Pengenalan Pemrograman Berorientasi Objek (OOP) - Inheritance

Mekanisme Pewarisan (Inheritance)

Paradigma Pemrograman Berorientasi Objek memiliki konsep pewarisan atau dalam bahasa Inggris disebut inheritance, tentunya di Python mendukung fitur ini.

Suatu kelas B dapat mewarisi kelas A, sehingga secara otomatis memiliki semua fitur yang dimiliki oleh kelas A, dalam hal ini atribut-atribut dan metode-metode.

Dalam contoh ini, kelas A disebut sebagai kelas dasar, yakni kelas yang memberikan warisan atau biasa juga disebut kelas yang diturunkan.

Kemudian kelas B disebut sebagai kelas turunan, yakni kelas yang mendapatkan warisan.

Jika di kelas B memiliki metode dengan nama yang sama dengan yang dimiliki kelas A, maka metode tersebut akan menimpa metode yang diwariskan dari kelas A.

Catatan:

- Frasa kelas dasar adalah terjemahan bahasa Inggris dari frasa base class.
- Frasa kelas turunan adalah terjemahan bahasa Inggris dari frasa derived class.
- Frasa menimpa metode adalah terjemahan bahasa Inggris dari frasa method override.
- Di Python, mekanisme pewarisan memungkinkan untuk memiliki lebih dari satu kelas dasar (kelas orang tua, yang diwarisi).

Referensi: https://docs.python.org/id/3.8/tutorial/classes.html#multiple-inheritance

Kita akan mengembangkan aplikasi yang sudah dimiliki di atas, *class Kalkulator* sebagai kelas dasar yang mempunyai fungsi melakukan penambahan melalui metode *tambah_angka*.

```
    """contoh kelas kalkulator sederhana. anggap kelas ini tidak boleh diubah!"""

    def __init__(self, nilai=0):
        self.nilai = nilai

    def tambah_angka(self, angka1, angka2):
        self.nilai = angka1 + angka2
        if self.nilai > 9:  # kalkulator sederhana hanya memroses sampai 9
            print('kalkulator sederhana melebihi batas angka: {}'.format(self.nilai))
        return self.nilai
```

Kemudian kita punya kebutuhan membuat sebuah kelas yang punya fitur perkalian tapi juga punya fitur penambahan, dalam contoh ini misalnya kita tidak boleh mengubah kalkulator yang sudah ada. Dibandingkan dengan membuat kelas baru kemudian menuliskan kembali implementasi penambahan angka, maka mewarisi kelas yang sudah ada akan lebih efisien.

Dari situ, kita membuat *class KalkulatorKali* yang mewarisi *class Kalkulator*.

```
class KalkulatorKali(Kalkulator):
    """contoh mewarisi kelas kalkulator sederhana"""

    def kali_angka(self, angka1, angka2):
        self.nilai = angka1 * angka2
        return self.nilai
```

Dengan pemanggilan *class KalkulatorKali* sebagai berikut.

```
kk = KalkulatorKali()
a = kk.kali_angka(2, 3)  # sesuai dengan definisi class memiliki fitur kali_angka
print(a)

b = kk.tambah_angka(5, 6)  # memiliki fitur tambah_angka karena mewarisi dari Kalkulator
print(b)
```

Dengan melakukan pewarisan, Anda dengan mudah bisa menambahkan (*extend*) kemampuan dari suatu class dengan fitur yang ingin Anda buat sendiri. Hal tersebut akan sangat berguna jika Anda ingin membuat aplikasi yang mudah diguna-ulang (*reusable*).

Menimpa (Override) Metode dengan Nama yang Sama Deng Dasar

tidak boleh mengubah *class Kalkulator* yang sudah ada.

Dalam proses pewarisan, kita bisa menimpa (*override*) definisi metode yang dimiliki oleh kelas dasar (kelas orang tua, yang diwarisi) dengan nama metode yang sama. Misalnya kita menimpa metode *tambah_angka* untuk menghilangkan batasan yang dimiliki.

```
class KalkulatorKali(Kalkulator):
    """contoh mewarisi kelas kalkulator sederhana"""

    def kali_angka(self, angka1, angka2):
        self.nilai = angka1 * angka2
        return self.nilai

    def tambah_angka(self, angka1, angka2):
        self.nilai = angka1 + angka2
        return self.nilai
```

Kemudian kita coba kembali, apakah batasan yang dimiliki sudah hilang?

```
kk = KalkulatorKali()

b = kk.tambah_angka(5, 6)  # fitur tambah_angka yang dipanggil milik KalkulatorKali
print(b)
```

Pemanggilan Metode Kelas Dasar dari Kelas Turunan dengan Sintaksis Super

Anggaplah fungsi *tambah_angka* adalah sebuah fungsi yang rumit, dimana kita sebaiknya gunakan saja kemampuan yang sudah ada di kelas dasar, kemudian kita hanya ubah sebagian fiturnya saja dengan yang kita inginkan.

```
    """contoh mewarisi kelas kalkulator sederhana"""

    def tambah_angka(self, angka1, angka2):
        if angka1 + angka2 <= 9:  # fitur ini sudah oke di kelas dasar, gunakan yang ada saja
            super().tambah_angka(angka1, angka2)  # panggil fungsi dari Kalkulator lalu isi nilai
        else:  # ini adalah fitur baru yang ingin diperbaiki dari keterbatasan kelas dasar
            self.nilai = angka1 + angka2
        return self.nilai
```

Variabel Privat di Python

Jika Anda sebelumnya pernah belajar bahasa pemrograman yang memiliki variabel privat, dimana variabel tersebut tidak dapat diakses kecuali dari objek yang bersangkutan, di Python hal tersebut tidak ada.

Terkait variabel privat tersebut, di Python ada konvensi dimana penggunaan nama yang diawali dengan garis bawah (underscore), baik itu fungsi, metode, maupun anggota data, akan dianggap sebagai non-publik.

Pernak-Pernik Terkait Struktur Data

Buat Anda yang pernah membuat program dengan menggunakan bahasa pemrograman C atau Pascal, Anda mungkin tertarik untuk membuat sebuah struktur data seperti halnya *struct* pada C atau *record* pada Pascal, bertujuan menyatukan sejumlah penamaan item data menjadi satu.

Dalam Python, dimana Anda sebelumnya pernah mempelajari mengenai *duck typing*, maka Anda cukup mendefinisikan saja sebuah class kosong, selanjutnya penamaan item data dapat secara langsung didefinisikan dan diisikan saat sudah *instantiation*.

```
class Pegawai:
    pass  # definisi class kosong

don = Pegawai()  # membuat Pegawai baru menjadi objek bernama don

# tambahkan item data pada objek sebagai record
don.nama = 'Don Doo'
don.bagian = 'IT'
don.gaji = 999
```

Dicoding Space
Jl. Batik Kumeli No.50, Sukaluyu,
Kec. Cibeunying Kaler, Kota Bandung
Jawa Barat 40123

Decode Ideas
Discover Potential

Tentang Kami

Blog
Reward
Showcase

Hubungi Kami
FAQ

Penghargaan

image
click bila
belum muncul