Pengantar Object Oriented Programming

Apa itu Object Oriented Programming?

Pemrograman Berbasis Objek (Object Oriented Programming - OOP) merupakan salah satu paradigma pemrograman yang cukup populer di antara paradigma-paradigma lainnya.

Pada paradigma OOP, struktur dari sebuah program dikemas ke dalam sebuah objek yang memiliki serangkaian properti (properties) dan fungsi (behaviours). Sebagai contoh, aku dapat merepresentasikan seorang karyawan ke dalam sebuah program melalui konsep OOP.

Seorang karyawan dapat memiliki serangkaian properti seperti nama, usia, keahlian, dll. Kemudian, seorang karyawan juga dapat memiliki fungsi-fungsi seperti hadir ke kantor, absen, lembur, tugas dinas, dll.

Konsep dalam Object Oriented Programming

Sebagai salah satu bahasa pemrograman yang bersifat multi-purposive, Python juga mendukung paradigma Object Oriented (OO).

Konsep OO pada Python memiliki tujuan untuk menciptakan potongan-potongan kode yang bersifat reusable dan tidak redundan. Konsep ini dikenal dengan istilah konsep DRY - Don't Repeat Yourself (berlawanan dengan konsep WET - Write Everything Twice).

Dalam bahasa pemrograman Python, terdapat 3 konsep utama OO yaitu.

1. **Encapsulation: Menyembunyikan sebagian detail yang dimiliki oleh sebuah objek terhadap objek-objek lainnya.**
2. **Inheritance: Menurunkan serangkaian fungsi-fungsi yang dimiliki oleh sebuah objek ke sebuah objek baru tanpa mengubah makna dari objek acuan yang digunakan.**
3. **Polymorphism: Konsep untuk menggunakan fungsi-fungsi dengan nama/ tujuan yang sama dengan cara yang berbeda.**

Classes & Objects

Class dan Objek dalam Python

In [ ]:

```
# Definisikan class Karyawan
class Karyawan:
 nama_perusahaan = 'ABC'
# Inisiasi object yang dinyatakan dalam variabel aksara dan senja
aksara = Karyawan()
senja = Karyawan()
# Cetak nama perusahaan melalui penggunaan keyword __class__
# pada class attribute nama_perusahaan
print(aksara.__class__.nama_perusahaan)
# Ubah nama_perusahaan menjadi "DEF"
aksara.__class__.nama_perusahaan = 'DEF'
# Cetak nama_perusahaan objek aksara dan senja
print(aksara.__class__.nama_perusahaan)
print(senja.__class__.nama_perusahaan)
```

```
ABC
DEF
DEF
```

Class dan Objek dalam Python - Part 2

In [ ]:

```
# Definisikan class Karyawan
class Karyawan:
    nama_perusahaan = 'ABC'
    def __init__(self, nama, usia, pendapatan):
        self.nama = nama
        self.usia = usia
        self.pendapatan = pendapatan
# Buat object bernama aksara dan senja
aksara = Karyawan('Aksara', 25, 8500000)
senja = Karyawan('Senja', 28, 12500000)
# Cetak objek bernama aksara
print(aksara.nama + ', Usia: ' + str(aksara.usia) + ', Pendapatan ' + str(aksara.pendapa
tan))
# Cetak objek bernama senja
print(senja.nama + ', Usia: ' + str(senja.usia) + ', Pendapatan ' + str(senja.pendapatan
))
```

```
Aksara, Usia: 25, Pendapatan 8500000
Senja, Usia: 28, Pendapatan 12500000
```

Behavior pada Class

In [ ]:

```
# Definisikan class Karyawan berikut dengan attribut dan fungsinya
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
# Buat object dari karwayan bernama Aksara dan Senja
aksara = Karyawan('Aksara', 25, 8500000)
senja = Karyawan('Senja', 28, 12500000)
# Aksara melaksanakan lembur
aksara.lembur()
# Senja memiliki proyek tambahan
senja.tambahan_proyek(2500000)
# Cetak pendapatan total Aksara dan Senja
print('Pendapatan Total Aksara: ' + str(aksara.total_pendapatan()))
print('Pendapatan Total Senja: ' +str(senja.total_pendapatan()))
```

```
Pendapatan Total Aksara: 8750000
Pendapatan Total Senja: 15000000
```

Tugas Praktek

In [ ]:

```
# Definisikan class Karyawan
class Karyawan:
    def __init__(self, nama, usia, pendapatan, insentif_lembur):
        self.nama = nama
        self.usia = usia
        self.pendapatan = pendapatan
        self.pendapatan_tambahan = 0
        self.insentif_lembur = insentif_lembur
    def lembur(self):
        self.pendapatan_tambahan += self.insentif_lembur
    def tambahan_proyek(self, jumlah_tambahan):
        self.pendapatan_tambahan += jumlah_tambahan
```

```
    def total_pendapatan(self):
        return self.pendapatan + self.pendapatan_tambahan
# Definisikan class Perusahaan
class Perusahaan:
    def __init__(self, nama, alamat, nomor_telepon):
        self.nama = nama
        self.alamat = alamat
        self.nomor_telepon = nomor_telepon
        self.list_karyawan = []
    def aktifkan_karyawan(self, karyawan):
        self.list_karyawan.append(karyawan)
    def nonaktifkan_karyawan(self, nama_karyawan):
        karyawan_nonaktif = None
        for karyawan in self.list_karyawan:
            if karyawan.nama == nama_karyawan:
                karyawan_nonaktif = karyawan
                break
        if karyawan_nonaktif is not None:
            self.list_karyawan.remove(karyawan_nonaktif)
```

Tugas Praktek

In [ ]:

```
# Definisikan perusahaan
perusahaan = Perusahaan('ABC', 'Jl.Jendral Sudirman, Blok 11', '(021)95205XX')
# Definisikan nama-nama karyawan
karyawan_1 = Karyawan('Ani', 25, 8500000, 100000)
karyawan_2 = Karyawan('Budi', 28, 12000000, 150000)
karyawan_3 = Karyawan('Cici', 30, 15000000, 200000)
# Aktifkan karyawan di perusahaan ABC
perusahaan.aktifkan_karyawan(karyawan_1)
perusahaan.aktifkan_karyawan(karyawan_2)
perusahaan.aktifkan_karyawan(karyawan_3)
```

Encapsulation & Inheritance

Encapsulation pada Python

In [ ]:

```
# Definisikan class Karyawan
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
# Buat objek karyawan bernama Aksara
aksara = Karyawan('Aksara', 25, 8500000)
# Akses ke attribute class Karyawan
print(aksara.__class__)
# Akan menimbulkan error ketika di run
print(aksara.__nama)
```

```
<class '__main__.Karyawan'>

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-6-e061f662a501> in <module>()
     19 print(aksara.__class__)
```

```
     20 # Akan menimbulkan error ketika di run
---> 21 print(aksara.__nama)

AttributeError: 'Karyawan' object has no attribute '__nama'
```

Inheritance pada Python – Part 1

In [ ]:

```
# Definisikan class Karyawan (sebagai base class)
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
# Buat class turunan (sebagai inherit class) dari class karyawan,
# yaitu class AnalisData
class AnalisData(Karyawan):
    def __init__(self, nama, usia, pendapatan):
    # melakukan pemanggilan konstruktur class Karyawan
        super().__init__(nama, usia, pendapatan)
# Buat kembali class turunan (sebagai inherit class) dari class karyawan,
# yaitu class IlmuwanData
class IlmuwanData(Karyawan):
    def __init__(self, nama, usia, pendapatan):
        # melakukan pemanggilan konstruktur class Karyawan
        super().__init__(nama, usia, pendapatan)
# Buat objek karyawan yang bekerja sebagai AnalisData
aksara = AnalisData('Aksara', 25, 8500000)
aksara.lembur()
print(aksara.total_pendapatan())
# Buat objek karyawan yang bekerja sebagai IlmuwanData
senja = IlmuwanData('Senja', 28, 13000000)
senja.tambahan_proyek(2000000)
print(senja.total_pendapatan())
```

```
8750000
15000000
```

Inheritance pada Python – Part 2

In [ ]:

```
# Definisikan class Karyawan (sebagai base class)
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
# Buat class turunan (sebagai inherit class) dari class karyawan,
# yaitu class AnalisData
class AnalisData(Karyawan):
```

```
    def __init__(self, nama, usia, pendapatan):
    # melakukan pemanggilan konstruktur class Karyawan
        super().__init__(nama, usia, pendapatan)
# Buat kembali class turunan (sebagai inherit class) dari class karyawan,
# yaitu class IlmuwanData
class IlmuwanData(Karyawan):
    # mengubah atribut insentif_lembur yang digunakan pada fungsi lembur()
    insentif_lembur = 500000
    def __init__(self, nama, usia, pendapatan):
        super().__init__(nama, usia, pendapatan)
# Buat objek karyawan yang bekerja sebagai AnalisData
aksara = AnalisData('Aksara', 25, 8500000)
aksara.lembur()
print(aksara.total_pendapatan())
# Buat objek karyawan yang bekerja sebagai IlmuwanData
senja = IlmuwanData('Senja', 28, 13000000)
senja.lembur()
print(senja.total_pendapatan())
```

```
8750000
13500000
```

Polymorphism & Overloading

Polymorphism pada Python - Part 1

In [ ]:

```
# Definisikan class Karyawan (sebagai base class)
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
# Buat class turunan (sebagai inherit class) dari class karyawan,
# yaitu class AnalisData
class AnalisData(Karyawan):
    def __init__(self, nama, usia, pendapatan):
    # melakukan pemanggilan konstruktur class Karyawan
        super().__init__(nama, usia, pendapatan)
    # menerapkan polymorphism dengan mendefinisikan kembali fungsi
    # lembur() pada class AnalisData
    def lembur(self):
        # pendapatan tambahan pada class AnalisData sebesar
        # 10 % dari pendapatannya.
        self.pendapatan_tambahan += int(self.pendapatan * 0.1)
# Buat objek karyawan yang bekerja sebagai AnalisData
aksara = AnalisData('Aksara', 25, 8500000)
aksara.lembur()
print(aksara.total_pendapatan())
```

```
9350000
```

Polymorphism pada Python - Part 2

In [ ]:

```
# Definisikan class Karyawan (sebagai base class)
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
# Buat class turunan (sebagai inherit class) dari class karyawan,
# yaitu class AnalisData
class AnalisData(Karyawan):
    def __init__(self, nama, usia, pendapatan):
        super().__init__ (nama, usia, pendapatan)
    # mendefinisikan kembali fungsi lembur() pada class AnalisData
    def lembur(self):
        # memanggil fungsi lembur pada class Karyawan
        super().lembur()
        # pendapatan tambahan pada class AnalisData sebesar
        # 5 % dari pendapatannya.
        self.pendapatan_tambahan += int(self.pendapatan * 0.05)
# Buat objek karyawan yang bekerja sebagai AnalisData
aksara = AnalisData('Aksara', 25, 8500000)
aksara.lembur()
print(aksara.total_pendapatan())
```

```
9175000
```

Tugas Praktek

In [ ]:

```
# Definisikan class Karyawan
class Karyawan:
    def __init__(self, nama, usia, pendapatan, insentif_lembur):
        self.nama = nama
        self.usia = usia
        self.pendapatan = pendapatan
        self.pendapatan_tambahan = 0
        self.insentif_lembur = insentif_lembur
    def lembur(self):
        self.pendapatan_tambahan += self.insentif_lembur
    def tambahan_proyek(self,jumlah_tambahan):
        self.pendapatan_tambahan += jumlah_tambahan
    def total_pendapatan(self):
        return self.pendapatan + self.pendapatan_tambahan
# Definisikan class TenagaLepas sebagai child class dari
# class Karyawan
class TenagaLepas(Karyawan):
    def __init__(self, nama, usia, pendapatan):
        super().__init__(nama, usia, pendapatan, 0)
    def tambahan_proyek(self, nilai_proyek):
        self.pendapatan_tambahan += nilai_proyek * 0.01
```

Tugas Praktek

In [ ]:

```
# Definisikan class Karyawan sebagai parent class
class Karyawan:
    def __init__(self, nama, usia, pendapatan, insentif_lembur):
        self.nama = nama
        self.usia = usia
        self.pendapatan = pendapatan
        self.pendapatan_tambahan = 0
        self.insentif_lembur = insentif_lembur
    def lembur(self):
```

```
        self.pendapatan_tambahan += self.insentif_lembur
    def tambahan_proyek(self,jumlah_tambahan):
        self.pendapatan_tambahan += jumlah_tambahan
    def total_pendapatan(self):
        return self.pendapatan + self.pendapatan_tambahan
# Definisikan class TenagaLepas sebagai child class dari
# class Karyawan
class TenagaLepas(Karyawan):
    def __init__(self, nama, usia, pendapatan):
        super().__init__(nama, usia, pendapatan, 0)
    def tambahan_proyek(self, nilai_proyek):
        self.pendapatan_tambahan += nilai_proyek * 0.01
# Definisikan class AnalisData sebagai child class dari
# class Karyawan
class AnalisData(Karyawan):
    pass
# Definisikan class IlmuwanData sebagai child class dari
# class Karyawan
class IlmuwanData(Karyawan):
    def tambahan_proyek(self, nilai_proyek):
        self.pendapatan_tambahan += 0.1 * nilai_proyek
# Definisikan class PembersihData sebagai child class dari
# class TenagaLepas
class PembersihData(TenagaLepas):
    pass
# Definisikan class DokumenterTeknis sebagai child class dari
# class TenagaLepas
class DokumenterTeknis(TenagaLepas):
    def tambahan_proyek(self, jumlah_tambahan):
        return
```

Overloading

Pada bahasa pemrograman lain yang mendukung paradigma OO seperti C# ataupun Java, polymorphism juga dapat diterapkan melalui sebuah fitur yang dikenal dengan istilah method overloading.

Metode overloading mengizinkan sebuah class untuk memiliki sekumpulan fungsi dengan nama yang sama dan parameter yang berbeda. Berkaitan dengan hal ini, Python tidak mengizinkan pendeklarasian fungsi (baik pada class ataupun tidak) dengan nama yang sama.

Untuk mengimplementasikan method overloading pada Python, aku dapat menggunakan sebuah teknik yang dikenal dengan function default parameters.

Tugas Praktek

In [ ]:

```
class Karyawan:
    nama_perusahaan = 'ABC'
    insentif_lembur = 250000
    # usia akan di-set nilainya menjadi 21 saat tidak
    # dispesifikasikan dan pendapatan akan di-set nilainya
    # menjadi 5000000 saat tidak dispesifikasikan
    def __init__(self, nama, usia=21, pendapatan=5000000):
        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
# Karyawan baru pertama yang bernama Budi
karyawan_baru1 = Karyawan('Budi')
print(karyawan_baru1.nama)
print(karyawan_baru1.usia)
print(karyawan_baru1.total_pendapatan())
```

```
# Karyawan baru ke-2 yang bernama Didi, umur 25
karyawan_baru2 = Karyawan('Didi', 25)
print(karyawan_baru2.nama)
print(karyawan_baru2.usia)
print(karyawan_baru2.total_pendapatan())
# Karyawan baru ke-3 yang bernama Hadi, pendapatan 8000000
karyawan_baru3 = Karyawan('Hadi', pendapatan=8000000)
print(karyawan_baru3.nama)
print(karyawan_baru3.usia)
print(karyawan_baru3.total_pendapatan())
```

```
Budi
21
5000000
Didi
25
5000000
Hadi
21
8000000
```

Studi Kasus dari Senja

In [ ]:

```
# Definisikan class Karyawan sebagai parent class
class Karyawan:
    def __init__(self, nama, usia, pendapatan, insentif_lembur):
        self.nama = nama
        self.usia = usia
        self.pendapatan = pendapatan
        self.pendapatan_tambahan = 0
        self.insentif_lembur = insentif_lembur
    def lembur(self):
        self.pendapatan_tambahan += self.insentif_lembur
    def tambahan_proyek(self,jumlah_tambahan):
        self.pendapatan_tambahan += jumlah_tambahan
    def total_pendapatan(self):
        return self.pendapatan + self.pendapatan_tambahan

# Definisikan class TenagaLepas sebagai child class dari class Karyawan
class TenagaLepas(Karyawan):
    def __init__(self, nama, usia, pendapatan):
        super().__init__(nama, usia, pendapatan, 0)
    def tambahan_proyek(self, nilai_proyek):
        self.pendapatan_tambahan += nilai_proyek * 0.01

# Definisikan class AnalisData sebagai child class dari class Karyawan
class AnalisData(Karyawan):
    def __init__(self, nama, usia = 21, pendapatan = 6500000,
                 insentif_lembur = 100000):
        super().__init__(nama, usia, pendapatan, insentif_lembur)

# Definisikan class IlmuwanData sebagai child class dari class Karyawan
class IlmuwanData(Karyawan):
    def __init__(self, nama, usia = 25, pendapatan = 12000000,
                 insentif_lembur = 150000):
        super().__init__(nama, usia, pendapatan, insentif_lembur)
    def tambahan_proyek(self, nilai_proyek):
        self.pendapatan_tambahan += 0.1 * nilai_proyek

# Definisikan class PembersihData sebagai child class dari class TenagaLepas
class PembersihData(TenagaLepas):
    def __init__(self, nama, usia, pendapatan = 4000000):
        super().__init__(nama, usia, pendapatan)

# Definisikan class DokumenterTeknis sebagai child class dari class TenagaLepas
class DokumenterTeknis(TenagaLepas):
    def __init__(self, nama, usia, pendapatan = 2500000):
        super().__init__(nama, usia, pendapatan)
        def tambahan_proyek(self, jumlah_tambahan):
```

```
            return

# Definisikan class Perusahaan
class Perusahaan:
    def __init__(self, nama, alamat, nomor_telepon):
        self.nama = nama
        self.alamat = alamat
        self.nomor_telepon = nomor_telepon
        self.list_karyawan = []
    def aktifkan_karyawan(self, karyawan):
        self.list_karyawan.append(karyawan)
    def nonaktifkan_karyawan(self, nama_karyawan):
        karyawan_nonaktif = None
        for karyawan in self.list_karyawan:
            if karyawan.nama == nama_karyawan:
                karyawan_nonaktif = karyawan
                break
        if karyawan_nonaktif is not None:
            self.list_karyawan.remove(karyawan_nonaktif)
    def total_pengeluaran(self):
        pengeluaran = 0
        for karyawan in self.list_karyawan:
            pengeluaran += karyawan.total_pendapatan()
        return pengeluaran
    def cari_karyawan(self, nama_karyawan):
        for karyawan in self.list_karyawan:
            if karyawan.nama == nama_karyawan:
                return karyawan
        return None

# Create object karyawan sesuai dengan tugasnya masing-masing
# seperti yang dinyatakan dalam tabel.
ani = PembersihData('Ani',25)
budi = DokumenterTeknis('Budi',18)
cici = IlmuwanData('Cici')
didi = IlmuwanData('Didi',32,20000000)
efi = AnalisData('Efi')
febi = AnalisData('Febi',28,12000000)

# Create object perusahaan
perusahaan = Perusahaan('ABC','Jl. Jendral Sudirman, Blok 11','(021)95812XX')

# Aktifkan setiap karyawan yang telah didefinisikan
perusahaan.aktifkan_karyawan(ani)
perusahaan.aktifkan_karyawan(budi)
perusahaan.aktifkan_karyawan(cici)
perusahaan.aktifkan_karyawan(didi)
perusahaan.aktifkan_karyawan(efi)
perusahaan.aktifkan_karyawan(febi)

# Cetak keseluruhan total pengeluaran perusahaan
print(perusahaan.total_pengeluaran())
```

```
57000000
```

Kesimpulan

1. **Konsep Object Oriented Programming (OOP) pada Python.**
2. **Teknik Class dan Objects, serta Class Behavior pada Python.**
3. **Teknik Encapsulation dan Inheritance dalam pemograman OOP dengan Python.**
4. **Teknik Polymorphism dan teknik Overloading dalam pemograman OOP pada Python.**
5. **Teknik membuat program OOP pada Python untuk kasus bisnis sederhana.**