**Pengenalan Pemrograman Berorientasi Objek (OOP) dengan Python - Class, Objek, dan Method**

Kelas atau dalam bahasa Inggris disebut class, merupakan sebuah konsep yang menyediakan sarana untuk menyatukan data dan fungsionalitas secara satu kesatuan. Membuat sebuah kelas artinya membuat sebuah tipe baru, kemudian dengan membuat instance dari kelas tersebut akan menghasilkan objek baru dari tipe tersebut. Setiap objek (hasil instance dari kelas tersebut) dapat memiliki atribut untuk mengelola status dari objek tersebut, juga dapat memiliki metode untuk mengubah status atau informasinya.

Catatan:

* Kata objek adalah terjemahan bahasa Inggris dari kata object.
* Kata metode adalah terjemahan bahasa Inggris dari kata method.

Selanjutnya kita akan mempelajari secara mendalam implementasi kelas dan fitur-fitur terkait di bahasa pemrograman Python.

**Class**

Class merupakan sintaksis di Python yang menyediakan semua fitur-fitur standar dari Pemrograman Berorientasi Objek atau dalam bahasa Inggris disebut dengan Object Oriented Programming (OOP).

Definisi dari kelas menggunakan sintaksis class seperti hanya definisi fungsi yang menggunakan sintaksis def, kemudian perlu dipanggil (dieksekusi) dahulu sebelum dapat digunakan dan memiliki efek pada program.

1. class NamaKelas:
2. pass  # gantikan dengan pernyataan-pernyataan, misal: atribut atau metode

Pada pemanggilan sintaksis class tersebut, setelah seluruh pernyataan-pernyataan semuanya selesai diproses (didaftarkan sebagai atribut ataupun metode), maka kelas sudah dibuat dan dapat digunakan.

Sebuah kelas sendiri mendukung dua macam operasi:

1. Mengacu pada atribut.
2. Pembuatan instance atau dalam bahasa Inggris disebut instantiation.

Agar lebih jelas, kita akan membahas menggunakan contoh berikut.

1. class Kalkulator:
2. """contoh kelas kalkulator sederhana"""
3. i = 12345
5. def f(self):
6. return 'hello world'

Dari pembuatan class Kalkulator di atas, di dalamnya ada definisi atribut *i* dan definisi fungsi *f*.

Proses mengacu atribut yaitu*Kalkulator.i* dan *Kalkulator.f* sesuai definisi akan mengembalikan nilai integer dan fungsi. Pada proses mengacu atribut tersebut juga dapat mengubah nilainya, misalnya dengan memberikan bilangan bulat lain ke *Kalkulator.i* akan mengubah nilai yang ada saat ini.

1. Kalkulator.i = 1024  # maka nilai atribut i dalam Kalkulator berubah dari 12345 menjadi 1024

**Objek (object: an instance of a class)**

Pembahasan berikutnya adalah instantiation dari sebuah class, menggunakan notasi fungsi yaitu dengan kurung buka-kurung tutup, akan menghasilkan sebuah objek. Kemudian hasil instantiation ini biasanya disimpan dalam sebuah variabel dengan nama yang representatif.

Berikut ini adalah contoh membuat instance dari *class Kalkulator* menghasilkan sebuah objek.

1. k = Kalkulator()  # membuat instance dari kelas jadi objek, kemudian disimpan pada variabel k

Sebagai hasil instance sebuah class, suatu objek memiliki atribut dan metode yang didapatkan dari class. Sebuah metode atau dalam bahasa Inggris disebut method, adalah sebuah fungsi khusus yang menjadi "milik" suatu objek.

Untuk memanggil metode f dari objek k, hasil *instance* dari *class Kalkulator* di atas sebagai berikut.

1. print(k.f())  # akan mencetak hello world ke layar

Kenapa metode adalah sebuah fungsi khusus?  
Jika diperhatikan kembali fungsi f dalam definisi *class Kalkulator* memiliki satu argumen bernama *self*, sedangkan dalam pemanggilan metode dari objek k di atas tidak menggunakan argumen. Apabila f adalah fungsi biasa pada Python tentu pemanggilan ini akan mengembalikan kesalahan (error). Lebih detail mengenai konvensi ini akan dibahas pada bagian metode dari *class*.

Pembahasan lebih lanjut mengenai metode dari class ada di bagian selanjutnya.

**Class’ Constructor**

Kembali membahas proses instantiation dari class, sering ditemui kebutuhan mengeset nilai awal atau kondisi awal dari atribut yang dimiliki oleh class tersebut, sehingga untuk kebutuhan ini digunakan sebuah fungsi khusus yang biasa disebut sebagai pembangun atau dalam bahasa Inggris disebut *constructor*. Di Python, fungsi khusus atau metode sebagai constructor ini bernama \_\_init\_\_ atau biasa diucapkan sebagai "double underscore init". Pada saat dilakukan instantiation dari class, metode \_\_init\_\_ ini secara otomatis akan dipanggil di terlebih dahulu.

Berikut adalah definisi *class Kalkulator* di atas jika diubah dengan menggunakan constructor.

1. class Kalkulator:
2. """contoh kelas kalkulator sederhana"""
4. def \_\_init\_\_(self):
5. self.i = 12345
7. def f(self):
8. return 'hello world'

Nilai dari atribut i tidak terdefinisi pada awal definisi *Kalkulator*, setelah dilakukan instantiation maka nilai atribut i akan bernilai 12345. Meskipun bisa mendefinisikan variabel i sebagai atribut dari *class Kalkulator*, tetapi sebaiknya berhati-hati mengenai variabel yang akan terbagi (shared) untuk semua *instance* dari *class*, terutama untuk tipe yang dapat berubah (mutable), misalnya *list* dan *dictionary*.  
referensi: <https://docs.python.org/id/3.8/tutorial/classes.html#class-and-instance-variables>

1. class KeranjangBelanja:
2. """contoh tidak baik dilakukan dengan definisi variabel terbagi"""
3. isi = []  # menggunakan list di sini akan terbagi untuk semua instance. JANGAN DILAKUKAN

Lanjut pembahasan *constructor*, dengan dilengkapi *constructor* pun proses *instantiation* tidak berubah dari sebelumnya.

1. k = Kalkulator()  # membuat instance dari kelas jadi objek, kemudian disimpan pada variabel k

Lebih lanjut tentang constructor, tentu saja untuk mendukung aplikasi yang lebih dinamis maka constructor dapat memiliki parameter yang bisa dikirimkan saat proses instantiation, bahkan parameternya bisa lebih dari satu jika diperlukan.

Pada contoh berikut ini, constructor memiliki parameter i yang bersifat opsional, apabila dalam proses instantiation tidak dikirimkan parameter, secara otomatis i akan diisi nilai bawaan 12345.

1. class Kalkulator:
2. """contoh kelas kalkulator sederhana"""
4. def \_\_init\_\_(self, i=12345):
5. self.i = i  # i adalah variabel pada constructor, self.i adalah variabel dari class
7. def f(self):
8. return 'hello world'

Dengan contoh pemanggilan berikut.

1. k = Kalkulator(i=1024)  # melakukan instantiation sekaligus mengisi atribut i jadi 1024
2. print(k.i)             # mencetak atribut i dari objek k dengan keluaran nilai 1024

**Metode (Method)**

Pembahasan lebih detail mengenai metode, selain yang dibahas sebelumnya, kita akan membahas 3 jenis metode:

1. Metode dari objek (object method)
2. Metode dari class (class method)
3. Metode secara static (static method)

Pertama kita membahas metode dari objek, seperti yang sempat dijelaskan secara singkat di atas mengenai metode, atau dalam bahasa Inggris disebut method, secara umum metode adalah sebuah fungsi khusus yang menjadi “milik” suatu objek, yakni hasil instantiation dari *class*.

Salah satu hal khusus yang dimiliki oleh metode dengan adanya argumen bernama self, Anda tentu bertanya-tanya tentang argumen self pada metode-metode dalam kelas tersebut sebetulnya apa?  
Argumen pertama dari metode-metode dalam class, biasa diberikan nama self sebagai suatu konvensi atau standar penamaan, meskipun Anda bisa juga menggunakan nama lain. Bahkan dalam Python tidak ada arti khusus tentang sintaksis self ini, namun sangat disarankan menggunakan konversi ini agar program Python yang Anda buat akan lebih mudah dimengerti oleh pemrogram lainnya.

Seperti yang Anda sudah perkirakan, untuk sebuah metode, sebetulnya dikirimkan objek (hasil instance dari class) sebagai argumen pertamanya, dalam hal ini bernama self.

Misalnya menggunakan contoh di atas, jika k adalah objek hasil instance dari *class Kalkulator*, saat melakukan pemanggilan metode f.

1. k.f()

ekuivalen dengan

1. Kalkulator.f(k)

Argumen self pada metode f akan diisi dengan objek hasil instance dari *class Kalkulator*.

Sebelum kita membahas yang kedua dan ketiga, yakni metode dari class dan metode secara static, Anda tentu mengingat bahwa sebelumnya sudah belajar fungsi-fungsi bawaan (built-in) dari Python, antara lain: open, sorted, int, str, dan sejumlah lainnya. Terkait metode, ada dua fungsi bawaan yang akan kita bahas, yakni [classmethod](https://docs.python.org/id/3.8/library/functions.html#classmethod) dan [staticmethod](https://docs.python.org/id/3.8/library/functions.html#staticmethod).

Catatan:

fungsi decorator adalah sebuah fungsi yang mengembalikan fungsi lain, biasanya digunakan sebagai fungsi transformasi dengan "pembungkus" sintaksis @wrapper.

Referensi: <https://docs.python.org/id/3.8/glossary.html#term-decorator>.

Classmethod adalah sebuah fungsi yang mengubah metode menjadi metode dari class (class method). Dalam penggunaannya, fungsi ini dijadikan sebagai fungsi decorator *@classmethod*, kemudian pemanggilannya bisa langsung dari class yang terdefinisi ataupun melalui objek.  
Metode dari class (*class method*) menerima masukan class secara implisit sebagai argumen pertama yang secara konvensi diberikan nama cls.

Berdasar contoh yang sama dengan class sebelumnya, berikut adalah metode dari class.

1. class Kalkulator:
2. """contoh kelas kalkulator sederhana"""
4. def f(self):
5. return 'hello world'
7. @classmethod
8. def tambah\_angka(cls, angka1, angka2):
9. return '{} + {} = {}'.format(angka1, angka2, angka1 + angka2)

Nampak pada kode, sesuai konvensi ada metode yang menggunakan argumen pertama self, sedangkan untuk class method menggunakan konvensi argumen pertama *cls*.

Untuk melakukan pemanggilan dari class, dilakukan seperti berikut, dimana argumen pertama cls sudah mendapatkan masukan *class Kalkulator*.

1. Kalkulator.tambah\_angka(1, 2)  # tanpa perlu memberikan masukan untuk argumen cls

Metode dari *class* (class method) juga dapat dipanggil dari objek, hasil instantiation dari *class Kalkulator*, contohnya mirip seperti pemanggilan metode dari objek (object method).

1. k = Kalkulator()
2. print(k.tambah\_angka(1, 2))

*Staticmethod* adalah sebuah fungsi yang mengubah metode menjadi metode statis (static method). Dalam penggunaannya, fungsi ini dijadikan sebagai fungsi *decorator* *@staticmethod*, kemudian pemanggilannya bisa langsung dari *class* yang terdefinisi ataupun melalui objek.  
Metode statis (static method) tidak menerima masukan argumen pertama secara implisit.

Untuk Anda yang pernah memrogram Java atau C++, metode statis ini mirip seperti yang ada di bahasa pemrograman tersebut.

Berdasar contoh yang sama dengan *class* sebelumnya, berikut adalah metode statis.

1. class Kalkulator:
2. """contoh kelas kalkulator sederhana"""
4. def f(self):
5. return 'hello world'
7. @staticmethod
8. def kali\_angka(angka1, angka2):
9. return '{} x {} = {}'.format(angka1, angka2, angka1 \* angka2)

Nampak pada kode, tidak ada argumen pertama yang implisit seperti halnya pada dua metode sebelumnya.

Pemanggilan dari *class* seperti halnya pemanggilan fungsi biasa.

1. a = Kalkulator.kali\_angka(2, 3)
2. print(a)

Metode statis (*static method*) juga dapat dipanggil dari objek, hasil instantiation dari *class Kalkulator*, mirip seperti pemanggilan fungsi biasa meskipun dipanggil dari objek.

1. k = Kalkulator()
2. a = k.kali\_angka(2, 3)
3. print(a)

## 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.

1. class Kalkulator:
2. """contoh kelas kalkulator sederhana. anggap kelas ini tidak boleh diubah!"""
4. def \_\_init\_\_(self, nilai=0):
5. self.nilai = nilai
7. def tambah\_angka(self, angka1, angka2):
8. self.nilai = angka1 + angka2
9. if self.nilai > 9:  # kalkulator sederhana hanya memroses sampai 9
10. print('kalkulator sederhana melebihi batas angka: {}'.format(self.nilai))
11. 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.

1. class KalkulatorKali(Kalkulator):
2. """contoh mewarisi kelas kalkulator sederhana"""
4. def kali\_angka(self, angka1, angka2):
5. self.nilai = angka1 \* angka2
6. return self.nilai

Dengan pemanggilan class KalkulatorKali sebagai berikut.

1. kk = KalkulatorKali()
2. a = kk.kali\_angka(2, 3)  # sesuai dengan definisi class memiliki fitur kali\_angka
3. print(a)
5. b = kk.tambah\_angka(5, 6)  # memiliki fitur tambah\_angka karena mewarisi dari Kalkulator
6. 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 Dengan Kelas Dasar

Anda melihat bahwa kalkulator yang anda operasikan mendapatkan peringatan melebihi batas angka yang bisa diproses? Lalu bagaimana kalau Anda ingin mengubah keterbatasan itu? Meski dalam contoh ini anggap tetap 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.

1. class KalkulatorKali(Kalkulator):
2. """contoh mewarisi kelas kalkulator sederhana"""
4. def kali\_angka(self, angka1, angka2):
5. self.nilai = angka1 \* angka2
6. return self.nilai
8. def tambah\_angka(self, angka1, angka2):
9. self.nilai = angka1 + angka2
10. return self.nilai

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

1. kk = KalkulatorKali()
3. b = kk.tambah\_angka(5, 6)  # fitur tambah\_angka yang dipanggil milik KalkulatorKali
4. 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.

1. class KalkulatorTambah(Kalkulator):
2. """contoh mewarisi kelas kalkulator sederhana"""
4. def tambah\_angka(self, angka1, angka2):
5. if angka1 + angka2 <= 9:  # fitur ini sudah oke di kelas dasar, gunakan yang ada saja
6. super().tambah\_angka(angka1, angka2)  # panggil fungsi dari Kalkulator lalu isi nilai
7. else:  # ini adalah fitur baru yang ingin diperbaiki dari keterbatasan kelas dasar
8. self.nilai = angka1 + angka2
9. 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.

1. class Pegawai:
2. pass  # definisi class kosong
4. don = Pegawai()  # membuat Pegawai baru menjadi objek bernama don
6. # tambahkan item data pada objek sebagai record
7. don.nama = 'Don Doo'
8. don.bagian = 'IT'
9. don.gaji = 999