan kita gabung ke dalam 1 buah fungsi baru, dan untuk fungsi simpan kita gabungkan ke dalam 1 buah fungsi baru. Hal ini bertujuan supaya proses penambahan fitur dapat sesuai dengan yang ada.

Oleh karena itu, untuk method simpanData saya bagi menjadi 2 fungsi, yaitu :

```
def cekNama(nama):
    for kar in nama:
        if(not(ord("A") <= ord(kar) <= ord("Z") or ord("a") <= ord(kar) <= ord("z")) or kar == " "):
            return False
    return True

def simpanData(nama):
    if(cekNama(nama)):
        print(f"{nama} berhasil disimpan ke dalam data\n")
    else:
        print("Mohon maaf nama tidak bisa disimpan ya ... !!!\n")
```

Figure 160 Contoh SRP (4)

Gimana? Pastinya diawal kita merasa jumlah baris menjadi bertambah. Namun, apabila kita menambahkan 1 buah fitur baru kayak yang tadi berupa loginNama, maka akan menjadi seperti ini :

```
def loginNama(nama):
    if(cekNama(nama)):
        print(f"{nama} sesuai dengan ketentuan yang ada dan bisa login.\n")
    else:
        print("Mohon maaf nama tidak sesuai, sehingga Anda tidak bisa login.\n")
```

Figure 161 Contoh SRP (5)

Hal ini tentunya membuat kita menjadi lebih gampang untuk menambah fungsi berikut yang berkaitan dengan fungsi sebelumnya. Oleh karena itu, mari kita mulai membuat kode kita dengan memulai dari prinsip SRP ini.

Terus, apakah ketika kode yang kita buat ini sudah selesai, apakah sudah membuat kita menjadi lebih mudah? Jawabannya masih belum tentu. Karena, apabila kita masuk ke dalam main program yang tadi, ketika kita ingin menjalankan simpanNama dan loginNama sekaligus, maka kita harus mengganti format pada bagian mainnya seperti tampilan berikut :

```
if __name__ == "__main__":
    simpanData("Apriyanto Halim")
    loginNama("Apriyanto Halim")

    simpanData("S1@p@ y@ng t@hu")
    loginNama("S1@p@ y@ng t@hu")

    simpanData("AnEh sAjA sIh")
    loginNama("AnEh sAjA sIh")
```

Figure 162 Contoh SRP (6)

Gimana? Ribet bukan. Adakah cara yang bisa kita gunakan untuk menyelesaikan permasalahan tersebut? Jawabannya ada, dengan menggunakan prinsip selanjutnya, yaitu Open-Closed Principle (OCP). Mari kita pelajari Bersama!

LESSON 2: OPEN-CLOSED PRINCIPLE

LESSON 2: OVERVIEW

Open-closed principle (OCP) merupakan prinsip kedua yang ada di dalam SOLID Principle. Dimana prinsip ini bertujuan untuk memberikan pengertian terhadap perubahan yang terjadi pada kelas, diharapkan perubahan tersebut tidak secara menyeluruh. Hal ini tentunya menjelaskan bahwa kode yang telah kita buat sebelumnya tidak sesuai dengan prinsip ini. Oleh karena itu, gimana caranya kita bisa memperbaiki kode sebelumnya? Mari kita lihat kembali kode sebelumnya :

```
def cekNama(nama):
    for kar in nama:
        if(not(ord("A") <= ord(kar) <= ord("Z") or ord("a") <= ord(kar) <= ord("z") or kar == " ")):
            return False
    return True

def simpanData(nama):
    if(cekNama(nama)):
        print(f"{nama} berhasil disimpan ke dalam data\n")
    else:
        print("Mohon maaf nama tidak bisa disimpan ya ... !!!\n")

def loginNama(nama):
    if(cekNama(nama)):
        print(f"{nama} sesuai dengan ketentuan yang ada dan bisa login.\n")
    else:
        print("Mohon maaf nama tidak sesuai, sehingga Anda tidak bisa login.\n")
```

Figure 163 Contoh OCP (1)

Terus, untuk kode bagian mainnya :

```
if __name__ == "__main__":
    simpanData("Apriyanto Halim")
    loginNama("Apriyanto Halim")

    simpanData("S1@p@ y@ng t@hu")
    loginNama("S1@p@ y@ng t@hu")

    simpanData("AnEh sAjA sIh")
    loginNama("AnEh sAjA sIh")
```

Figure 164 Contoh OCP (2)

Untuk menyelesaikan permasalahan ini, terlebih dahulu kita buat terlebih dahulu kelas abstract yang nanti akan dijadikan sebagai kelas utama seperti kode berikut :

```
from abc import ABC, abstractmethod
def cekNama(nama):
    for kar in nama:
        if(not(ord("A") <= ord(kar) <= ord("Z") or ord("a") <= ord(kar) <= ord("z") or kar == " ")):
            return False
    return True

class absHasil(ABC):
    @abstractmethod
    def hasil():
        pass
```

Figure 165 Contoh OCP (3)

Pada kode tersebut, saya buat 1 buah kelas utama yang merupakan kelas abstract absHasil yang nantinya akan dijadikan sebagai kelas utama untuk melakan proses perulangan. Setelah itu, saya buat kembali fungsi untuk pengecekan. Setelah kedua proses tersebut telah ada, berikutnya saya buat proses untuk simpan dan login seperti tampilan berikut :

```
class simpanData(absHasil):
    def hasil(nama):
        if(cekNama(nama)):
            print(f"{nama} berhasil disimpan ke dalam data")
        else:
            print("Mohon maaf nama tidak bisa disimpan ya ... !!!")

class loginNama(absHasil):
    def hasil(nama):
        if(cekNama(nama)):
            print(f"{nama} sesuai dengan ketentuan yang ada dan bisa login.\n")
        else:
            print("Mohon maaf nama tidak sesuai, sehingga Anda tidak bisa login.\n")
```

Figure 166 Contoh OCP (4)

Dari kelas tersebut terlihat bahwa setiap kelas bergantungan dengan kelas utama, yaitu kelas absHasil yang telah kita buat sebelumnya. Setelah itu, kita baru lanjutkan ke dalam pembuatan kelas utamanya menjadi seperti ini :

```
if __name__ == "__main__":
    nama = "Apriyanto Halim"
    for proses in absHasil.__subclasses__():
        proses.hasil(nama)

    nama = "S1@p@ y@ng t@hu"
    for proses in absHasil.__subclasses__():
        proses.hasil(nama)

    nama = "AnEh sAjA sIh"
    for proses in absHasil.__subclasses__():
        proses.hasil(nama)
```

Figure 167 Contoh OCP (5)

Pada kelas utama saya ada memanggil 1 buah fungsi bawaan, yaitu subclasses yang berfungsi untuk memanggil semua kelas bawahannya dari kelas absHasil. Setelah itu, pada bagian prosesnya, kita tinggal panggil method hasil, yang merupakan method yang ada pada kelas utama.

Tentunya bagian ini sangat membantu kita dalam membuat atau menambahkan fitur baru yang terdapat di dalam aplikasi yang kita buat. Nah setelah itu, mari kita kembali ke dalam permasalahan kita!

SOLUTION

Setelah kita baca kembali studi kasus tersebut, yang menjadi permasalahan kita adalah terdapat perbedaan pendapat dan adanya kebutuhan terkait perubahan akibat dari perkembangan zaman. Oleh karena itu diperlukan sebuah solusi yang bisa mengatasi permasalahan tersebut. Dari hasil pembelajaran yang telah kita pelajari sebelumnya, untuk mengatasi masalah tersebut mungkin kita bisa menggunakan prinsip dan SRP dan OCP. Dimana kedua prinsip ini dapat membantu kita untuk menyelesaikan masalah tersebut. Namun, gimana cara penerapannya ke dalam kode?

Nah supaya lebih jelas, yuk kita bahas 1 per 1.

INSTRUCTION

- Untuk kode sebelumnya, sebenarnya untuk proses ini kita telah menggunakan facadePattern untuk mengatasinya, hanya saja proses tersebut mungkin bisa dibilang masih belum sesuai dengan yang pada prinsip OCP. Dimana kodennya seperti berikut :

```
class cekTahun(BUKU):
    def __init__(self, proses):
        self._proses = proses
    def cetakBuku(self):
        pass

class cekTahunManajemen(cekTahun):
    def cetakBuku(self):
        thn = 2021
        hasil = self._proses.cetakBuku()
        hasil += "Buku termasuk buku {}".format("Baru" if (thn - self._proses.tahun) > 5 else "Lama")
        hasil += " dimana buku terbit pada tahun {} \n".format(self._proses.tahun)
        return hasil

class cekTahunPemilik(cekTahun):
    def cetakBuku(self):
        thn = 2021
        hasil = self._proses.cetakBuku()
        hasil += "Buku terbit pada tahun {} ".format(self._proses.tahun)
        hasil += " sehingga, termasuk buku {} \n".format("Baru" if (thn - self._proses.tahun) > 5 else "Lama")
        return hasil

class facadeCekTahun:
    def __init__(self, buku):
        self.denganPemilik = cekTahunPemilik(buku)
        self.denganManajemen = cekTahunManajemen(buku)
    def proses(self):
        print("Saran dari \"Pemilik\" =")
```

```
print("Dengan Cek Tahun :\n", self.denganPemilik.cetakBuku())
print("\nSaran dari \"Manajemen\" =")
print("Dengan Cek Tahun :\n", self.denganManajemen.cetakBuku())
print()
```

2. Oleh karen itu, kita ubah menjadi seperti berikut :

```
class cekTahun(ABC):
    @abstractmethod
    def cetakBuku():
        pass

class cekTahunManajemen(cekTahun):
    def cetakBuku(buku):
        thn = 2021
        hasil = buku.cetakBuku()
        hasil += "Buku termasuk buku {}".format("Baru" if (thn - buku.tahun) > 5 else "Lama")
        hasil += " dimana buku terbit pada tahun {} \n".format(buku.tahun)
        return hasil

class cekTahunPemilik(cekTahun):
    def cetakBuku(buku):
        thn = 2021
        hasil = buku.cetakBuku()
        hasil += "Buku terbit pada tahun {} ".format(buku.tahun)
        hasil += " sehingga, termasuk buku {} \n".format("Baru" if (thn - buku.tahun) > 5 else "Lama")
        return hasil
```

Namun, hal ini dapat mengakibat terdapat kesalahan pada kode seperti berikut :

```
cekThn = facadeCekTahun(buku1)
```

3. Oleh karen itu, kita ubah juga menjadi seperti berikut :

```
for cetak in cekTahun.__subclasses__():
    cetak.cetakBuku(buku1)
    denganCekJumlah = cekStok(buku1)
```

Gimana? Mudah bukan? Jadi tidak begitu ribet, dan apabila ada keperluan baru terkait perubahan, silahkan langsung ditambahkan dan diproses langsung!

EXERCISE

EXERCISE OBJECTIVES

Pada latihan berikut ini mahasiswa diharapkan dapat melakukan :

- Melakukan analisis masalah terhadap studi kasus yang diberikan.
- Mencari solusi atas masalah yang diberikan.
- Menerjemahkan masalah tersebut ke dalam bentuk kode pemrograman.

TASK 1:

Pada sebuah kelas di Kampus Mikroskil terdapat mahasiswa dan dosen pada kelas tersebut. Mahasiswa tersebut tentunya memiliki data berupa (*bagian ini kalian cari tahu sendiri*). Sedangkan dosen memiliki data berupa NIP, Nama, jabatan, jenis kelamin, dan no HP. Pada suatu acara, seorang mahasiswa ditujukan untuk melakukan rekap absensi dari acara tersebut. Absensi tersebut nantinya harus bisa dicetak serta bisa dihitung kira-kira berapa banyak jumlah pengunjung pada acara tersebut. Bisakah kalian membantu mahasiswa tersebut?

Ketentuan yang dilakukan pada task-01 ini adalah :

- a. Gunakan kode untuk task-01 pertemuan-01.
- b. Silahkan kalian terapkan prinsip S dan O pada bagian SOLID untuk kode sehingga kode yang dibuat menjadi lebih jelas.
- c. Namun, apabila kode kalian rasa sudah menerapkan atau tidak memerlukan prinsip tersebut, kalian bisa berikan alasan tersebut pada kode kalian dengan menambahkan komentar pada bagian akhirnya.

UNIT 8

PRINSIP SOLID PADA PYTHON (2)

UNIT OVERVIEW

Pada pertemuan kali ini kita kembali membahas terkait prinsip SOLID pada python. Dimana dijelaskan sebelumnya prinsip ini tidak harus diterapkan secara keseluruhan. Kalian, hanya perlu memasukkan beberapa bagian saja supaya ketika terjadi pengembangan pada perangkat lunak kalian bisa langsung mengatasi secara langsung dan sangat baik.

UNIT OBJECTIVES

Adapun capaian yang akan kita dapatkan pada modul ini, yaitu :

- Cara menerapkan prinsip Liskov Substitution Principle (SRP).
- Cara menerapkan prinsip Interface Segregation Principle (ISP)
- Cara menerapkan prinsip Dependency Inversion Principle (DIP)

UNIT CONTENTS

Lesson 3: Liskov Substitution Principle	132-
135	
Lesson 4: Interface Segregation Principle	135-
136	
Lesson 5: Dependency Inversion Principle.....	136-
138	

PRE LAB

Sebelum masuk ke dalam lab pertama ini, terlebih dahulu kita coba jawab beberapa pertanyaan ini berdasarkan hasil penjelasan pada modul teori!

QUESTION

1. Apa tujuan diterapkan prinsip SOLID?
2. Menurut kalian, apa fungsi dari prinsip Open-Closed Principle?
3. Dari permasalahan berikut, manakah yang dapat diselesaikan dengan SRP atau OCP :
 - Pembuatan dua proses yang tidak saling bergantungan.
 - Pemanggilan beberapa kelas yang memiliki kelas utama yang sama.

CONTENT LESSON

CASE STUDY / PROJECT

Pada perpustakaan “Home Learning is Best” telah dibuat terkait permasalahan dalam perbedaan pendapat tersebut. Namun, mungkin menjadi permasalahan baru terhadap kemungkinannya terjadinya proses ketergantungan terhadap tingkat proses yang lebih tinggi terhadap tingkat proses yang lebih rendah.

Dari hasil yang kita ketahui tentunya hal ini tidak bisa. Karena, proses sebelumnya sudah fix. Jadi, gimana cara mengatasinya? Sebelum itu, mari kita cari tahu dulu permasalahan yang ada ... !!!

IDENTIFICATION CONCEPT OF PROBLEM / PROJECT

Permasalahan yang muncul pada proses tersebut adalah ketakutan terhadap proses ketergantungan pada kode yang dibuat. Hal ini tentunya dapat mengakibatkan proses yang kompleks menjadi tidak berfungsi apabila terjadi perubahan pada proses yang memiliki proses yang lebih rendah. Gimana cara mengetahuinya dan apakah kode yang kita buat sudah dapat mengatasi masalah tersebut?

Sebelum menyelesaikan permasalahan tersebut, mari kita belajar terlebih dahulu ... !!!

LESSON 3: LISKOV SUBSTITUTION PRINCIPLE

LESSON 3: OVERVIEW

Liskov substitution principle (LSP) merupakan sebuah prinsip yang memperbolehkan subclass bisa diganti dengan super class atau bisa dibilang sebagai kelas anak bisa diganti menjadi kelas induk. Terus, kalau tidak bisa diganti apa yang akan terjadi?

Untuk menjawab permasalahan tersebut, mungkin kita perlu perhatikan kode berikut :

```
class Vehicle:  
    """A demo Vehicle class"""\n\n    def __init__(self, name: str, speed: float):  
        self.name = name  
        self.speed = speed\n\n    def get_name(self) -> str:  
        """Get vehicle name"""\n        return f"The vehicle name {self.name}"\n\n    def get_speed(self) -> str:  
        """Get vehicle speed"""\n        return f"The vehicle speed {self.speed}"\n\n    def engine(self):  
        """A vehicle engine"""\n        pass\n\n    def start_engine(self):  
        """A vehicle engine start"""\n        self.engine()
```

Figure 168 Contoh LSP (1)

Pada kelas tersebut terlebih dahulu kita deklarasikan abstract class yang nantinya akan dijadikan sebagai proses untuk subclassnya. Setelah itu, kita kembali membuat untuk subclassnya seperti berikut :

```
class Car(Vehicle):  
    """A demo Car Vehicle class"""\n    def start_engine(self):  
        pass\n\n\nclass Bicycle(Vehicle):  
    """A demo Bicycle Vehicle class"""\n    def start_engine(self):  
        pass
```

Figure 169 Contoh LSP (2)

Pada kelas tersebut sekilas tidak ada yang aneh. Hanya saja ketika kita lihat kembali pada kelas Bicycle (sepeda) memang merupakan vehicle (kendaraan), tapi kelas tersebut tidak memiliki start_engine (mesin). Oleh karena itu, tentunya kelas ini memang bisa jalan, hanya saja tidak sesuai dengan yang kita harapkan.

Oleh karena itu, kita perlu mengubah sedikit pada bagian kelas utamanya menjadi seperti berikut :

```
class Vehicle:  
    """A demo Vehicle class"""\n    def __init__(self, name: str, speed: float):\n        self.name = name\n        self.speed = speed\n    def get_name(self) -> str:\n        """Get vehicle name"""\n        return f"The vehicle name {self.name}"\n    def get_speed(self) -> str:\n        """Get vehicle speed"""\n        return f"The vehicle speed {self.speed}"
```

Figure 170 Contoh LSP (3)

Pada kelas utama tersebut tidak ada lagi fungsi start_engine (mesin). Setelah itu, saya tambahkan kelas berikut :

```
class VehicleWithoutEngine(Vehicle):\n    """A demo Vehicle without engine class"""\n    def start_moving(self):\n        """Moving"""\n        raise NotImplemented\n\nclass VehicleWithEngine(Vehicle):\n    """A demo Vehicle engine class"""\n    def engine(self):\n        """A vehicle engine"""\n        pass\n    def start_engine(self):\n        """A vehicle engine start"""\n        self.engine()
```

Figure 171 Contoh LSP (4)

Tujuan dari kelas ini adalah supaya nanti kita bisa membedakan, antara kendaraan yang memiliki mesin dan yang tidak. Karena, tidak semua kendaraan memiliki mesin. Setelah itu, baru kita tambahkan kelas berikut :

```
class Car(VehicleWithEngine):\n    """A demo Car Vehicle class"""\n    def start_engine(self):\n        pass\n\nclass Bicycle(VehicleWithoutEngine):\n    """A demo Bicycle Vehicle class"""\n    def start_moving(self):\n        pass
```

Figure 172 Contoh LSP (5)

Dari sini terlihat bahwa, car (mobil) memang memiliki mesin sehingga kita tambahkan mesin. Sedangkan untuk kelas Bicycle (sepeda) tidak memiliki mesin, sehingga kita tidak perlu tambahkan mesin disana. Hal ini tentunya membuat setiap kelas yang awal kita buat sesuai dengan apa yang kita harapkan.

Mari kita lanjut ke pelajaran selanjutnya ... !!!

LESSON 4: INTERFACE SEGREGATION PRINCIPLE

LESSON 4: OVERVIEW

Interface segregation principle (ISP) merupakan sebuah kelas yang bertujuan untuk menspesifikasi kegunaan dari sebuah kelas. Hal ini bertujuan supaya hasil yang diharapkan sesuai dengan yang ada diawal. Sebagai contoh perhatikan kode berikut :

```
from abc import ABC, abstractmethod
class Laundry:
    @abstractmethod
    def wash(self):
        pass
    @abstractmethod
    def dry(self):
        pass

class expertMachine(Laundry):
    def wash(self):
        # Mengerjakan proses mencuci
        pass
    def dry(self):
        # Mengerjakan proses pengeringan
        pass

class simpleMachine(Laundry):
    def wash(self):
        # Mengerjakan proses mencuci
        pass
```

Figure 173 Contoh ISP (1)

Pada contoh kode tersebut, terlihat bahwa terdapat sebuah kelas utama, yaitu Laundry. Dimana pada kelas utama tersebut berisi proses-proses apa saja yang dilakukan pada proses Laundry dimulai dari mencuci hingga proses pengeringan. Setelah itu ada terdapat dua buah mesin baru yang baru dibeli. Dimana kedua mesin tersebut, yaitu expertMachine yang melakukan semua proses pada kelas Laundry. Sedangkan satu kelas lagi, yaitu simpleMachine yang hanya mengerjakan proses pencucian saja tanpa melakukan proses pengeringan.

Secara berjalannya program, tentunya hal ini tidaklah masalah. Hanya saja, ketika saya jalankan proses pencucian pada kelas simpleMachine, hal ini tetap dapat dilakukan. Sehingga, proses ini menjadi tidak sesuai dengan harapan awal. Jadi, gimana solusinya?

Salah satu solusi yang bisa kita tawarkan adalah dengan menggunakan prinsip ISP. Dimana setiap kriteria dibagi sesuai dengan kebutuhan seperti berikut :

```
from abc import ABC, abstractmethod
class washing:
    @abstractmethod
    def wash(self):
        pass

class drying:
    @abstractmethod
    def dry(self):
        pass
```

Figure 174 Contoh ISP (2)

Setelah itu, baru kita buat kelas turunannya menjadi seperti berikut :

```
class expertMachine(washing, drying):
    def wash(self):
        # Mengerjakan proses mencuci
        pass
    def dry(self):
        # Mengerjakan proses pengeringan
        pass

class simpleMachine(washing):
    def wash(self):
        # Mengerjakan proses mencuci
        pass
```

Figure 175 Contoh ISP (3)

Tentunya hal ini dapat menjadi lebih efektif ketimbang yang telah kita buat sebelumnya. Dan apabila kita memanggil proses pengeringan pada kelas simpleMachine, maka proses itu akan dianggap tidak ada atau tidak bisa diproses. Gimana? Mudah bukan?

Mari kita lanjut ke pelajaran selanjutnya !

LESSON 5: DEPENDENCY INVERSION PRINCIPLE

LESSON 5: OVERVIEW

Dependency inversion principle (DIP) merupakan prinsip yang berfokus dimana kelas yang memiliki kompleksitas yang lebih tinggi tidak boleh bergantung kepada kelas yang memiliki kompleksitas yang rendah. Hal ini bertujuan supaya proses yang ada pada kelas yang memiliki kompleksitas yang tinggi, tidak terganggu dan dapat berjalan sesuai dengan yang kita harapkan.

Sebagai contoh, perhatikan kode berikut :

```
class BackendDeveloper:  
    """This is a low-level module"""  
    @staticmethod  
    def python():  
        print("Writing Python code")  
  
class FrontendDeveloper:  
    """This is a low-level module"""  
    @staticmethod  
    def javascript():  
        print("Writing JavaScript code")  
  
class Project:  
    """This is a high-level module"""  
    def __init__(self):  
        self.backend = BackendDeveloper()  
        self.frontend = FrontendDeveloper()  
    def develop(self):  
        self.backend.python()  
        self.frontend.javascript()  
        return "Develop codebase"  
  
project = Project()  
print(project.develop())
```

Figure 176 Contoh DIP (1)

Pada kode tersebut, mungkin bisa kita bilang Project merupakan kelas yang memiliki tingkat kompleksitas yang tinggi, sedangkan untuk kelas Backend dan Frontend memiliki tingkat kompleksitas yang rendah. Namun, pada proses tersebut kelas project sangat bergantung terhadap kelas frontend dan backend. Hal ini dikarenakan, ketika proses proses pada bagian backend dan frontend tidak berjalan, maka proses pengembangan tidak dapat dilakukan dan hasilnya project tidak berhasil dibuat. Tentunya ini menjadi sebuah permasalahan ketika kita membuat sebuah kode.

Oleh karena itu, hal yang harus kita lakukan untuk mengatasi masalah tersebut adalah memisah bagian kode yang bergantungan tersebut. Dimana pada proses tersebut, yaitu proses develop (pengembangan).

Oleh karena itu, kita sedikit ubah kelasnya menjadi seperti berikut :

```
class BackendDeveloper:  
    """This is a low-level module"""  
    def develop(self):  
        self._python_code()  
    @staticmethod  
    def __python_code():  
        print("Writing Python code")  
  
class FrontendDeveloper:  
    """This is a low-level module"""  
    def develop(self):  
        self._javascript()  
    @staticmethod  
    def __javascript():  
        print("Writing JavaScript code")  
  
class Developers:  
    """An Abstract module"""  
    def __init__(self):  
        self.backend = BackendDeveloper()  
        self.frontend = FrontendDeveloper()  
    def develop(self):  
        self.backend.develop()  
        self.frontend.develop()  
  
class Project:  
    """This is a high-level module"""  
    def __init__(self):  
        self.__developers = Developers()  
    def develops(self):  
        return self.__developers.develop()  
  
project = Project()  
print(project.develops())
```

Figure 177 Contoh DIP (2)

Pada kode tersebut, meskipun kalian sedikit berbeda pada bagian frontend dan backendnya, tetapi prinsipnya tetap sama. Hanya saja pada bagian project terlihat tidak begitu tergantung lagi terhadap frontend dan backend, tetapi lebih tergantung terhadap developer. Hal ini tentunya dapat membuat kode tersebut menjadi lebih baik.

Setelah kita belajar beberapa pelajaran ini, mari kita selesaikan permasalahan yang kita hadapi pada studi kasus sebelumnya!

SOLUTION

Pada studi kasus tersebut terdapat ketakutan terhadap proses yang saling bergantung. Apalagi kalau proses bergantungan tersebut melibatkan proses yang memiliki tingkat kinerja atau kompleksitas yang tinggi. Hal ini tentunya bisa kita selesaikan dengan menggunakan prinsip DIP. Hanya saja, kita harus pastikan kembali, apakah proses tersebut sudah berfungsi dengan baik?

Mari kita cari tahu terhadap kode sebelumnya ... !!!

INSTRUCTION

1. Berikut merupakan kode lengkap yang telah kita buat pada pertemuan 14 sebelumnya :
from abc import ABC, abstractmethod

```
class BUKU:  
    def __init__(self, nama, tahun, jlh):  
        self.nama = nama  
        self.tahun = tahun  
        self.jlh = jlh
```

```

def cetakBuku(self):
    hasil = "Nama Buku = {}\\n".format(self.nama)
    hasil += "Tahun Buku = {}\\n".format(self.tahun)
    hasil += "Jumlah Buku = {}\\n".format(self.jlh)
    return hasil

class cekTahun(ABC):
    @abstractmethod
    def cetakBuku():
        pass

class cekTahunManajemen(cekTahun):
    def cetakBuku(buku):
        thn = 2021
        hasil = buku.cetakBuku()
        hasil += "Buku termasuk buku {}".format("Baru" if (thn - buku.tahun) > 5 else "Lama")
        hasil += " dimana buku terbit pada tahun {}\\n".format(buku.tahun)
        return hasil

class cekTahunPemilik(cekTahun):
    def cetakBuku(buku):
        thn = 2021
        hasil = buku.cetakBuku()
        hasil += "Buku terbit pada tahun {} ".format(buku.tahun)
        hasil += "sehingga, termasuk buku {}\\n".format("Baru" if (thn - buku.tahun) > 5 else "Lama")
        return hasil

class cekStok(BUKU):
    def __init__(self, proses):
        self._proses = proses
    def cetakBuku(self):
        hasil = self._proses.cetakBuku()
        hasil += "Perlu {}\\n".format("Mencari Buku" if self._proses.jlh < 3 else "Mencari Pembaca")
        return hasil

#Abstract Factory Design
class kartuBuku(ABC):
    @abstractmethod
    def cetakKartu(self):
        pass

class kodeBuku(kartuBuku):
    def cetakKartu(self, buku):
        print(f"Kode Buku = {buku.nama[0:3] + str(buku.tahun)}")

class kartuBuku(kartuBuku):
    def cetakKartu(self, buku):
        print(f"Nama Buku = {buku.nama}")
        print(f"Jumlah Buku = {buku.jlh}")

```

```

class factoryKartuBuku(ABC):
    @abstractmethod
    def cetakKode(self):
        pass
    @abstractmethod
    def cetakData(self):
        pass

class afdKartuBuku(factoryKartuBuku):
    def cetakKode(self):
        return kodeBuku()
    def cetakData(self):
        return kartuBuku()

if __name__ == '__main__':
    buku1 = BUKU("Pemrograman Python", 2019, 3)
    for cetak in cekTahun.__subclasses__():
        cetak.cetakBuku(buku1)
    denganCekJumlah = cekStok(buku1)

    print("Awal :\n", buku1.cetakBuku())
    print()
    #cekThn.proses()
    print("Dengan Cek Jumlah:\n", denganCekJumlah.cetakBuku())

    ctkKrt = afdKartuBuku()
    kdBuku = ctkKrt.cetakKode()
    dataBuku = ctkKrt.cetakData()
    print("\nKartu Buku")
    kdBuku.cetakKartu(buku1)
    dataBuku.cetakKartu(buku1)

```

Pada kode tersebut, ada bagian yang harus kita perhatikan, yaitu pada bagian yang diberikan tanda warna merah.

Proses yang diberikan tanda merah, merupakan proses main yang merupakan proses paling penting atau memiliki kompleksitas yang paling tinggi.

2. Pada proses main tersebut, kita melihat adanya proses :

```

for cetak in cekTahun.__subclasses__():
    cetak.cetakBuku(buku1)

```

Proses tersebut merupakan bagian dari pada proses yang telah kita buat pada pertemuan sebelumnya. Dan proses tersebut nantinya akan memanggil kelas ini :

```

class cekTahun(ABC):
    @abstractmethod
    def cetakBuku():
        pass

class cekTahunManajemen(cekTahun):
    def cetakBuku(buku):
        thn = 2021

```

```

hasil = buku.cetakBuku()
hasil += "Buku termasuk buku {}".format("Baru" if (thn - buku.tahun) > 5 else "Lama")
hasil += " dimana buku terbit pada tahun {}\\n".format(buku.tahun)
return hasil

class cekTahunPemilik(cekTahun):
    def cetakBuku(buku):
        thn = 2021
        hasil = buku.cetakBuku()
        hasil += "Buku terbit pada tahun {} ".format(buku.tahun)
        hasil += "sehingga, termasuk buku {}\\n".format("Baru" if (thn - buku.tahun) > 5 else "Lama")
        return hasil

```

Semisalnya, pada proses `cekTahunPemilik` tidak berjalan dengan baik, apakah dapat mempengaruhi proses tersebut?

Jawabanya adalah **TIDAK**. Oleh karena itu, kita sebenarnya sudah bisa menerapkan prinsip DIP ke dalam kode.

EXERCISE

EXERCISE OBJECTIVES

Pada latihan berikut ini mahasiswa diharapkan dapat melakukan :

- Melakukan analisis masalah terhadap studi kasus yang diberikan.
- Mencari solusi atas masalah yang diberikan.
- Menerjemahkan masalah tersebut ke dalam bentuk kode pemrograman.

TASK 1:

Pada sebuah kelas di Kampus Mikroskil terdapat mahasiswa dan dosen pada kelas tersebut. Mahasiswa tersebut tentunya memiliki data berupa (*bagian ini kalian cari tahu sendiri*). Sedangkan dosen memiliki data berupa NIP, Nama, jabatan, jenis kelamin, dan no HP. Pada suatu acara, seorang mahasiswa ditunjukan untuk melakukan rekap absensi dari acara tersebut. Absensi tersebut nantinya harus bisa dicetak serta bisa dihitung kira-kira berapa banyak jumlah pengunjung pada acara tersebut. Bisakah kalian membantu mahasiswa tersebut?

Ketentuan yang dilakukan pada task-01 ini adalah :

- a. Gunakan kode untuk task-01 pertemuan-01.
- b. Silahkan kalian pahami kode tersebut, apakah kode tersebut sudah menerapkan prinsip SOLID dengan baik? Jika belum, kenapa dan jelaskan alasannya!



UNIVERSITAS
MIKROSKIL