Algoritma Pemrograman

ABSTRACT DATA TYPE (ADT)

NISA'UL HAFIDHOH, M.T.

TIM ALPRO – TI S1

Review

- Sebelumnya sudah memahami dan menggunakan berbagai macam tipe data (integer, float, character, dll)

- Sebelumnya sudah belajar untuk menggunakan dan memanipulasi string, list, tuple dan dictionary pada pemrograman python.

Tujuan Pembelajaran

Mahasiswa mampu mengidentifikasi dan menerapkan komponen data dan perilaku dari satu Abstract data Type (ADT).

Mahasiswa mampu membuat lebih dari satu Abstract data Type (ADT) dan dapat digunakan bersama-sama untuk pemecahan suatu masalah.

Tipe Data

Pola representasi suatu data dalam komputer -> menentukan secara internal bagaimana data disimpan dalam computer

Jenis tipe data :

a. Tipe Data Dasar / Primitif

Tipe data yang sudah tersedia / didefinisikan dalam suatu bahasa

b. Tipe Data Bentukan / Abstract Data Type

Tipe data yang dapat didefinisikan sendiri dan disusun dari berbagai tipe data dasar

Abstract Data Type (ADT)

ADT / Tipe data abstrak merupakan jenis spesial dari tipe data

Didefinisikan oleh set dari nilai-nilai dan operasi-operasi

ADT dibentuk oleh tipe data primitif yang sudah ada, tetapi operasi logika yang ada didalamnya disembunyikan.

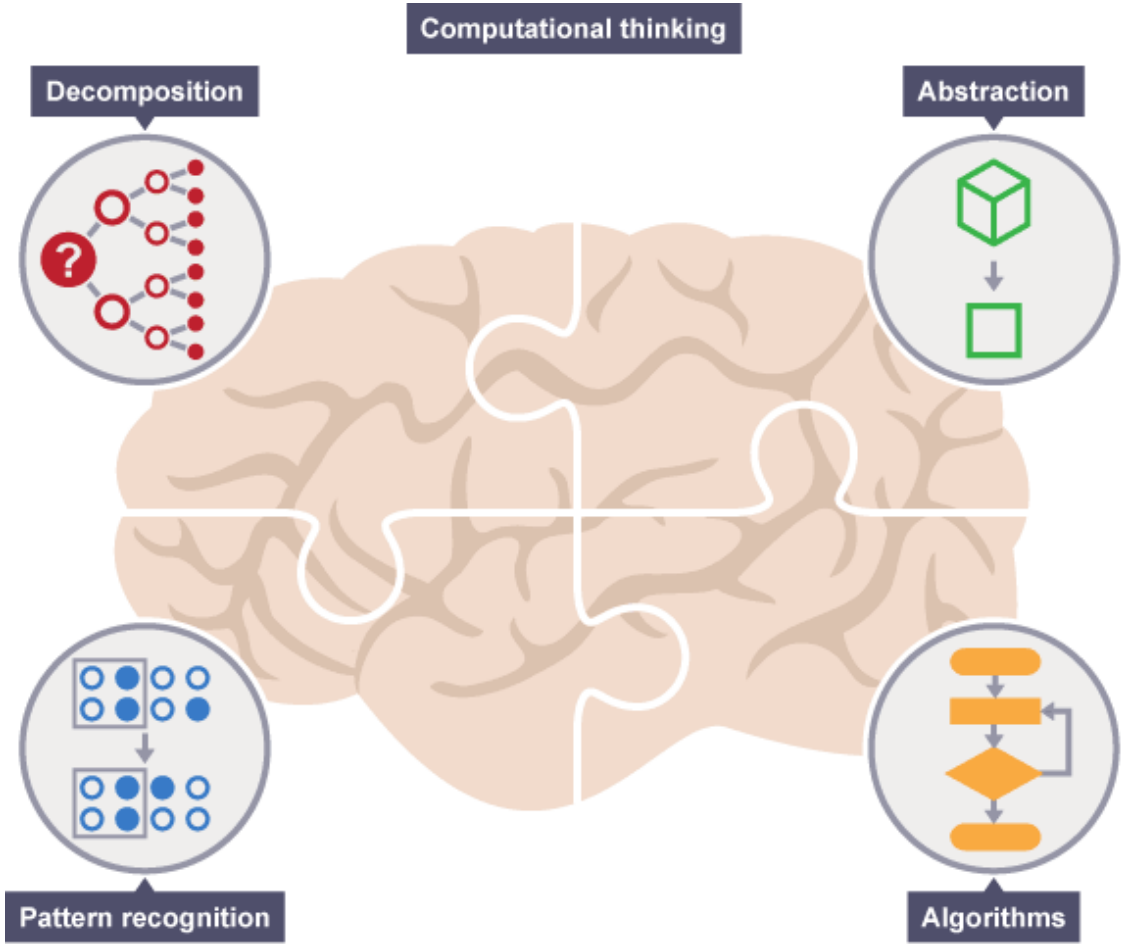
Kata “Abstract” dapat diartikan sebagai “kita dapat menggunakan tipe data ini dan bisa menggunakan operasi yang berbeda-beda tanpa harus mengetahui bagaimana operasi tersebut bekerja”

ADT dicapai dengan Pemrograman Berorientasi Object (di Python)

Berpikir Komputasional

Masih ingat di awal semester?

- Dekomposisi
- Abstraksi
- Pengenalan Pola
- Algoritma



Abstraksi

Merupakan ide yang sangat berguna dalam ilmu komputer.

Hal ini dapat memisahkan dari APA sampai BAGAIMANA

Abstraksi dan dekomposisi akan menyediakan modularitas.

Dimana modularitas bisa dicapai dengan FUNGSI atau CLASS.

Pada python Suatu Abstract Data Type/Tipe Data Abstrak hanya dapat diciptakan dengan suatu Class.

Sehingga paradigma yang akan kita pelajari adalah Object Oriented Programming/ Pemrograman Berorientasi Objek

Pemrograman Berorientasi Objek

Semua yang ada di python adalah objek (memiliki tipe data)!

Dapat menciptakan objek baru dengan beberapa tipe data

Dapat memanipulasi objek

Dapat menghancurkan objek (dimana ketika objek tadi sudah tidak dibutuhkan lagi)

- Dilakukan dengan perintah “del” di python
- Sistem python akan mengakui kembali objek yang dihancurkan atau objek yang tidak dapat di akses yang dikenal dengan istilah “garbage collection”

Objek

Python mendukung apapun bentuk data

Setiap data tersebut dikenal sebagai objek

Jadi objek memiliki:

- Tipe
- Representasi internal dari data (bisa primitif atau gabungan/composit)
- Berisi set prosedur dari interaksi dengan objek lain

Dengan kata lain Objek merupakan **instance** dari tipe (terminologi yang lain mengatakan instance of class)

- 23245 merupakan instance dari int
- “Hai” merupakan instance dari string

Objek

Secara sederhana, objek adalah **abstraksi dari data** yang menangkap:

1. Suatu representasi internal (melalui atribut data) - APA
2. Suatu penghubung untuk berinteraksi dengan objek lain - BAGAIMANA
   - Melalui suatu yang disebut dengan method (sudah kita kenal sebagai fungsi/prosedur)
   - Dimana method tadi mendefinisikan suatu perilaku/behavior dengan implementasi yang tersebumnyi.

Contoh: Objek Mahasiswa memiliki atribut data NIM dan memiliki tingkah laku “Mengambil mata kuliah”, dimana tingkah laku dari objek Mahasiswa terhubung dengan objek lain yaitu Mata Kuliah.

Contoh: Objek List → [1,2,3,4]

Bagaimana sebetulnya list direpresentasikan secara internal?

- Terdiri dari sel list yang saling tertaut satu sama lain
- Setiap sel terdiri dari nilai yang memiliki tipe tertentu, dan pointer untuk mentautkan(link) ke sel yang lain

L = 

Bagaimana cara memanipulasi List? L[i], L[i:j], + , len(), max(), sum(), L.append(),L.remove(),L.sort()

Representasi internal tersebut harus tersembunyi (private)

Bisakah kita membuat “LIST” kita sendiri?

Kita dapat membuat ADT kita sendiri dengan cara membuat **class.**

Perhatikan antara perbedaan dari class dan menggunakan instance dari class (objek)

- Ilustrasi sederhana: Blueprint design rumah merupakan class dari objek rumah

Pembuatan class melibatkan:

- Pendefinisian nama class
- Pendefinisian attribut (atau property)
- Pendefinisian method
- Contoh konkrit: Programmer menuliskan kode program untuk merepresentasikan List dengan mengetikan kode class List

Menggunakan class melibatkan:

- Pembuatan instance baru dari objek
- Melakukan operasi dari instance tersebut
- Contoh: L = [1,2] dan L.append(3)

Mari kita membuat ADT kita sendiri!

Misal membuat class Koordinat. Bagaimana cara membuat kode programmnya?

Gunakan keyword “class” untuk mendefinisikan tipe yang baru

```
class Koordinat(object):
    #mulai definisikan atribut dan/atau method disini
```

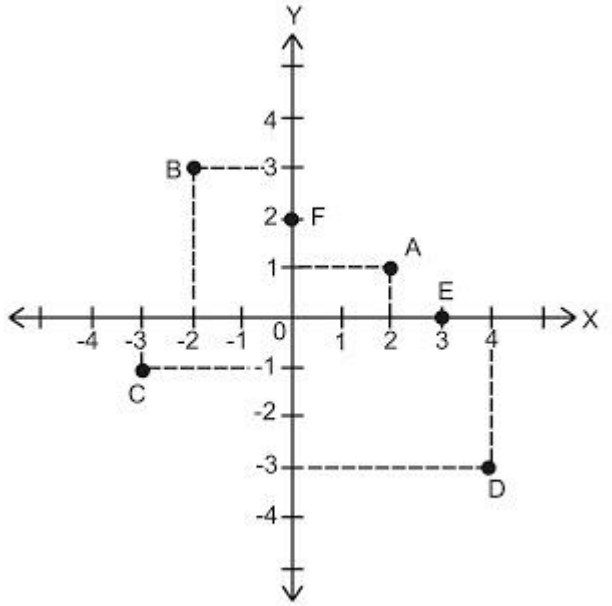
Nama class

Parent/superclass

Gunakan indentasi sama halnya ketika kita mendefiniskan fungsi dengan “def”

“object” berarti class Koordinat yang kita buat merupakan Objek Python dan mewarisi semua atrributnya (berkaitan dengan konsep inheritance/pewarisan)

- Kita bisa mengatakan bahwa Koordinat merupakan subclass dari object
- Dan object merupakan superclass dari Koordinat



Apa isi atribut dan method dari class Koordinat?

Lihat representasi gambar dari Koordinat sebelumnya.

Atribut data:

- Pikirkan data yang terkait langsung dengan class yang dibangun
- Contoh: koordinat dibentuk dari dua data integer yang diwakili oleh x dan y

Method:

- Pikirkan fungsi/prosedur apa yang terkait dengan class yang dibangun
- Pikirkan juga bagaimana caranya agar dapat terkait dengan objek lain
- Contoh: membuat method untuk menghitung jarak antar dua titik koordinat

Mendefinisikan bagaimana cara untuk membuat objek

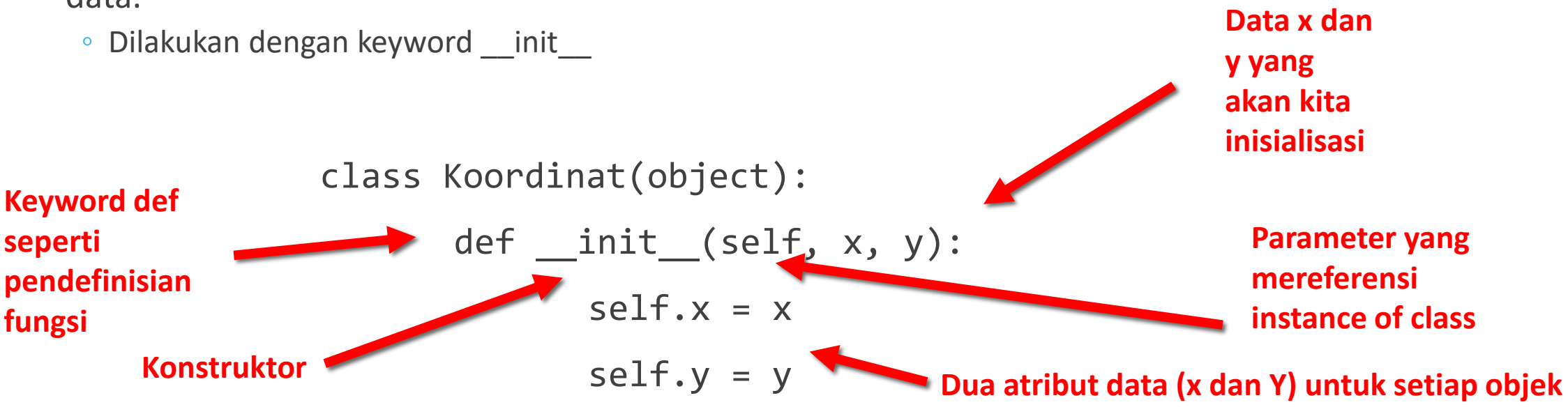
Membuat Objek bisa dilakukan dengan men-assign ke suatu variabel

- Contoh: c = Koordinat()

Perhatikan bahwa contoh sebelumnya tidak menginisialisasi nilai atribut data. Misal: x = 10, y = 4

Gunakan spesial method (disebut juga dengan istilah konstruktor) untuk menginisialisasi atribut data.

- Dilakukan dengan keyword __init__



Pembuatan Object

```
titik_c = Koordinat(4,5)
titik_asal = Koordinat(0,0)
print(titik_c.x)
print(titik_c.y)
print(titik_asal.x)
print(titik_asal.y)
```

Data atribut dari suatu instance atau objek disebut dengan instances variables

Jangan memanggil self karena python sudah melakukannya secara otomatis.

Full Code!

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

def main():
    titik_c = Koordinat(4,5)
    titik_asal = Koordinat(0,0)
    print("titik c: x=",titik_c.x,"y=",titik_c.y)
    print("titik asal: x=",titik_asal.x,"y=",titik_asal.y)
    titik_c.x = 12
    print("titik c: x=",titik_c.x,"y=",titik_c.y)

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

Full Code! (2 File)

Main.py

```
from Koordinat import *

def main():
    titik_c = Koordinat(4,5)
    titik_asal = Koordinat(0,0)
    print("titik c: x=",titik_c.x,"y=",titik_c.y)
    print("titik asal: x=",titik_asal.x,"y=",titik_asal.y)
    titik_c.x = 12
    print("titik c: x=",titik_c.x,"y=",titik_c.y)

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

Koordinat.py

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

Btw, Kita belum membuat method menghitung jarak..

Method

Merupakan atribut prosedural, atau anggap saja itu sebagai fungsi pada class

Python selalu meneruskan objek sebagai argumen pertama

- Biasanya menggunakan self sebagai nama dari argumen pertama method yang ada.
- Argumen yang dimaksudkan disini adalah parameter formal

Simbol “.” atau titik digunakan untuk mengakses apapun atribut dari class.

- Bisa jadi atribut data dari class
- Atau method dari class

Mendefinisikan Method!

```
class Koordinat(object):
   def __init__(self,x,y):
      self.x = x
      self.y = y
   def jarak(self,lain):
      x_jar = (self.x - lain.x)**2
      y_jar = (self.y - lain.y)**2
      return (x_jar+y_jar)**0.5
```

Method seperti halnya fungsi membawa parameter

Perhatikan untuk mengakses atribut digunakan dot atau titik

Memanggil Method

Cara 1 (yang biasa):

- `k = Koordinat(3,4)`
- `nol = Koordinat(0,0)`
- `print(k.jarak(nol))`

Cara 2:

- `k = Koordinat(3,4)`
- `nol = Koordinat(0,0)`
- `print(Koordinat.jarak(k,nol))`

Mencetak representasi dari objek

K = Koordinat(4,6)

print(K)

Output dari kode diatas adalah <__main__.Coordinateobject at 0x7fa918510488>

Bagaimana caranya ketika kita mencetak K akan berisi representasi dari objek tersebut, contoh output: <4,6>

Gunakan mencetak representasi un-informatif dengan cara definisikan __str__ method ketika digunakan dengan print pada objek class.

Mendefinisikan “print” kita sendiri!

```
class Koordinat(object):
   def __init__(self,x,y):
      self.x = x
      self.y = y
   def jarak(self,lain):
      x_jar = (self.x - lain.x)**2
      y_jar = (self.y - lain.y)**2
      return (x_jar+y_jar)**0.5
   def __str__(self):
      return “<”+str(self.x)+”,”+str(self.y)+”>”
```

Sekarang coba panggil:

```
K = Koordinat(1,2)
print(K)
print(type(K))
```

Perhatikan bahwa

Kita dapat bertanya untuk tipe dari objek instance

- K = Koordinat(1,2)
- print(K) #output <1,2>
- print(type(K)) #output <class __main___.Koordinat>

Ini akan menjadi masuk akal dimana,

- print(Koordinat) #output <class __main__.Koordinat>
- print(type(Koordinat)) #output <type ‘type’>

Gunakan isinstance() untuk mengecek suatu objek instance merupakan instance dari class tertentu

- print(isinstance(K, Koordinat)) #output True

Built-in class Attributes

Setiap class di python akan secara otomatis memiliki attribut-attribut yang memang sudah disiapkan tanpa harus mendefinisikan atribut tersebut didalam class.

Atribut-atribut built-in ini dapat diakses dengan titik.

- **__dict__** − Dictionary yang berisi namespace dari class.
- **__doc__** − Docstring class.
- **__name__** − nama class.
- **__module__** − Nama modul dimana class tersebut diciptakan. Biasanya mengembalikan main"__main__" pada mode interaktif.
- **__bases__** − Bisa jadi tuple kosong, mengembalikan parent/superclass/base class.

Panggil itu dengan cara:

- print(Koordinat.__dict__)
- print(Koordinat.__doc__)

Custom Operator

Misalnya kita membuat class Vector 2D lalu kita membuat instance dari Vector tersebut dengan dua objek, misal: vekA dan vekB, Ketika ingin menjumlahkan kedua vektor tersebut, programmer ingin menjumlahkan kedua sumbu dari vekA dan vekB tersebut dengan operator “+”, contoh: print(vekA + vekB)

Penjumlahan kedua vektor tadi tentu saja berbeda dengan penjumlahan integer atau float biasa.

Maka dari itu kita dapat membuat method custom didalam class untuk memodifikasi operator + yang digunakan untuk menjumlahkan kedua aksis yang ada di objek vekA dan vekB.

Custom to Special Operator

+,-,/,*,==,<,>,len(),print

Definisikan dengan:

- __add__(self, other)
- __sub__(self, other)
- __eq__(self, other)
- __lt__(self, other)
- __len__(self)
- __str__(self)
- dll

Contoh Lain

Mahasiswa memiliki atribut Nama dan NIM, kita dapat menampilkan informasi untuk mahasiswa tersebut.

```
class Mahasiswa(object):
    def __init__(self,nama,nim):
        self.nama = nama
        self.nim = nim
    def tampilkanInfo(self):
        print("Nama : ", self.nama)
        print("NIM : ", self.nim)
        print()

mhs=Mahasiswa("nisa", 123)
mhs.tampilkanInfo()
```

Latihan

Buatlah 3 objek dengan masing-masing minimal memiliki 3 atribut dan 2 method.

Referensi

Introduction to Computer Science and Programming in Python, MIT
https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-0001-introduction-to-computer-science-and-programming-in-python-fall-2016/lecture-slides-code/MIT6_0001F16_Lec8.pdf

Guttag, John. Introduction to Computation and Programming Using Python: With Application to Understanding Data Second Edition. MIT Press, 2016. ISBN: 9780262529624.