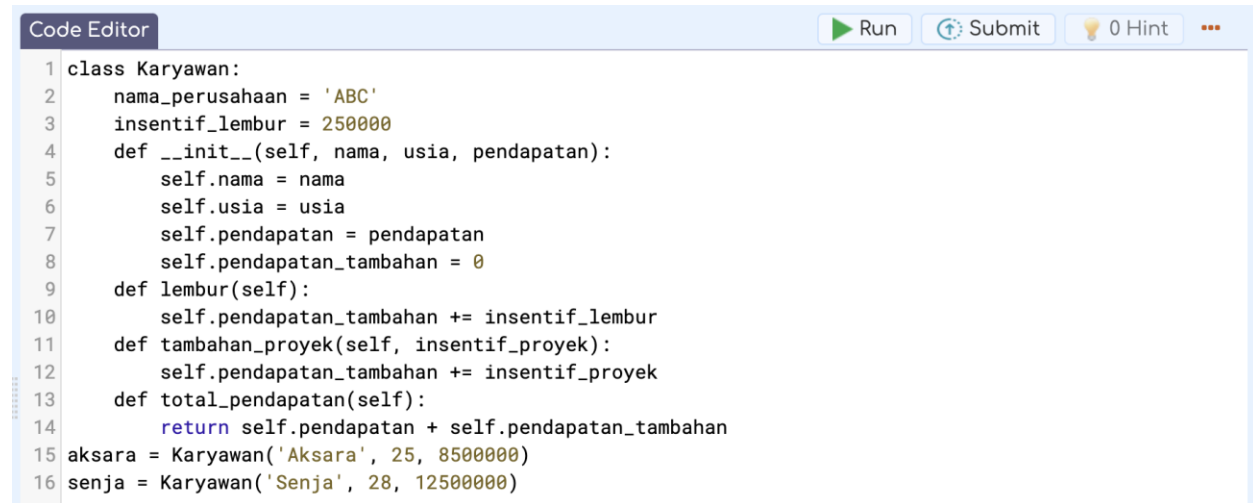


Encapsulation pada Python - Part 1

Praktik tadi yang dibantu Senja, membuatku lebih percaya diri untuk lanjut ke materi selanjutnya mengenai “Encapsulation & Inheritance”. Ini apa lagi yah? Aku membolak-balik halaman modul materi ini. Aku bersemangat untuk memulai!

Enkapsulasi (*Encapsulation*) adalah sebuah teknik dalam OOP yang mengizinkan aku untuk menyembunyikan detail dari sebuah atribut dalam sebuah class. Pada contoh-contoh sebelumnya, setiap atribut dan fungsi yang telah aku definisikan belum menggunakan konsep enkapsulasi, yang mengartikan bahwa setiap atribut dan fungsi dapat diakses di luar class.

Pada potongan kode yang sebelumnya aku tuliskan,

A screenshot of a code editor window titled "Code Editor". The editor contains Python code for a class named "Karyawan". The code defines attributes like "nama_perusahaan", "insentif_lembur", and methods like "__init__", "lembur", "tambahan_proyek", and "total_pendapatan". It also shows two instances of the class being created: "aksara" and "senja". The code is as follows:

```
1 class Karyawan:
2     nama_perusahaan = 'ABC'
3     insentif_lembur = 250000
4     def __init__(self, nama, usia, pendapatan):
5         self.nama = nama
6         self.usia = usia
7         self.pendapatan = pendapatan
8         self.pendapatan_tambahan = 0
9     def lembur(self):
10        self.pendapatan_tambahan += insentif_lembur
11    def tambahan_proyek(self, insentif_proyek):
12        self.pendapatan_tambahan += insentif_proyek
13    def total_pendapatan(self):
14        return self.pendapatan + self.pendapatan_tambahan
15 aksara = Karyawan('Aksara', 25, 8500000)
16 senja = Karyawan('Senja', 28, 12500000)
```

Aku dapat mengakses setiap atribut milik class Karyawan secara langsung di luar scope milik **class Karyawan**. Sebagai contoh,

```
Code Editor
Run Submit 0 Hint
1 class Karyawan:
2     nama_perusahaan = 'ABC'
3     insentif_lembur = 250000
4     def __init__(self, nama, usia, pendapatan):
5         self.nama = nama
6         self.usia = usia
7         self.pendapatan = pendapatan
8         self.pendapatan_tambahan = 0
9     def lembur(self):
10        self.pendapatan_tambahan += insentif_lembur
11    def tambahan_proyek(self, insentif_proyek):
12        self.pendapatan_tambahan += insentif_proyek
13    def total_pendapatan(self):
14        return self.pendapatan + self.pendapatan_tambahan
15 aksara = Karyawan('Aksara', 25, 8500000)
16 senja = Karyawan('Senja', 28, 12500000)
17 print(aksara.nama)
```

akan menghasilkan output: **Aksara**

```
Code Editor
Run Submit 0 Hint
1 class Karyawan:
2     nama_perusahaan = 'ABC'
3     insentif_lembur = 250000
4     def __init__(self, nama, usia, pendapatan):
5         self.nama = nama
6         self.usia = usia
7         self.pendapatan = pendapatan
8         self.pendapatan_tambahan = 0
9     def lembur(self):
10        self.pendapatan_tambahan += insentif_lembur
11    def tambahan_proyek(self, insentif_proyek):
12        self.pendapatan_tambahan += insentif_proyek
13    def total_pendapatan(self):
14        return self.pendapatan + self.pendapatan_tambahan
15 aksara = Karyawan('Aksara', 25, 8500000)
16 senja = Karyawan('Senja', 28, 12500000)
17 print(senja.nama)
```

akan menghasilkan output: **Senja**

Encapsulation pada Python – Part 2

Pada contoh di atas, terlihat bahwa atribut nama pada setiap objek dapat diakses secara bebas di luar scope dari sebuah class. Agar suatu properti ataupun fungsi dari sebuah class tidak dapat diakses secara bebas di luar scope milik suatu class, aku dapat mendefinisikan access modifier (level akses) saat sebuah atribut/fungsi didefinisikan.

Terdapat 2 macam access modifier dalam Python, yakni.

1. **Public access:** dapat aku definisikan dengan secara langsung menuliskan nama dari atribut/fungsi. Dalam sebuah objek, atribut/fungsi yang bersifat public access dapat diakses di luar scope sebuah class
2. **Private access:** dapat aku definisikan dengan menambahkan **double underscore** (__) sebelum menuliskan nama dari atribut/fungsi. Dalam sebuah objek, atribut/fungsi yang bersifat private access hanya dapat diakses di dalam scope sebuah class.

Tugas:

Untuk memperkuat pemahamanku terkait konsep enkapsulasi dalam paradigma OO, aku menggunakan contoh syntax dan mengetiknnya pada live code editor:

```
Code Editor Run Submit 0 Hint ...
1 class Karyawan:
2     nama_perusahaan = 'ABC'
3     __insentif lembur = 250000
4     def __init__(self, nama, usia, pendapatan):
5         self.__nama = nama
6         self.__usia = usia
7         self.__pendapatan = pendapatan
8         self.__pendapatan_tambahan = 0
9     def lembur(self):
10        self.__pendapatan_tambahan += self.__insentif_lembur
11    def tambahan_proyek(self, insentif_proyek):
12        self.__pendapatan_tambahan += insentif_proyek
13    def total_pendapatan(self):
14        return self.__pendapatan + self.__pendapatan_tambahan
15 aksara = Karyawan('Aksara', 25, 8500000)
```

Pada potongan kode di atas, atribut **nama_perusahaan** bersifat **public** yang mengartikan bahwa aku dapat mengakses atribut ini di luar scope class Karyawan.

```
Code Editor Run Submit 0 Hint ...
1 class Karyawan:
2     nama_perusahaan = 'ABC'
3     __insentif_lembur = 250000
4     def __init__(self, nama, usia, pendapatan):
5         self.__nama = nama
6         self.__usia = usia
7         self.__pendapatan = pendapatan
8         self.__pendapatan_tambahan = 0
9     def lembur(self):
10        self.__pendapatan_tambahan += self.__insentif_lembur
11    def tambahan_proyek(self, insentif_proyek):
12        self.__pendapatan_tambahan += insentif_proyek
13    def total_pendapatan(self):
14        return self.__pendapatan + self.__pendapatan_tambahan
15 aksara = Karyawan('Aksara', 25, 8500000)
16 print(aksara.__class__.nama_perusahaan)
```

tidak akan menyebabkan error dan menghasilkan output: **ABC**

Kemudian, atribut `__nama`, `__usia`, `__pendapatan_tambahan`, `__insentif_lembur` dan `pendapatan` bersifat **private** sehingga atribut ini hanya dapat diakses di dalam scope class `Karyawan`.

Saat aku mencoba mengakses atribut-atribut ini di luar scope class `Karyawan`, Python akan mengembalikan error yang menyatakan bahwa **class `Karyawan`** tidak memiliki atribut tersebut. Sebagai contoh,

```
Code Editor Run Submit 0 Hint ...
1 class Karyawan:
2     nama_perusahaan = 'ABC'
3     __insentif_lembur = 250000
4     def __init__(self, nama, usia, pendapatan):
5         self.__nama = nama
6         self.__usia = usia
7         self.__pendapatan = pendapatan
8         self.__pendapatan_tambahan = 0
9     def lembur(self):
10        self.__pendapatan_tambahan += self.__insentif_lembur
11    def tambahan_proyek(self, insentif_proyek):
12        self.__pendapatan_tambahan += insentif_proyek
13    def total_pendapatan(self):
14        return self.__pendapatan + self.__pendapatan_tambahan
15 aksara = Karyawan('Aksara', 25, 8500000)
16 print(aksara.__nama)
```

akan menyebabkan error yang menyatakan bahwa objek `Karyawan` tidak memiliki atribut `__nama`.

```
Console Graphic
File "<stdin>", line 18, in <module>
AttributeError: 'Karyawan' object has no attribute '__nama'
```