Modul 2

Mengenal OOP pada Python

Di bab ini akan kita bahas sekilas pemrograman berorientasi obyek dengan Python. Pemrograman berorientasi obyek (*object-oriented programming*, OOP) adalah sebuah konsep *powerful* yang berguna dalam pembuatan program secara modular.

Sebelum membahas lebih jauh topik OOP, kita akan membahas topik module terlebih dahulu.

2.1 Module

Module secara mudah bisa dipahami sebagai kumpulan prosedur dan nilai yang tersimpan dalam satu atau beberapa file, yang bisa diakses dengan meng-`import`-nya. Perhatikan contoh berikut.

Latihan 2.1 Sebuah module sederhana. Ketik dan simpan file `.py` berikut.

ModulePythonPertamaku.py

```
1 def ucapkanSalam():
2     print("Assalamu 'alaikum!")
3
4 def kuadratkan(x):
5     return x*x
6
7 buah = 'Mangga'
8 daftarBaju = ['batik','loreng','resmi berdasi']
9 jumlahBaju = len(daftarBaju)
```

Kembalilah ke Python Shell (JANGAN larikan programnya). Di Python Shell, ketikkan perintah import seperti berikut

```
>>> import ModulePythonPertamaku
```

Kalau berhasil, maka fungsi dan variabel di file `ModulePythonPertamaku.py` sudah termuat ke memori. Sekarang contohlah yang berikut ini

```
>>> ModulPythonPertamaku.ucapkanSalam()  # INGAT tanda kurungnya
Assalamu 'alaikum!
>>> ModulPythonPertamaku.kuadratkan(5)
25
```

```
>>> ModulPythonPertamaku.buah
'Mangga'
>>>
```

□

Dapat dilihat bahwa sebuah module mengelompokkan fungsi dan variabel dalam satu ikatan. "Ruang-nama"-nya (*name-space*) berada di ModulPythonPertamaku. Nama modulnya dibawa ke ruang-nama lokal. Sehingga untuk mengakses metode/fungsinya kita harus mengikutsertakan nama modulnya.

Menulis "`ModulPythonPertamaku`" setiap saat tampaknya bikin capek. Bisakah kita menyingkatnya? Python bisa. Seperti ini.

```
>>> import ModulPythonPertamaku as mpp
>>> mpp.ucapkanSalam()
Assalamu 'alaikum!
>>> mpp.daftarBaju
['batik', 'loreng', 'resmi berdasi']
>>> mpp.jumlahBaju
3
```

Ruang-namanya sekarang adalah[1] `mpp`.

Bagaimana kalau dari sekian ratus metode yang ditawarkan suatu modul, kita hanya memerlukan beberapa saja? Kita bisa mengambil sesuai keperluan.

```
>>> from ModulPythonPertamaku import kuadratkan, daftarBaju
>>> kuadratkan(6)
36
>>> daftarBaju
['batik', 'loreng', 'resmi berdasi']
```

Penting: perhatikan bahwa fungsi dan variabel yang diimport sekarang berada di ruang-nama lokal. Yakni kita bisa *memanggilnya secara langsung*.

Kita dapat juga memberi nama lain pada metode yang diimport

```
>>> from ModulPythonPertamaku import ucapkanSalam as ucap
>>> ucap()
Assalamu 'alaikum!
```

Kita dapat meng-import seluruh metode dan variabel di suatu module. Silakan coba yang berikut

```
dir()               # Melihat isi ruangnama lokal
import math as m    # Mengimpor math sebagai m
dir()               # Lihat lagi. Pastikan ada module 'm' di sana
dir(m)              # Melihat isi module 'm'
from sys import *   # Mengimpot semua yang di 'sys' ke ruangnama lokal
dir()               # Melihat isi ruangnama lokal lagi.
```

[1] `import xxx as yy` berguna tidak terutama pada kenyamanan menyingkatnya, tapi pada konsep ruang-nama, *namespace*.

2.2 Class dan Object

Pada bagian ini kita akan mempelajari konsep class dan object pada Python.

Di dalam Python, sebuah *class*[2] adalah sebuah konsep atau cetak biru mengenai 'sesuatu' (umumnya kata benda). Sebuah *object*[3] adalah 'sebuah class yang mewujud'[4].

Ketika sebuah class X akan diwujudkan menjadi sebuah (atau beberapa) object di memori, berarti class X itu di-instantiasi menjadi object a,b,c, dan seterusnya. Sering pula dikatakan bahwa object a,b,c adalah *instance* dari class X. Beberapa contoh akan membantu.

- Ada class Manusia. Instance dari class Manusia misalnya: mbak Sri temanmu, mas Joko tetanggamu, si Janto sepupumu, bu Ratih gurumu. (Mereka adalah object 'yang dibangkitkan' dari class yang sama). Lebih jauh lagi, tiap instance mempunyai hal-hal yang sama. Katakanlah: setiap Manusia mempunyai nama, setiap Manusia mempunyai metode 'ucapkanSalam'.

  - Class Mahasiswa bisa dibuat dengan landasan class Manusia di atas.

  - Lebih jauh lagi, class MhsTIF bisa dibuat dengan landasan class Mahasiswa barusan.

  Dalam *object-oriented programming*, ini disebut *inheritance* atau pewarisan. Semua object yang di-instantiasi dari class Mahasiswa akan mewarisi metode dan state yag dimiliki class Manusia. Semua object yang di-instantiasi dari class MhsTIF akan mewarisi metode dan state yag dimiliki class Mahasiswa dan yang dimiliki class Manusia.

- Ada class SepedaMotor (atau 'konsep', atau 'ide tentang' SepedaMotor). Instance dari class ini misalnya: sepeda motor yang kamu miliki dan semua sepeda motor yang sekarang sedang berjalan di jalanraya. Jadi, class SepedaMotor itu di-instantiasi menjadi, salah satunya, object sepeda motor yang kamu naiki.

- Ada class Pesan. Instance dari class ini adalah pesan1 dengan isi pesan 'Aku suka kuliah ini' dan pesan2 dengan isi pesan 'Aku senang struktur data'.

Sekarang mari kita tinjau beberapa contoh

Latihan 2.2 Sebuah kelas sederhana: `Pesan`. Ketik[5] dan simpan.

[2] Secara resmi Bahasa Indonesia punya kata 'kelas', namun untuk kejelasan paparan, kita memakai kata 'class' di sini.

[3] Bahasa Indonesia punya kata 'objek' dan 'obyek' (maknanya sama). Tapi sekali lagi untuk kejelasan paparan dan pengkodean, kita pakai kata 'object' di sini.

[4] Ini mungkin mirip istilah di matakuliah filsafat, tapi percayalah, idenya sebenarnya tidak rumit.

[5] Jika ada kata yang terasa asing seperti `self` dan `__init__`, ini akan dijelaskan kemudian

LatOOP2.py

```
class Pesan(object):
    """
       Sebuah class bernama Pesan.
       Untuk memahami konsep Class dan Object.
    """
    def __init__(self, sebuahString):
        self.teks = sebuahString
    def cetakIni(self):
        print(self.teks)
    def cetakPakaiHurufKapital(self):
        print(str.upper(self.teks))
    def cetakPakaiHurufKecil(self):
        print(str.lower(self.teks))
    def jumKar(self):
        return len(self.teks)
    def cetakJumlahKarakterku(self):
        print('Kalimatku mempunyai',len(self.teks),'karakter.')
    def perbarui(self,stringBaru):
        self.teks = stringBaru
```

Jalankan programnya dengan memencet F5 atau meng-`import`-nya. Maka class Pesan akan sudah berada di memori dan siap di-instantiasi. Cobalah menginstansiasi ('membuat object dari') class Pesan di atas seperti berikut[6]

```
>>> pesanA = Pesan('Aku suka kuliah ini')
>>> pesanB = Pesan('Surakarta: the Spirit of Java')
```

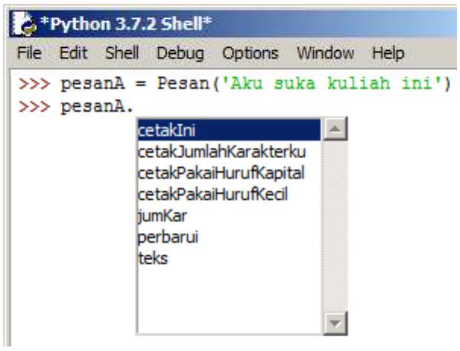
Yang barusan kamu lakukan adalah membuat dua buah object [dari class] `Pesan`, yakni `pesanA` dan `pesanB`. Object (atau variabel) ini siap dimanfaatkan sebagaimana object-object yang lain yang bertipe, misal, `int`, `str`, `float`, `bool`. Ketik yang berikut

```
>>> pesanA.cetakIni()
Aku suka kuliah ini
>>> pesanA.cetakJumlahKarakterku()
Kalimatku mempunyai 19 karakter.
>>> pesanB.cetakJumlahKarakterku()
Kalimatku mempunyai 29 karakter.
>>> pesanA.cetakPakaiHurufKapital()
AKU SUKA KULIAH INI
>>> pesanA.cetakPakaiHurufKecil()
aku suka kuliah ini
>>> pesanA.perbarui('Aku senang struktur data')
>>> pesanA.cetakIni()
Aku senang struktur data
```

□

Cool, isn't it? Jadi dengan class kita bisa membuat tipe data baru yang juga mempunyai metode-metode, seperti halnya tipe data bawaan. Lihat Gambar 2.1. Akan kita perjelas sedikit kejadian pembuatan class dan pemwujudan class `Pesan` di atas.

[6] Jika kamu meng-`import`-nya, boleh jadi cara memanggilnya berbeda. Tergantung caramu meng`import`.



Gambar 2.1: `pesanA` adalah sebuah object yang mempunyai metode-metode. Ketik `pesanA.` (jangan lupa titiknya), lalu pencet `Ctrl` + `Space` untuk melihat efeknya (di sini saya memakai IDLEX).

- Sebuah class dibuat dengan dimulai dengan kata kunci `class`, diikuti nama class-nya dengan parameter 'kelas induk'. Untuk contoh di atas kelas induknya adalah `object`.
- Sebuah class umumnya mempunyai data (seperti `teks` pada contoh di atas) dan metode (seperti `cetakJumlahKarakterku()` pada contoh di atas). Data dan metode ini bisa satu, bisa banyak, dan bahkan bisa tidak ada sama sekali.
- Penulisan metode sama dengan penulisan fungsi pada umumnya. Metode pada dasarnya adalah fungsi yang diikat pada sebuah class.
- Terdapat beberapa data dan metode khusus yang ditandai dengan awalan dan akhiran dua garis datar. Dua di antaranya:
  - `__init__()`. Ini adalah *constructor*. Ketika pembuatan object terjadi, metode inilah (kalau ada) yang dipanggil.
  - `__doc__`. Ini adalah dokumentasi. Coba ketik ini: `pesanA.__doc__`
- **Sebuah object di-instantiasi dengan mengetik nama class-nya**[7], dengan parameter sesuai dengan yang ada di metode `__init__()`.
  - `pesanA = Pesan('Aku suka kuliah ini')`
  - `pesanB = Pesan('Surakarta: the Spirit of Java')`
- Kata `self` maksudnya adalah mengacu pada diri si instance itu. Jadi kalau di class `Pesan` kita ketikkan `self.teks`, maka saat class-nya terinstantiasi menjadi object `pesanA`, kita jadi punya variable `pesanA.teks`. Cobalah!
- Sebuah metode di suatu class umumnya mempunyai bentuk

```
class sembarangKelas(object):
    def metodeSatu(self):
        pass
```

[7]To be exact, ini hanya salah satu cara. Ada cara-cara yang lain.

```
    def metodeSembilan(self,stringBaru):
        pass
```

Kata `self` ini selalu ada di situ, mengacu pada dirinya. Saat memanggil metode ini, kata `self` ini **jangan dihitung**. Jadi untuk dua di atas pemanggilannya adalah

```
>>> obQ = sembarangKelas()                  # instantiasi
>>> obQ.metodeSatu()                        # pemanggilan metodeSatu()
>>> obQ.metodeSembilan('Aku suka mie ayam') #
```

Bandingkanlah dengan class `Pesan` yang telah kamu buat.

- Kita dapat mengubah nilai suatu data dengan memanggil metode tertentu yang dibuat untuk itu. Dalam class Pesan itu, kita mempunyai metode

  ```
  def perbarui(self,stringBaru):
      self.teks = stringBaru
  ```

  yang mengubah variabel `teks` milik instance yang relevan. Misal, seperti sudah dilakukan di atas, `pesanA.perbarui('Aku senang struktur data')`.

Latihan 2.3 Sebuah kelas sederhana lainnya. Ketik dan simpan yang berikut ini.

LatOOP3.py

```
1  class Manusia(object):
2      """ Class 'Manusia' dengan inisiasi 'nama' """
3      keadaan = 'lapar'
4      def __init__(self,nama):
5          self.nama = nama
6      def ucapkanSalam(self):
7          print("Salaam, namaku", self.nama)
8      def makan(self, s):
9          print("Saya baru saja makan", s)
10         self.keadaan = 'kenyang'
11     def olahraga(self,k):
12         print("Saya baru saja latihan", k)
13         self.keadaan = 'lapar'
14     def mengalikanDenganDua(self,n):
15         return n*2
16
17 ## Kali ini melarikannya lewat file yang sama.
18 ## Lewat python shell juga bisa.
19 p1 = Manusia('Fatimah')
20 p1.ucapkanSalam()
```

Larikan program di atas. Maka kamu akan mendapatkan

```
Salaam, namaku Fatimah
```

Kamu juga dapat membuat instance yang lain dan mengetes-nya. Seperti ini:

```
>>> p2 = Manusia('Budi')
>>> p2.ucapkanSalam()
Salaam, namaku Budi
```

□

Perhatikan bahwa class `Manusia` di atas mempunyai beberapa metode lain. Ada metode yang mengubah kedaan di dalam object (*internal state*), ada pula metode yang mengembalikan sesuatu. Jalankan contoh berikut di Python Shell -mu.

```
>>> ak = Manusia('Abdul Karim')
>>> ak.ucapkanSalam()
Salaam, namaku Abdul Karim
>>> ak.keadaan
'lapar'
>>> ak.makan('nasi goreng')
Saya baru saja makan nasi goreng
>>> ak.keadaan
'kenyang'
>>> ak.olahraga('renang')
Saya baru saja latihan renang
>>> ak.keadaan
'lapar'
>>> ak.makan('bakso')
Saya baru saja makan bakso
>>> ak.keadaan
'kenyang'
>>> ak.mengalikanDenganDua(8)
16
```

Bagaimana cara mengubah `keadaan` dari `'lapar'` ke `'kenyang'`? Metode apa yang membuat keadaan menjadi `'lapar'` lagi? Metode apa yang mengembalikan sesuatu?

2.2.1 Pewarisan

Pewarisan atau *inheritance* adalah pembuatan suatu class berdasarkan class lain. Ini adalah topik yang luas sekali, kami paparkan di sini sebagai perkenalan.

Contoh berikut memberi gambaran konsep pewarisan ini. Di sini kita akan membuat class `Mahasiswa` yang dibangun dari class `Manusia`. Ingat, seorang mahasiswa adalah juga seorang manusia.

Class Mahasiswa ini dibangun dari class manusia. Berarti dia mewarisi semua hal yang dipunyai Manusia. Class Mahasiswa bisa mempunyai metode lain yang tidak dipunyai manusia lain pada umumnya. Plus, class mahasiswa bisa saja mempunyai metode yang menutupi metode tertentu – yang sama amanya – sebagai manusia.

Latihan 2.4 Mari kita sebuah class yang bisa menampung data-data mahasiswa: nama, NIM, kotaTinggal, uangSaku. Ketika dipanggil pertama kali untuk membuat instance seorang mahasiswa, data-data ini akan sudah disediakan.

Class ini bisa dibuat di file yang sama dengan file yang memuat class Manusia. Atau di bagian awalnya meng-import file latihan sebelumnya. Class Mahasiswa ini mewarisi semua metode dari class manusia. Juga mewarisi semua state manusia. Akan tetapi di sini ada dua metode yang menutupi metode (dengan nama yang sama) di kelas manusia. Jika metodenya dipanggil, yang dijalankan adalah metode yang paling dekat ke dirinya sendiri.

LatOOP4.py

```
1 class Mahasiswa(Manusia):
```

```
    """Class Mahasiswa yang dibangun dari class Manusia."""
    def __init__(self,nama,NIM,kota,us):
        """Metode inisiasi ini menutupi metode inisiasi di class Manusia."""
        self.nama = nama
        self.NIM = NIM
        self.kotaTinggal = kota
        self.uangSaku = us
    def __str__(self):
        s = self.nama + ', NIM ' + str(self.NIM) \
            + '. Tinggal di ' + self.kotaTinggal \
            + '. Uang saku Rp ' + str(self.uangSaku) \
            + ' tiap bulannya.'
        return s
    def ambilNama(self):
        return self.nama
    def ambilNIM(self):
        return self.NIM
    def ambilUangSaku(self):
        return self.uangSaku
    def makan(self,s):
        """Metode ini menutupi metode 'makan'-nya class Manusia.
        Mahasiswa kalau makan sambil belajar."""
        print("Saya baru saja makan",s,"sambil belajar.")
        self.keadaan = 'kenyang'

# ada kelanjutannya (lihat di "Soal-soal untuk Mahasiswa").
```

Untuk mengetesnya, larikan (atau `import`) script di atas lalu eksekusi yang berikut

```
m1 = Mahasiswa('Jamil',234,'Surakarta',250000)
m2 = Mahasiswa('Andi',365,'Magelang',275000)
m3 = Mahasiswa('Sri', 676,'Yogyakarta',240000)
```

Sekarang di memori sudah termuat tiga 'object mahasiswa' dengan tiap-tiap object itu mempunyai data dan metode-metode. Contoh dan kembangkan yang berikut ini:

```
>>> m1.ambilNama()
'Jamil'
>>> m2.ambilNIM()
365
>>> m3.ucapkanSalam()
Salaam, namaku Sri
>>> m3.keadaan
'lapar'
>>> m3.makan('gado-gado')
Saya baru saja makan gado-gado sambil belajar.
>>> m3.keadaan
'kenyang'
>>> print(m3)
Sri, NIM 676. Tinggal di Yogyakarta. Uang saku Rp 240000 tiap bulannya.
```

Perhatikan dua baris terakhir di atas. Bagaimana mungkin perintah `print(m3)` menghasilkan string seperti itu? Jawabannya adalah karena kita telah meng-override metode bawaan `__str__()` di class itu. Metode inilah yang dieksekusi oleh python ketika object yang bersangkutan diminta mengeluarkan suatu string[8], misal karena perintah `print()` atau 'dipaksa jadi string'

[8]Contoh di atas sebenarnya tidak begitu umum dalam penggunaan `__str__()`. Umumnya, metode ini akan

dengan *cast* `str(m3)`. Metode `__str__()` ini kegunaannya kurang lebih sama dengan metode *`toString()`* di Java. □

Dari contoh-contoh di atas, kamu sekarang sudah mengenal beberapa metode yang spesial, seperti `__init__()`, `__doc__`, `__str__()`. Masih banyak lagi metode-metode seperti ini. Silakan baca dokumentasi dan buku referensi Python yang bagus.

Contoh pewarisan lagi. Misal kita akan membuat class MhsTIF, yakni class khusus mahasiswa teknik informatika. Tapi mahasiswa teknik informatika kan juga mahasiswa? (Dan berarti juga manusia.) Kalau begitu kita pakai saja class `Mahasiswa` di atas sebagai basis.

Latihan 2.5 Membuat class `MhsTIF` yang didasarkan pada class `Mahasiswa`.

```
import LatOOP4              # Atau apapun file-nya yang kamu buat tadi
class MhsTIF(Mahasiswa):    # perhatikan class induknya: Mahasiswa
    """Class MhsTIF yang dibangun dari class Mahasiswa"""
    def katakanPy(self):
        print('Python is cool.')
```

Sekarang **semua object yang di-instantiasi dari class `MhsTIF` akan mempunyai metode dan atribut yang sama dengan metode dan atribut class induknya, `Mahasiswa`**. Tapi beda dengan mahasiswa lainnya, mahasiswa teknik informatika mempunyai metode yang hanya dimiliki mereka, yakni `katakanPy()`. Jadi kita bisa langsung mengeksekusi yang berikut ini

```
>>> m4 = MhsTIF('Badu',334,'Sragen',230000)
>>> m4.katakanPy()
Python is cool.
>>> print(m4)
Badu, NIM 334. Tinggal di Sragen. Uang saku Rp 230000 tiap bulannya.
>>> m4.keadaan
'lapar'
>>> m4.makan('pecel')
Saya baru saja makan pecel sambil belajar.
>>> m4.keadaan
'kenyang'
>>> m4.ucapkanSalam()
Salaam, namaku Badu
```

□

2.3 Object dan List

Seperti halnya object-object yang berasal dari class lain seperti `int`, `str`, `float`, object yang kamu buat di atas bisa juga dikumpulkan di dalam suatu list.

mengeluarkan sesuatu yang simple, yang kalau dengan contoh di atas dia akan mengeluarkan, misal, nama si mahasiswa. Selengkapnya:

```
def __str__(self):
    return self.nama
```

Latihan 2.6 Daftar mahasiswa. Masih melanjutkan Contoh 2.4 di atas, kita sekarang akan mencoba mengumpulkan semua object mahasiswa di atas dalam suatu list. Kamu bisa membuat tambahan object mahasiswa lain terlebih dahulu. Contohlah yang berikut ini:

```
>>> daftar = [m1, m2, m3]  # tambahkan lainnya jika kamu punya
>>> for i in daftar: print(i.NIM) # tekan <Enter> dua kali

234
365
676
>>> for i in daftar: print(i)

Jamil, NIM 234. Tinggal di Surakarta. Uang saku Rp 250000 tiap bulannya.
Andi, NIM 365. Tinggal di Magelang. Uang saku Rp 275000 tiap bulannya.
Sri, NIM 676. Tinggal di Yogyakarta. Uang saku Rp 240000 tiap bulannya.
>>>
>>> daftar[2].ambilNama()
'Sri'
```

□

Sekarang, kita bisa mencoba hal-hal seperti berikut

- Dari suatu daftar mahasiswa, carilah mahasiswa yang namanya 'Sri'.
- Urutkan daftar mahasiswa itu berdasarkan NIM.

Itu adalah beberapa hal di antara yang insya Allah akan dipelajari di pertemuan-pertemuan berikutnya.

2.4 Class sebagai *namespace*

Sesudah sebuah class di-instantiasi menjadi suatu object, object tersebut dapat 'digantoli' dengan berbagai macam variabel.

Latihan 2.7 Ketik dan larikan file ini

LatOOP7.py

```
class kelasKosongan(object):
    pass

## Sekarang kita coba
k = kelasKosongan()
k.x = 23
k.y = 47
print(k.x + k.y)
k.mystr = 'Indonesia'
print(k.mystr)
```

Bagaimanakah hasilnya? □

Seperti sudah kamu lihat di atas, variabel-variabel bisa dicantholkan dengan bebas pada instance suatu class. Feature ini merupakan salah satu kekuatan Python, meski bagi yang datang dari Java atau C++ ini juga merupakan suatu kejutan.

2.5 Topik berikutnya di OOP

Topik-topik yang dibahas di atas merupakan pengenalan awal OOP. Masih banyak sekali topik-topik yang harus kamu pelajari untuk menjadi mahir dalam menggunakan konsep ini. Tujuan modul ini adalah untuk memotivasti dan memberi paparan awal pada mahasiswa akan paradigma pemorgraman berorientasi objek dan mempersiapkan mahasiswa untuk menghadapi bahan-bahan praktikum selanjutnya.

Untuk belajar lebih lanjut terkait topik ini, silakan buka

- docs.python.org/3/tutorial/classes.html
- www.tutorialspoint.com/python3/python_classes_objects.htm

2.6 Soal-soal untuk Mahasiswa

1. Pada Contoh 2.2, kita telah membuat class Pesan yang berisi beberapa metode. Tambahkan metode-metode di bawah ini ke dalam class itu.

   (a) Metode untuk memeriksa apakah suatu string terkandung di object Pesan itu. Seperti ini hasilnya:

   ```
   >>> p9 = Pesan('Indonesia adalah negeri yang indah')
   >>> p9.apakahTerkandung('ege')
   True
   >>> p9.apakahTerkandung('eka')
   False
   ```

   (b) Metode untuk menghitung jumlah konsonan.

   ```
   >>> p10 = Pesan('Surakarta')
   >>> p10.hitungKonsonan()
   5
   ```

   (c) Metode untuk menghitung jumlah huruf vokal.

   ```
   >>> p10.hitungVokal()
   4
   ```

2. Lihat kembali contoh 2.4. Tambahkan beberapa metode seperti dijelaskan di bawah ini

   (a) Metode untuk mengambil kota tempat tinggal si mahasiswa. Seperti ini hasilnya:

   ```
   >>> m9.ambilKotaTinggal()
   'Surabaya'
   ```

   (b) Metode untuk memperbarui kota tinggal. Seperti ini hasilnya:

   ```
   >>> m9.perbaruiKotaTinggal('Sleman')
   >>> m9.ambilKotaTinggal()
   'Sleman'
   ```

   (c) Metode untuk **menambah** uang saku. Seperti ini hasilnya:

   ```
   >>> m7.ambilUangSaku()
   270000
   ```

```
>>> m7.tambahUangSaku(50000)
>>> m7.ambilUangSaku()
320000
```

3. Masih di contoh 2.4. Buatlah suatu program untuk memasukkan data mahasiswa baru lewat Python Shell secara interaktif. Seperti sudah kamu duga, gunakanlah `input()`.

4. Buatlah state baru di class `Mahasiswa` bernama `listKuliah` yang berupa list berisi daftar matakuliah yang diambil. Buat pula metode `ambilKuliah()` yang akan menambah daftar matakuliah ini. Contoh pemanggilan:

```
>>> m234.listKuliah
[]
>>> m234.ambilKuliah('Matematika Diskrit')
>>> m234.listKuliah
['Matematika Diskrit']
>>> m234.ambilKuliah('Algoritma dan Struktur Data')
>>> m234.listKuliah
['Matematika Diskrit', 'Algoritma dan Struktur Data']
>>>
```

5. Berkaitan dengan nomer sebelumnya, buatlah metode untuk menghapus sebuah matakuliah dari `listKuliah`.

6. Dari class `Manusia`, buatlah sebuah class `SiswaSMA` yang memuat metode-metode baru (kamu bebas menentukan).

7. Dengan membuat suatu instance dari class MhsTIF (halaman 25), beri keterangan pada setiap metode dan *state* yang tampak di object itu (lihat gambar di bawah): apakah metode/*state* itu berasal dari class `Manusia`, `Mahasiswa`, atau `MhsTIF`?