Python: Import, Function, Class

https://github.com/dudung/sk5003-02-2022-2

Sparisoma Viridi
Master Program in Computational Science, Nuclear Physics and Biophysics Research Division, Department of Physics, Faculty of Mathematics and Natural Sciences, Institut Teknologi Bandung, Bandung 40132, Indonesia

20230401-v6| https://doi.org/10.5281/zenodo.7809397

Silakan berdiskusi untuk kuliah hari ini di
https://github.com/dudung/sk5003-02-2022-2/issues/6

Kerangka

- SAP dan referensi 4
- Object orientation 8
- Object-oriented programs 23
- Diskusi dan latihan 32

SAP dan referensi

Minggu 5

Minggu	Topik	Subtopik	Capaian Belajar
5	Struktur data, orientasi objek, rekursi dalam Python	Orientasi objek dan program berorientasi objek	Kemampuan untuk memahami dan menguasai orientasi objek dan program berorientasi objek dengan Python

Referensi

- Jose M. Garrido, "Introduction to Computational Models with Python", Routledge, 1st edition, 2020,

  url https://isbnsearch.org/isbn/9780367575533.

R1

C8

- Object in problem domain
- Defining class
- Describing objects
- Interaction between objects
- Design with classes

C9

- Introduction and programs
- Classes definition in Python
- Create / manipulate objects
- Program with classes
- Scope of variables
- Class hierarchy with inheritance
- Overloading / overriding methods

Object orientation

Object orientation

- Object in problem domain
- Defining class
- Describing objects
- Interaction between objects
- Design with classes

Object in problem domain

- Real-world entities (RWEs) or objects are fundamental components of a real-worl system.
- Identifying and modeling RWEs in the problem domain are central focus of the object-oriented approach.
- A RWE has responsibility of carrying out a specific taks.
- A RWE entity is modeled as an object.

Abstraction

- A RWE has a lot of characteristics.
- A process called abstraction is used to modeled RWE in problem domain.
- The process also involves elimination of unessential characteristics, or parameters that are considered not important (include only relevant aspects of real-word system).
- Several levels of detail are required to define completely objects and collections of objects in a model.

Object-oriented modeling

It consists of

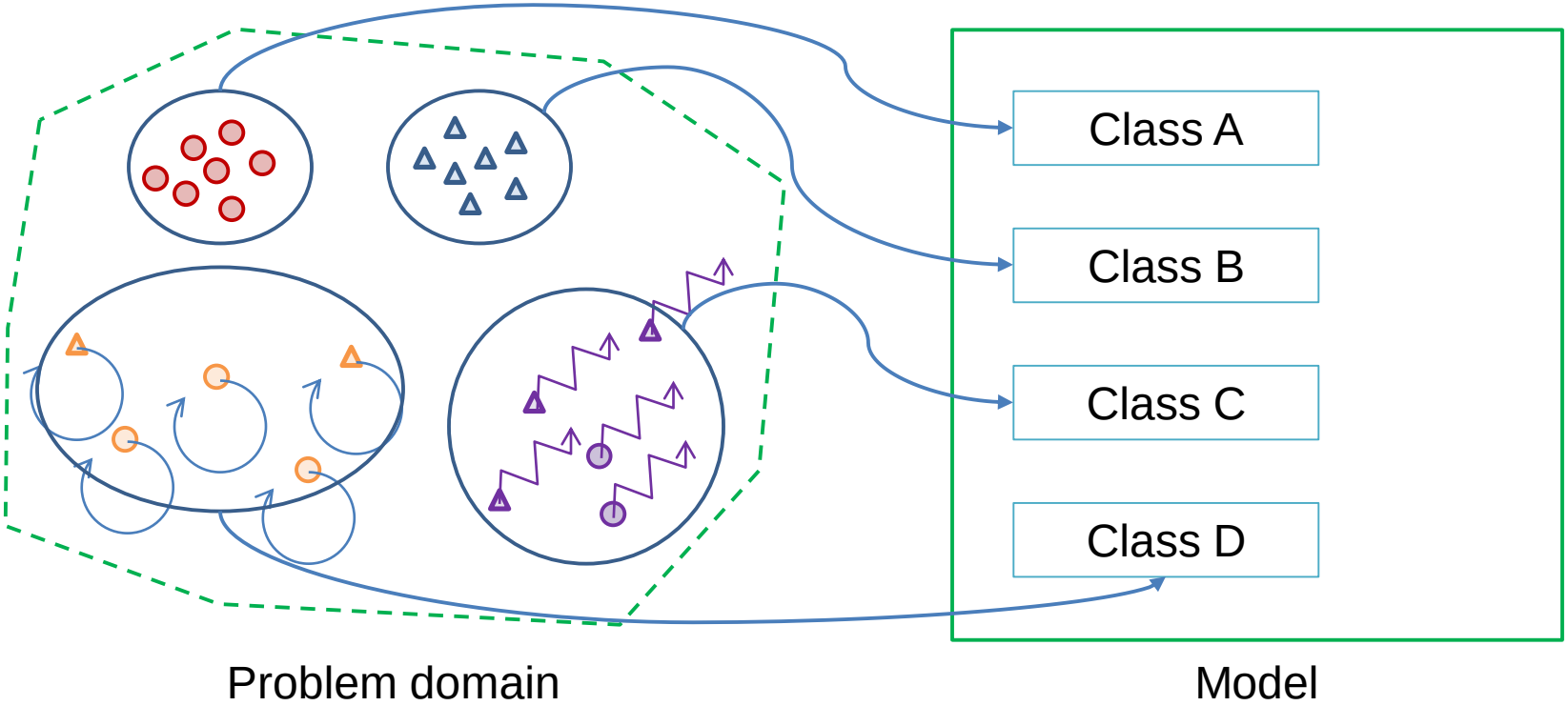
- identifying the relevant objects for the model,
- describing these objects using abstraction,
- defining collection of similar objects.

Objects with similar characteristics are grouped into collections, and these are modeled as classes, where UML (Unified Modeling Language) is a standard notation to describe object and classes in a problem domain.

Defining class

- All similar objects are group into a collection of objects.
- The collection of objects have same structure and behavior.
- A class is an abstract description of a collection of objects.
- A class defines attributes and behavior for all objects of the class.
- Software implementation of class consists of
  - Some data definitions represent attributes of the class,
  - Some methods (or operations) represent behaviors of the class.

Illustration in defining classes



Describing objects

- **A state**
  Represented by set of properties (or attributes) and their associated values.
- **Behavior**
  Represented by the operations, also known as methods, of the objects.
- **Identity**
  An implicit or explicit property that can uniquely identify an object.

```python
# Person class
class Person:

  # constructor
  def __init__(self, name="Unknown", age="0", weight="0"):
    self.name = name
    self.age = age
    self.weight = weight

  # string representation
  def __str__(self):
    vals = ""
    vals += f"Name   : {self.name}\n"
    vals += f"Age    : {self.age}\n"
    vals += f"Weight : {self.weight}\n"
    return vals
```

behavior

state

Interaction between objects

- Owner of an operation do something.
- The operation can change state of owner.
- The operation, when apply to other object, can also change state of the other object.

```python
from device import Computer as com
from device import Printer as prn
from device import Scanner as scn

p1 = prn("Printer_1")
print(p1.name)
print(p1.status)
print()

m1 = scn("Microscope_1")
print(m1.name)
print(m1.status)
print()

pc1 = com("Komputer_A")
pc1.add(p1)
print(pc1.name)
print()

print(p1.name)
print(p1.status)
print(p1.host)
print()

pc2 = com("Komputer_B")
pc2.add(m1)
print(pc2.name)
print()

print(m1.name)
print(m1.status)
print(m1.host)
print()
```

```
Printer_1
disconnected

Microscope_1
disconnected

Komputer_A

Printer_1
connected
Komputer_A

Komputer_B

Microscope_1
connected
Komputer_B
```

Design with classes

- **Encapsulation**
  This principle describes integration of object attributes and behavior as a single unit, which can be considered as a protection mechanism. Only certain objects can have access to other objects attributes and behaviors → private ones.
- **Data (or information) hiding**
  It hides implementation details, so there two views:
  - External view: List of services (or operations) an object can invoke.
  - Internal view: Details of implementation of data and operations.

```python
from device import Computer as com
from device import Monitor as mon
from device import Keyboard as kbd
from device import Mouse as mos
from device import Printer as prn
from device import Scanner as scn
from device import Network as net
from device import Speaker as spk
from device import Microphone as mic
from device import Camera as cam
from device import Digitizer as pen
from device import Storage as hdd
from device import Microscope as sco

pc1 = com("Komputer_1")
pc1.add(mon())
pc1.add(kbd())
pc1.add(mos())
pc1.add(spk())
pc1.add(mic())
pc1.add(net())
pc1.add(pen())
print(pc1.name)
print(pc1.list())

pc2 = com("Komputer_2")
pc2.add(mon())
pc2.add(kbd())
pc2.add(mos())
pc2.add(net())
pc2.add(pen())
pc2.add(hdd())
pc2.add(cam())
pc2.add(sco())
print(pc2.name)
print(pc2.list())
```

```
Komputer_1
> name Monitor, type output, status connected, host Komputer_1,
> name Keyboard, type input, status connected, host Komputer_1,
> name Mouse, type input, status connected, host Komputer_1,
> name Speaker, type output, status connected, host Komputer_1,
> name Microphone, type input, status connected, host Komputer_1,
> name Network, type bidirectional, status connected, host Komputer_1,
> name Digitizer, type input, status connected, host Komputer_1,

Komputer_2
> name Monitor, type output, status connected, host Komputer_2,
> name Keyboard, type input, status connected, host Komputer_2,
> name Mouse, type input, status connected, host Komputer_2,
> name Network, type bidirectional, status connected, host Komputer_2,
> name Digitizer, type input, status connected, host Komputer_2,
> name Storage, type bidirectional, status connected, host Komputer_2,
> name Camera, type input, zoom digital, status connected, host Komputer_2,
> name Microscope, type input, zoom optical, status connected, host Komputer_2,
```

```python
class Computer:
  def __init__(self, name="Computer"):
    self.name = name
    self.type = "bidrectional"
    self.devices = []

  def add(self, dev):
    self.devices.append(dev)
    dev.status = "connected"
    dev.host = self.name

  def list(self):
    lines = ""
    for i in self.devices:
      lines += "> "
      for attr, value in i.__dict__.items():
        lines += attr + " " + value + ", "
      lines += "\n"
    return lines
```

```python
class Camera:
  def __init__(self, name="Camera"):
    self.name = name
    self.type = "input"
    self.zoom = "digital"
    self.status = "disconnected"
    self.host = "none"


class Digitizer:
  def __init__(self, name="Digitizer"):
    self.name = name
    self.type = "input"
    self.status = "disconnected"
    self.host = "none"
```

```python
class Microscope(Camera):
  def __init__(self, name="Microscope"):
    Camera.__init__(self, name)
    self.zoom = "optical"
```

Object-oriented programs

Object-oriented programs

- Programs
- Classes definition in Python
- Create / manipulate objects
- Program with classes
- Scope of variables
- Class hierarchy with inheritance
- Overloading / overriding methods

Desired output

```
$ python group_of_person_<Approach>.py
Name   : Amir
Age    : 14
Weight : 40

Name   : Budi
Age    : 16
Weight : 42

Name   : Wati
Age    : 15
Weight : 38
```

Without class

```
4   names = ["Amir", "Budi", "Wati"]
5   ages = [14, 16, 15]
6   weights = [40, 42, 38]
7
8   n = len(names)
9
10  for i in range(0, n):
11    print("Name   :", names[i])
12    print("Age    :", ages[i])
13    print("Weight :", weights[i])
14    print()
```

With class

```python
# Person class
class Person:

  # constructor
  def __init__(self, name="Unknown", age="0", weight="0"):
    self.name = name
    self.age = age
    self.weight = weight

  # string representation
  def __str__(self):
    vals = ""
    vals += f"Name   : {self.name}\n"
    vals += f"Age    : {self.age}\n"
    vals += f"Weight : {self.weight}\n"
    return vals
```

```python
persons = []
persons.append(Person("Amir", 14, 40))
persons.append(Person("Budi", 16, 42))
persons.append(Person("Wati", 15, 38))

for i in persons:
  print(i)
```

Class using keywords

- It is clearer but more to type.
- Order of arguments depend on the given keywords.

```
persons = []
persons.append(Person(name="Amir", age=14, weight=40))
persons.append(Person(weight=42, name="Budi", age=16))
persons.append(Person(age=15, weight=38, name="Wati"))
```

Defining class in module

```python
# Person class
class Person:

  # constructor
  def __init__(self, name="Unknown", age="0", weight="0"):
    self.name = name
    self.age = age
    self.weight = weight

  # string representation
  def __str__(self):
    vals = ""
    vals += f"Name   : {self.name}\n"
    vals += f"Age    : {self.age}\n"
    vals += f"Weight : {self.weight}\n"
    return vals
```

Using class with import

- It hides all complex implementation in imported module.
- Main program is more simple.

```
4   from person import Person
5
6   persons = []
7   persons.append(Person(name="Amir", age=14, weight=40))
8   persons.append(Person(weight=42, name="Budi", age=16))
9   persons.append(Person(age=15, weight=38, name="Wati"))
10
11  for i in persons:
12    print(i)
```

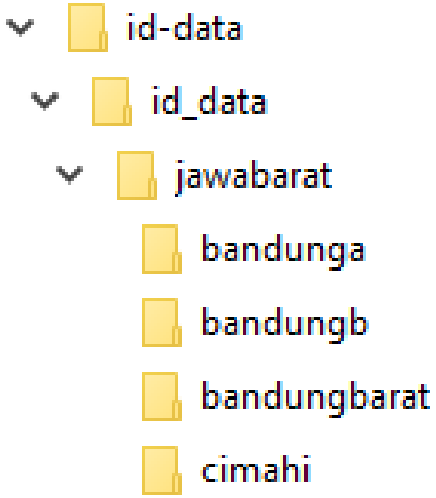
Comparison

Approach	Size (bytes)	Lines	Time (s)
multiple_lists	227	12	0.010093
list_of_class	552	25	0.006331
list_of_class_kwargs	600	25	0.005884
import	224	10	0.007803
(person module)	383	17	-

Diskusi dan latihan

Latihan dengan package id-data

- Package id_data



- bandungbarat
  - lembang

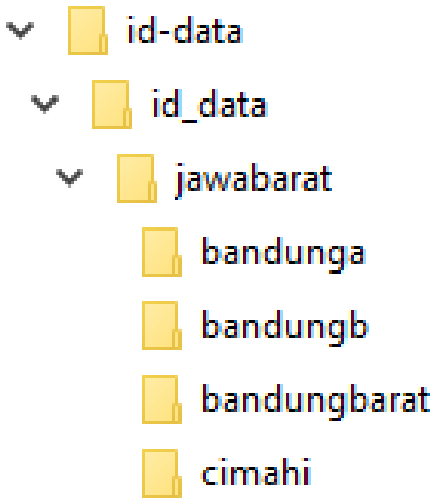
```
level = 3
name = 'Lembang'
capital = 'Lembang'
area = 95.56
```

use module lembang

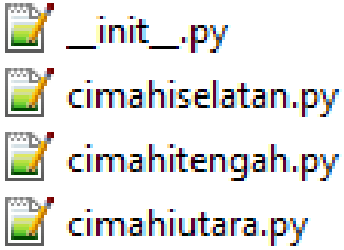
```python
import id_data.jawabarat.bandungbarat.lembang as lem

print(lem.name, lem.level, lem.area)


"""
$ python -m tests.ut_lembang
Lembang 3 95.56
"""
```

id-data

- Package id_data



- cimahi



cimahiselatan

```python
level = 3
name = 'Cimahi Selatan'
capital = 'Utama'
area = 16.94

population = {
  '2021': 240990000,
}

pdrb_adhk = {
  'A': {
    '2016': 32.3662E+9,
    '2017': 32.6328E+9,
    '2018': 32.8082E+9,
    '2019': 33.5885E+9,
    '2020': 34.0318E+9,
  },
  'B': {
    '2016': 0,
    '2017': 0,
    '2018': 0,
    '2019': 0,
    '2020': 0,
  },
  'C': {
    '2016': 8.6269025E+9,
    '2017': 8.9835191E+9,
    '2018': 9.6754401E+9,
    '2019': 10.7941862E+9,
    '2020': 10.2584726E+9,
  },
}
```

use module cimahiselatan, cimahitengah

```python
import id_data.jawabarat.cimahi.cimahiselatan as cims
import id_data.jawabarat.cimahi.cimahitengah as cimt
import id_data.jawabarat.cimahi.cimahiutara as cimu

print("Cimahi Selatan population in 2021")
print(cims.population['2021'])
print()

print("Cimahi Tengah population in 2021")
print(cimt.population['2021'])
print()
```

```
$ python -m tests.ut_cimahi
Cimahi Selatan population in 2021
240990000

Cimahi Tengah population in 2021
161758000
```

.. module cimahiselatan, cimahitengah

```
print("Cimahi Selatan PDRB ADHK sector C in 2016")
print(cims.pdrb_adhk['C']['2016'])
print()

print("Cimahi Selatan PDRB ADHK sector C in 2017")
print(cims.pdrb_adhk['C']['2017'])
print()
```

```
Cimahi Selatan PDRB ADHK sector C in 2016
8626902500.0

Cimahi Selatan PDRB ADHK sector C in 2017
8983519100.0
```

Diskusi

- Silakan bila ada pertanyaan.
- Setelah kuliah pertanyaan dapat diajukan secara asinkron di url https://github.com/dudung/sk5003-02-2022-2/issues/6

Terima kasih

-