Pemrograman Berorientasi Objek

Ali Akbar Septiandri

Universitas Al-Azhar Indonesia

aliakbars@live.com

March 6, 2017

Overview

1. Pemrograman Berorientasi Objek
   - Modul
   - Kelas dan Objek
   - Inheritance
2. Proyek Python
   - Struktur Sederhana Proyek
   - Penanganan Error
   - Pengujian Terotomasi

Bahan Bacaan

1. McLaughlin, B., Pollice, G., & West, D. (2006). Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. O'Reilly Media, Inc.
2. Shaw, Z. A. (2013). Learn Python the hard way: A very simple introduction to the terrifyingly beautiful world of computers and code. Addison-Wesley. (Exercise 40-47)
3. `https://docs.python.org/2.7/tutorial/modules.html`
4. `https://docs.python.org/2.7/tutorial/classes.html`
5. `https://docs.python.org/2.7/tutorial/errors.html`
6. `http://docs.python-guide.org/en/latest/writing/structure/`

Pemrograman Berorientasi Objek

Konsep OOP

Python memiliki fitur standar dari pemrograman berorientasi objek (OOP), yaitu:

1. *"the class inheritance mechanism allows multiple base classes"*
2. *"a derived class can override any methods of its base class or classes"*
3. *"a method can call the method of a base class with the same name"*

Modul sebagai Kelas

1. Kita dapat melihat modul sebagai kelas
2. Modul diwakili oleh *files*
3. Fungsi atau variabel dalam suatu modul diakses dengan operator dot

Contoh

Example (contoh.py)

```
def foo():
    print 'Foo'

bar = 'Hello, World'
```

Example

```
import contoh

contoh.foo()
print contoh.bar
```

Variasi dari `import`

Terkadang, tidak semua bagian dari modul perlu kita import. Cara untuk mengimpor hanya sebagian dari modul dapat dilakukan seperti contoh-contoh berikut.

Example

```
>>> from fibo import fib, fib2
>>> fib(500)
1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Example

```
>>> from fibo import *
>>> fib(500)
1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Apa yang terjadi jika ada eksekusi fungsi dalam *file* dan modul tersebut diimpor?

Mengeksekusi Modul

Untuk menghindari eksekusi modul saat melakukan `import`, ubah kode dengan memeriksa `__name__` terlebih dahulu.

Example (fibo.py)

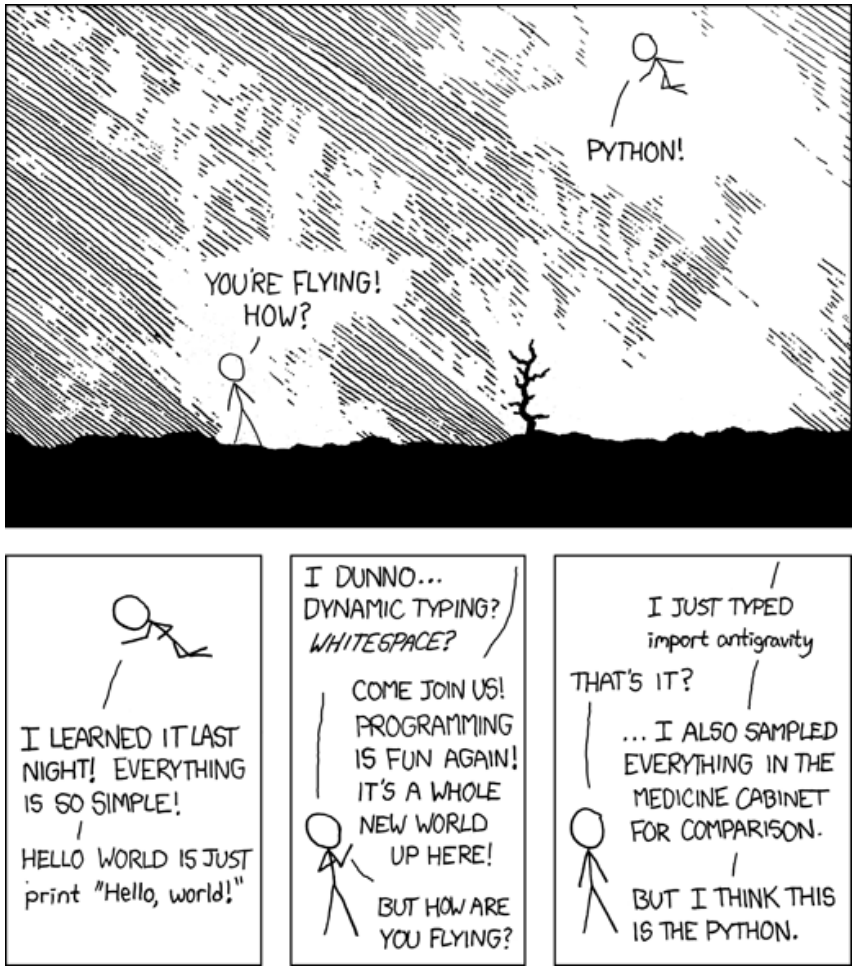
```
def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while b < n:
        print b,
        a, b = b, a+b

if __name__ == "__main__":
    import sys
    fib(int(sys.argv[1]))
```

Try this!

```
import antigravity
```

import antigravity



Gambar : Python | Sumber: `https://xkcd.com/353/`

Syntax Definisi Kelas

Definisi

```
class ClassName:
    <statement-1>
    .
    .
    .
    <statement-N>
```

Example

```
class MyClass:
    """A simple example class"""
    i = 12345

    def f(self):
        return 'hello world'
```

Instansiasi Kelas

1. Instansiasi kelas dilakukan dengan notasi pemanggilan fungsi

Instansiasi Kelas

1. Instansiasi kelas dilakukan dengan notasi pemanggilan fungsi
2. Dapat dilihat sebagai fungsi tanpa parameter yang dapat di-*assign* ke variabel

Instansiasi Kelas

1. Instansiasi kelas dilakukan dengan notasi pemanggilan fungsi
2. Dapat dilihat sebagai fungsi tanpa parameter yang dapat di-*assign* ke variabel
3. Hasil instansiasi tersebut dikenal dengan nama **objek**

Konstruktor

1. Setiap kelas secara bawaan memiliki metode spesial yang bernama `__init__()`

Konstruktor

1. Setiap kelas secara bawaan memiliki metode spesial yang bernama `__init__()`
2. Metode tersebut merupakan **konstruktor** dari kelas

Konstruktor

1. Setiap kelas secara bawaan memiliki metode spesial yang bernama `__init__()`
2. Metode tersebut merupakan **konstruktor** dari kelas
3. Instansiasi kelas secara otomatis memanggil metode `__init__()`

Konstruktor

1. Setiap kelas secara bawaan memiliki metode spesial yang bernama `__init__()`
2. Metode tersebut merupakan **konstruktor** dari kelas
3. Instansiasi kelas secara otomatis memanggil metode `__init__()`
4. `__init__()` dapat dimodifikasi agar dapat menerima argumen

Contoh

Example

```
>>> class Complex:
...     def __init__(self, realpart, imagpart):
...         self.r = realpart
...         self.i = imagpart
...
>>> x = Complex(3.0, -4.5)
>>> x.r, x.i
(3.0, -4.5)
```

Objek

1. **Atribut** merupakan variabel yang dimiliki objek
2. Atribut **tidak perlu dideklarasi**, seperti halnya variabel lokal
3. **Metode** adalah fungsi yang dimiliki objek

Enkapsulasi

- Secara umum, setiap atribut atau metode yang dimiliki sebuah kelas dapat diakses dari luar (publik)

Enkapsulasi

- Secara umum, setiap atribut atau metode yang dimiliki sebuah kelas dapat diakses dari luar (publik)
- Secara konvensi, untuk menghindari kesalahan, metode atau atribut privat **ditulis dengan minimal dua underscores di awalnya**

Enkapsulasi

- Secara umum, setiap atribut atau metode yang dimiliki sebuah kelas dapat diakses dari luar (publik)
- Secara konvensi, untuk menghindari kesalahan, metode atau atribut privat **ditulis dengan minimal dua underscores di awalnya**
- Komponen privat tersebut akan **berubah namanya**

Enkapsulasi

- Secara umum, setiap atribut atau metode yang dimiliki sebuah kelas dapat diakses dari luar (publik)
- Secara konvensi, untuk menghindari kesalahan, metode atau atribut privat **ditulis dengan minimal dua underscores di awalnya**
- Komponen privat tersebut akan **berubah namanya**
- Nama tersebut akan **tetap dipertahankan oleh anaknya**

Enkapsulasi

Example

```
class Foo:
    def __bar(self):
        pass
```

Example

```
foo = Foo()
foo._Foo__bar()
```

Berbagi Data Antarobjek

Atribut berupa objek yang *mutable* dapat menimbulkan masalah pada objek. Perhatikan contoh di bawah ini!

Example

```
class Dog:

    tricks = []             # penggunaan variabel kelas yang salah

    def __init__(self, name):
        self.name = name

    def add_trick(self, trick):
        self.tricks.append(trick)

>>> d = Dog('Fido')
>>> e = Dog('Buddy')
>>> d.add_trick('roll over')
>>> e.add_trick('play dead')
>>> d.tricks                # digunakan bersamaan untuk semua anjing
['roll over', 'play dead']
```

Berbagi Data Antarobjek (Solusi)

Solusinya hanya dengan mendesain proses instansiasi objek dengan benar

Example

```
class Dog:

    def __init__(self, name):
        self.name = name
        self.tricks = []    # membuat list kosong baru untuk setiap anjing

    def add_trick(self, trick):
        self.tricks.append(trick)

>>> d = Dog('Fido')
>>> e = Dog('Buddy')
>>> d.add_trick('roll over')
>>> e.add_trick('play dead')
>>> d.tricks
['roll over']
>>> e.tricks
['play dead']
```

Syntax untuk Inheritance

Definisi

```
class DerivedClassName(BaseClassName):
    <statement-1>
    .
    .
    .
    <statement-N>
```

Catatan: `BaseClassName` harus berada pada *scope* yang sama dengan pendefinisian kelas yang diturunkan.

Syntax untuk Inheritance

Definisi

```
class DerivedClassName(BaseClassName):
    <statement-1>
    .
    .
    .
    <statement-N>
```

Catatan: `BaseClassName` harus berada pada *scope* yang sama dengan pendefinisian kelas yang diturunkan.

Multiple Inheritance

```
class DerivedClassName(Base1, Base2, Base3):
    <statement-1>
    .
    .
    .
    <statement-N>
```

Meski *multiple inheritance* dapat dilakukan,
mungkin saja itu bukan solusi yang terbaik!

super()

Untuk memanggil metode yang telah diimplementasi pada *parent*, gunakan fungsi super()

Example

```
class Parent(object):

    def altered(self):
        print "PARENT altered()"

class Child(Parent):

    def altered(self):
        print "CHILD, BEFORE PARENT altered()"
        super(Child, self).altered()
        print "CHILD, AFTER PARENT altered()"

dad = Parent()
son = Child()

dad.altered()
son.altered()
```

Proyek Python

Packages

Contoh sebuah koleksi modul ("package") untuk mengolah *file* berupa suara

Struktur direktori

```
sound/                          Top-level package
    __init__.py                 Initialize the sound package
    formats/                    Subpackage for file format conversions
        __init__.py
        wavread.py
        wavwrite.py
        aiffread.py
        aiffwrite.py
        auread.py
        auwrite.py
        ...
    effects/                    Subpackage for sound effects
        __init__.py
        echo.py
        surround.py
        reverse.py
        ...
```

Contoh Penggunaan Package

Dari struktur *package* tersebut, modulnya dapat dimuat dengan cara berikut:

Example

```
from sound.effects import echo
```

Saran

Perhatian!

Example

```
from math import sqrt
sqrt(4)
```

atau

Example

```
import math
math.sqrt(4)
```

lebih disarankan dibandingkan

Example

```
from math import *
sqrt(4) # tidak jelas milik modul yang mana
```

Struktur Direktori Proyek

Untuk sebuah aplikasi, struktur minimal yang disarankan adalah sebagai berikut:

Direktori

```
skeleton/
    NAME/
        __init__.py
    bin/
    docs/
    setup.py
    tests/
        NAME_tests.py
        __init__.py
```

`NAME` dapat diganti dengan nama proyek Anda.

Errors & Exceptions

Ada dua jenis error:

1. syntax errors
2. exceptions

Syntax Errors

Dikenal juga dengan nama *parsing errors*, adalah error yang sering muncul saat masih berusaha menguasai bahasa pemrograman Python

Example

```
>>> while True print 'Hello world'
  File "<stdin>", line 1
    while True print 'Hello world'
                     ^
SyntaxError: invalid syntax
```

Exceptions

1. Di luar ekspresi yang kasus yang salah secara sintaksis, ada error yang muncul saat eksekusi program: *exceptions*
2. Punya beberapa tipe yang muncul bersamaan dengan pesan kesalahan
3. Sebagai programmer yang baik: baca pesan kesalahannya!

Contoh Exceptions

Example

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
```

Example

```
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'spam' is not defined
```

Example

```
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
```

Menangani Exceptions

Example

```
>>> while True:
...     try:
...         x = int(raw_input("Please enter a number: "))
...         break
...     except ValueError:
...         print "Oops! That was no valid number. Try again..."
...
```

Menangani Lebih dari Satu Jenis Exception

Example

```
import sys

try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except IOError as e:
    print "I/O error({0}): {1}".format(e.errno, e.strerror)
except ValueError:
    print "Could not convert data to an integer."
except:
    print "Unexpected error:", sys.exc_info()[0]
    raise
```

Catatan: except terakhir memungkinkan penggunaan *wildcard* - tanpa penyebutan jenis error. Ini dapat memudahkan penangkapan error yang belum diketahui, tapi menyulitkan proses *debugging*!

Built-in Exceptions

Beberapa jenis error sudah ada dalam modul `exceptions`, contohnya:

1. `AssertionError`
2. `IOError`
3. `ImportError`
4. `KeyboardInterrupt`
5. `NotImplementedError`

Lengkapnya bisa dilihat di `https://docs.python.org/2/library/exceptions.html#bltin-exceptions`

Membangkitkan Exceptions

Terkadang, kita perlu memanggil sendiri error yang ada (ingat `NotImplementedError`!). Pemanggilan ini dapat dilakukan dengan kata kunci `raise`.

Example

```
>>> try:
...     raise NameError('HiThere')
... except NameError:
...     print 'An exception flew by!'
...     raise
...
An exception flew by!
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
NameError: HiThere
```

Used-defined Exceptions

Jika diperlukan, kita dapat mendefinisikan kelas exception baru.

Example

```
>>> class MyError(Exception):
...     def __init__(self, value):
...         self.value = value
...     def __str__(self):
...         return repr(self.value)
...
>>> try:
...     raise MyError(2*2)
... except MyError as e:
...     print 'My exception occurred, value:', e.value
...
My exception occurred, value: 4
>>> raise MyError('oops!')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
__main__.MyError: 'oops!'
```

finally

Kita dapat mendefinisikan perintah yang harus dijalankan dalam situasi apapun, termasuk saat terjadi error.

Example

```
>>> def divide(x, y):
...     try:
...         result = x / y
...     except ZeroDivisionError:
...         print "division by zero!"
...     else:
...         print "result is", result
...     finally:
...         print "executing finally clause"
...
```

finally

Example

```
>>> divide(2, 1)
result is 2
executing finally clause
>>> divide(2, 0)
division by zero!
executing finally clause
>>> divide("2", "1")
executing finally clause
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in divide
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

assert

Pengujian paling sederhana di Python dapat dilakukan dengan perintah `assert`.

Example

```
def f(s):
    return s.upper().split(' ')

assert f('hello world') == ['HELLO', 'WORLD']
assert len(f('hello world')) == 2
```

Untuk pengelolaan kasus pengujian yang lebih baik, kita dapat menggunakan modul `unittest`.

Modul `unittest`

Beberapa konsep penting dalam otomasi pengujian dengan `unittest`:

1. **test fixture**: persiapan dan pembersihan pengujian, e.g. membuat basis data temporer
2. **test case**: unit terkecil pengujian, memeriksa apakah keluarannya sudah sesuai dengan yang diharapkan dari masukannya
3. **test suite**: sekumpulan *test cases* yang perlu dijalankan bersamaan
4. **test runner**: komponen yang menjalankan dan memberikan hasilnya kepada pengguna, e.g. GUI atau CLI

Contoh

Example (uji.py)

```
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

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

Kuis

Kuis

Buatlah sebuah program untuk menghitung luas *n* bangun datar yang terdiri dari:

1. Persegi
2. Persegi panjang
3. Segitiga
4. Lingkaran (gunakan 3.14 untuk nilai pi)
5. Trapesium

Permintaan masukan harus disesuaikan dengan bangun datar yang akan dihitung luasnya, e.g. jika persegi, maka program hanya akan meminta satu masukan, i.e. sisi. Setelah didapatkan *n* luas bangun datar, urutkan luas bangun datar dari terkecil hingga terbesar dan hitung total luas bangun datar tersebut!

Kuis

Soal 1 Buatlah penyelesaian kasus tersebut dalam paradigma pemrograman berorientasi objek! Petunjuk: Anda perlu membuat kelas *parent* dengan nama `BangunDatar`. Kelas `BangunDatar` akan memiliki metode kosong bernama `hitung_luas()`.

Soal 2 Buatlah masing-masing tiga kasus uji untuk kuis di minggu pertama tersebut! Tiap-tiap kasus uji fungsi perhitungan luas bangun datar tersebut harus memiliki:

1. kasus nilai benar (`assertEqual()`)
2. kasus nilai salah (`assertNotEqual()`)
3. kasus tipe salah (`assertRaises(TypeError)`)

Gunakan modul `unittest` dalam mengerjakan soal ini.

Soal 3 Buatlah deklarasi fungsi baru untuk menghitung luas segi enam! Namun, isi dari fungsi tersebut hanya mengeluarkan exception bahwa fungsi tersebut belum diimplementasi.

Terima kasih