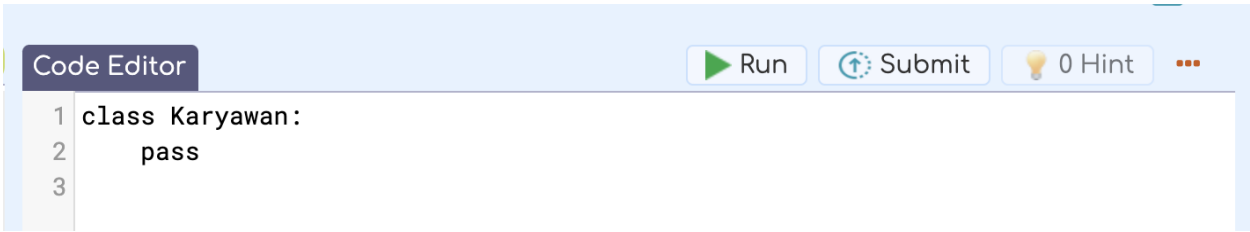
Class dan Objek dalam Python - Part 1

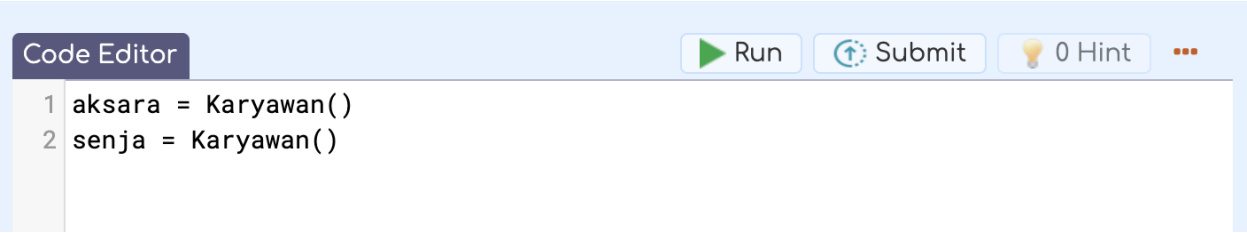
Setiap objek yang aku representasikan dalam program berbasis OOP merupakan instansi/ bentuk nyata dari sebuah konsep yang disebut dengan *class*. Oleh karena itu, *class* dapat juga aku sebutkan sebagai kerangka utama (blueprint) dari objek. Untuk mempermudah pemahaman konsep OO, aku menggunakan contoh berikut:

Asumsikan aku ingin merepresentasikan Diriku dan Senja sebagai karyawan di suatu perusahaan X. Untuk merepresentasikan Diriku dan Senja, aku dapat membuat sebuah *class* yang nantinya akan mencakup properti-properti yang umumnya dimiliki oleh sebuah karyawan.

Pada bahasa Python aku dapat membuat sebuah class dengan menggunakan *syntax* berikut.



Kemudian, dari class yang telah aku definisikan, aku dapat menciptakan objek (Diriku dan Senja) dari sebuah *class* dengan menggunakan s*yntax* berikut.



Pada contoh di atas, Aku dan senja merupakan realisasi (objek) dari *class* Karyawan yang sebelumnya telah aku buat. Dari potongan kode ini, **aksara** dan **senja** adalah dua objek yang berbeda.

Class dan Objek dalam Python - Part 2

Pada bagian pertama, aku telah berhasil membuat sebuah *class* dan objek-objek sebagai bentuk realisasi dari sebuah *class*. Akan tetapi, *class* yang telah aku definisikan belum memiliki atribut ataupun fungsi-fungsi yang dapat merepresentasikan objek Karyawan dengan baik.

Agar dapat membuat *class* Karyawan dengan baik, pertama, aku akan mempelajari cara merepresentasikan atribut/properti dalam sebuah class. Dalam sebuah *class*, aku dapat mendefinisikan dua jenis atribut yaitu.

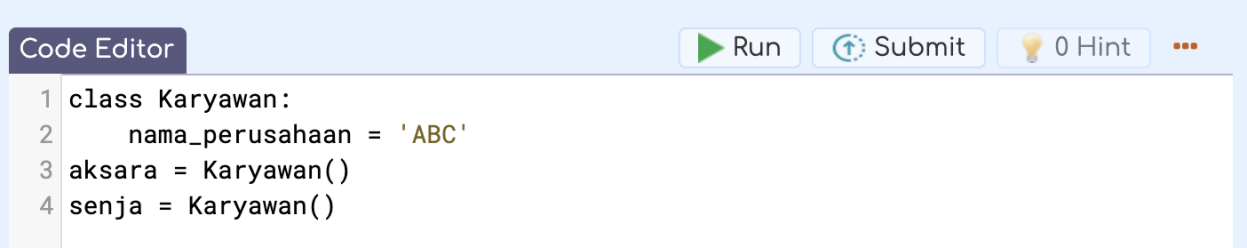
1. **Class Attribute** adalah properti/atribut yang bernilai sama untuk oleh seluruh objek

2. **Instance Attribute** adalah properti/atribut yang nilainya berbeda-beda untuk setiap objek dari sebuah *class*.

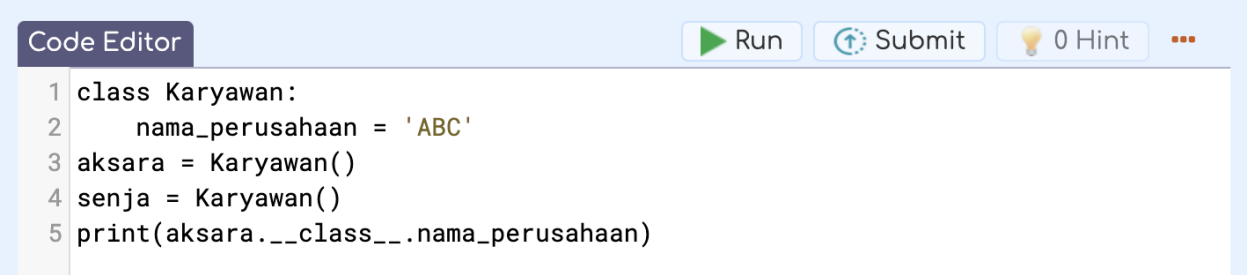
Class dan Objek dalam Python - Part 3

Berkaitan dengan kedua jenis atribut yang telah aku pelajari, aku menggunakan contoh berikut untuk memperkuat pemahamanku terkait dengan konsep **class attribute**.

Class Karyawan pada umumnya memiliki beberapa atribut seperti nama, usia, pendapatan serta nama perusahaan di mana karyawan tersebut bekerja. Untuk merepresentasikan diriku dan Senja sebagai karyawan yang bekerja di sebuah perusahaan yang sama (anggap saja perusahan ABC), aku dapat merepresentasikan dengan menggunakan konsep class attribute



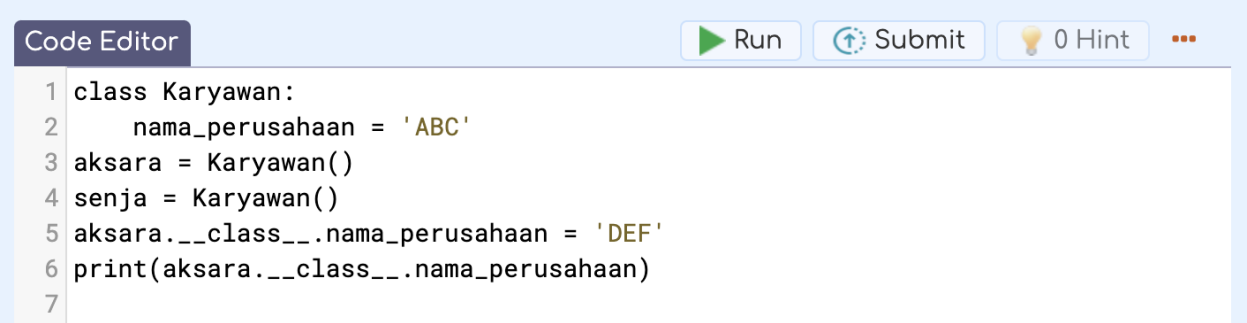
Menggunakan potongan kode di atas, nama_perusahaan sebagai class attribute dapat kita akses dengan menggunakan *syntax*:



akan menghasilkan output: **ABC**

Kemudian, sesuai dengan konsep yang telah aku pelajari sebelumnya, saat aku mengubah nilai atribut yang merupakan sebuah *class attribute*, nilai dari atribut akan berubah untuk seluruh objek.

Saat perusahaan berubah nama, misalkan nama perusahaan berubah dari 'ABC' ke 'DEF', dikarenakan atribut nama_perusahaan merupakan sebuah class attribute, aku hanya cukup mengganti nama_perusahaan pada salah satu objek saja (tidak perlu mengganti nama_perusahaan milik seluruh objek). Contohnya:



akan menghasilkan output: **DEF**

Dengan merubah nama perusahaan, maka nama perusahaan milik objek Senja juga secara otomatis berubah menjadi **DEF**.

Tugas:

Aku di minta untuk mengerjakan tutorial sederhana untuk membantu memahami konsep OOP:

1. Definisikan Class Karyawan
2. Inisiasi object yang dinyatakan dalam variabel Aksara dan Senja
3. Cetak Nama Perusahaan melalui penggunaan keyword __class__
4. Ubah Nama Perusahaan menjadi 'DEF'

Class dan Objek dalam Python - Part 4

Pada bagian sebelumnya aku telah mempelajari contoh deklarasi class Karyawan; nama, usia dan pendapatan karyawan adalah contoh dari konsep instance attribute. Hal ini dikarenakan setiap karyawan tentunya dapat memiliki nama, usia dan pendapatan yang berbeda.

Tugas:

Untuk merepresentasikan instance attribute milik Aksara, aku mengetik potongan kode berikut pada live code editor:

Code Editor Run Submit 0 Hint

```
class Karyawan:
    nama_perusahaan = 'ABC'
    def __init__(self, nama, usia, pendapatan):
        self.nama = nama
        self.usia = usia
        self.pendapatan = pendapatan
aksara = Karyawan('Aksara', 25, 8500000)
senja = Karyawan('Senja', 28, 12500000)
print(aksara.nama + ', Usia: ' + str(aksara.usia) + ', Pendapatan ' +
str(aksara.pendapatan))
```

menghasilkan output:

```
Aksara, Usia 25, Pendapatan: 8500000
```

Selanjutnya, untuk merepresentasikan instance attribute milik Senja, aku mengetik potongan kode berikut pada live code editor:

Code Editor Run Submit 0 Hint

```
class Karyawan:
    nama_perusahaan = 'ABC'
    def __init__(self, nama, usia, pendapatan):
        self.nama = nama
        self.usia = usia
        self.pendapatan = pendapatan
aksara = Karyawan('Aksara', 25, 8500000)
senja = Karyawan('Senja', 28, 12500000)
print(senja.nama + ', Usia: ' + str(senja.usia) + ', Pendapatan ' +
str(senja.pendapatan))

```

menghasilkan output:

```
Senja, Usia 28, Pendapatan: 12500000
```

Penjelasan:

Dari potongan kode di atas, atribut nama, usia dan pendapatan merupakan contoh dari instance variabel. Sebagai tambahan, fungsi **__init__()** di dalam class Karyawan secara khusus disebut sebagai constructor. Melalui sebuah constructor, aku dapat meng-assign (menginisialisasi) atribut-atribut milik sebuah objek.

Pada bahasa pemrograman Python, setiap fungsi (termasuk constructor) akan menerima dirinya sendiri (self) sebagai parameter pertama dari fungsi. Kemudian, aku dapat menambahkan parameter-parameter lain setelah parameter self sesuai dengan kebutuhan. Seperti pada contoh di atas, saat objek dibuat (diinisialisasi), aku dapat melemparkan nama, usia dan pendapatan melalui syntax,

```
aksara = Karyawan('Aksara', 25, 8500000)
```

Terakhir, aku belajar bahwa objek aksara dan senja diizinkan untuk memiliki nama, usia dan pendapatan yang berbeda. Untuk mengakses instance attribute dalam sebuah class, aku perlu menuliskan sintaks self diikuti dengan tanda titik (.) sebelum nama atribut.

Behavior pada Class

Selain dapat mendefinisikan atribut, dalam sebuah class, aku diperbolehkan untuk mendefinisikan fungsi-fungsi *(behavior)* dari sebuah class.

Dari potongan kode yang telah aku gunakan, aku dapat menambahkan fungsi-fungsi berkaitan dengan class Karyawan. Sebagai contoh, seorang karyawan tentunya mungkin saja memiliki pendapatan tambahan berdasarkan banyaknya kerja lembur dan jumlah proyek yang telah diselesaikan.

Tugas:

Untuk menghitung pendapatan tambahan dari jumlah kerja lembur dan jumlah proyek yang diselesaikan oleh seorang karyawan dan mengakses pendapatan total dari seorang karyawan, aku dapat menuliskan potongan kode berikut.

Code Editor Run Submit 0 Hint

```
class Karyawan:
    nama_perusahaan = 'ABC'
    insentif_lembur = 250000
    def __init__(self, nama, usia, pendapatan):
        self.nama = nama
        self.usia = usia
        self.pendapatan = pendapatan
        self.pendapatan_tambahan = 0
    def lembur(self):
        self.pendapatan_tambahan += self.insentif_lembur
    def tambahan_proyek(self, insentif_proyek):
        self.pendapatan_tambahan += insentif_proyek
    def total_pendapatan(self):
        return self.pendapatan + self.pendapatan_tambahan
aksara = Karyawan('Aksara', 25, 8500000)
senja = Karyawan('Senja', 28, 12500000)
```

untuk menambahkan pendapatan lembur diriku, aku dapat menggunakan fungsi lembur() pada objek aksara.

Code Editor Run Submit 0 Hint

```
class Karyawan:
    nama_perusahaan = 'ABC'
    insentif_lembur = 250000
    def __init__(self, nama, usia, pendapatan):
        self.nama = nama
        self.usia = usia
        self.pendapatan = pendapatan
        self.pendapatan_tambahan = 0
    def lembur(self):
        self.pendapatan_tambahan += self.insentif_lembur
    def tambahan_proyek(self, insentif_proyek):
        self.pendapatan_tambahan += insentif_proyek
    def total_pendapatan(self):
        return self.pendapatan + self.pendapatan_tambahan
aksara = Karyawan('Aksara', 25, 8500000)
senja = Karyawan('Senja', 28, 12500000)
aksara.lembur()

```

untuk menambahkan pendapatan tambahan proyek pada senja, aku dapat mengakses fungsi tambahan_proyek() pada objek senja

Code Editor Run Submit 0 Hint

```
class Karyawan:
    nama_perusahaan = 'ABC'
    insentif_lembur = 250000
    def __init__(self, nama, usia, pendapatan):
        self.nama = nama
        self.usia = usia
        self.pendapatan = pendapatan
        self.pendapatan_tambahan = 0
    def lembur(self):
        self.pendapatan_tambahan += self.insentif_lembur
    def tambahan_proyek(self, insentif_proyek):
        self.pendapatan_tambahan += insentif_proyek
    def total_pendapatan(self):
        return self.pendapatan + self.pendapatan_tambahan
aksara = Karyawan('Aksara', 25, 8500000)
senja = Karyawan('Senja', 28, 12500000)
aksara.lembur()
senja.tambahan_proyek(2500000)

```

Selanjutnya, aku dapat menghitung total pendapatanku dan Senja

Code Editor Run Submit 0 Hint

```
class Karyawan:
    nama_perusahaan = 'ABC'
    insentif_lembur = 250000
    def __init__(self, nama, usia, pendapatan):
        self.nama = nama
        self.usia = usia
        self.pendapatan = pendapatan
        self.pendapatan_tambahan = 0
    def lembur(self):
        self.pendapatan_tambahan += self.insentif_lembur
    def tambahan_proyek(self, insentif_proyek):
        self.pendapatan_tambahan += insentif_proyek
    def total_pendapatan(self):
        return self.pendapatan + self.pendapatan_tambahan
aksara = Karyawan('Aksara', 25, 8500000)
senja = Karyawan('Senja', 28, 12500000)
aksara.lembur()
senja.tambahan_proyek(2500000)
print('Pendapatan Total Aksara: ' + str(aksara.total_pendapatan()))
print('Pendapatan Total Senja: ' +str(senja.total_pendapatan()))
```

Layaknya proses pendefinisian fungsi pada Python, fungsi-fungsi dalam sebuah class juga dapat memiliki parameter (seperti fungsi tambahan_proyek dalam contoh) ataupun mengembalikan sebuah nilai (seperti fungsi total_pendapatan dalam contoh).