

MODUL 1

1. Pengenalan Bahasa Pemrograman Python

Guido Van Rossum mengembangkan bahasa pemrograman Python pada akhir 1980-an saat bekerja di Centrum Wiskunde & Informatica Belanda. Python dapat digunakan dalam berbagai paradigma pemrograman, termasuk object-oriented, functional, dan structured. Saat ini, Python menjadi salah satu bahasa pemrograman yang paling populer karena sintaksnya yang mudah dipahami dan dilengkapi dengan library atau modul yang melimpah.

Python memiliki filosofi yang sangat memperhatikan kejelasan atau readability pada kode, sehingga untuk mengimplementasikan filosofi tersebut, Python tidak menggunakan kurung kurawal ({}), atau kata kunci (seperti start, begin, end) sebagai penanda blok kode.

2. Dasar Pemrograman Python

- Operator Aritmatika

Operator aritmatika adalah operator yang digunakan untuk melakukan operasi matematika, seperti penjumlahan, pengurangan, perkalian, pembagian, dan sebagainya.

```
1. a = 10
2. b = 15
3.
4. print(a + b)
5. print(a - b)
6. print(a / b)
7. print(a * b)
```

- Operator Perbandingan

Operator perbandingan adalah operator yang digunakan untuk membandingkan 2 buah nilai.

```
1. print(a != b)
2. print(a >= b)
3. print(a < b)
```

- Tipe data bentukan

Ada 4 tipe data bentukan yakni: List, Tuple, Set dan Dictionary

```
1. a = [1, 2, 3, 4, 5]
2. b = ("xyz", 1, 3.14)
3. c = { "firstName": "Joko", "lastName": "Widodo"}
4. d = { "apple", "banana", "cherry" }
```

- Percabangan

Dalam bahasa pemrograman Python, terdapat beberapa percabangan yaitu IF, IF-ELIF, dan IF-ELIF-ELSE.

- Perulangan For

Pada perulangan for biasa digunakan untuk iterasi pada urutan berupa list, tuple, atau string. Sintaks dasar pada perulangan for di python:

MODUL 2

1. Kelas

Dalam bahasa pemrograman Python, kelas atau class dapat dianggap sebagai cetakan atau blueprint dari objek atau instance yang akan dibuat. Dengan menggunakan kelas, kita dapat merancang objek dengan cara yang fleksibel.

- Atribut/Property

Dalam sebuah kelas, biasanya didefinisikan suatu variabel yang disebut sebagai atribut.

```
1. class Mahasiswa:
2.     jumlah_kaki = 2
3.     def __init__(self, nama, nim, kelas_siakad, jumlah_sks):
4.         self.nama = nama
5.         self.nim = nim #ini atribut objek
6.         self.kelas_siakad = kelas_siakad
7.         self.jumlah_sks = jumlah_sks
```

- Method

Method adalah suatu fungsi yang terdapat di dalam kelas. Method dapat diibaratkan sebagai sebuah aktivitas/proses yang dapat dilakukan oleh sebuah objek.

```
1.     def print_data_diri(self):
2.         print(f"""
3. ===== Data diri =====
4. Nama           : {self.nama}
5. Nim            : {self.nim}
6. kelas siakad   : {self.kelas_siakad}
7. jumlah sks     : {self.jumlah_sks}
8. ===== """)
```

2. Objek

Objek adalah sesuatu yang “mewakili” kelas. Objek disini berfungsi sebagai pengganti pemanggilan sebuah kelas.

```
1. data_1 = Mahasiswa("paijo", "121140333", "RD", "19")
2. data_1.print_data_diri()
```

3. Magic Method

Magic method adalah metode yang diawali dan diakhiri dengan double underscore (dunder). Method ini tidak dipanggil secara langsung.

4. Konstruktor

Konstruktor adalah method yang “pasti” dijalankan secara otomatis pada saat sebuah objek dibuat untuk mewakili kelas tersebut.

```
1. def __init__(self, nama, nim, kelas_siakad, jumlah_sks): #ini konstruktor
2.     self.nama = nama
3.     self.nim = nim
4.     self.kelas_siakad = kelas_siakad
5.     self.jumlah_sks = jumlah_sks
6.
```

5. Destruktor

Destruktor adalah fungsi yang dipanggil ketika user menghapus objek.

```
1. def __del__(self):  
2.     print("data mahasiswa {self.nama} berhasil dihapus")
```

MODUL 3

1. Abstraksi

Abstraksi merupakan sebuah konsep dalam pemrograman berorientasi objek yang memungkinkan pembuatan model objek dengan hanya menampilkan atribut yang penting dan menyembunyikan detail-detail yang tidak terlalu penting dari pengguna.

2. Enkapsulasi (Encapsulation)

Enkapsulasi merupakan sebuah teknik dalam pemrograman berorientasi objek yang digunakan untuk mengatur struktur sebuah kelas dengan cara menyembunyikan detail dan alur kerja kelas tersebut. Struktur kelas yang dimaksud meliputi property dan method. Dengan menggunakan konsep enkapsulasi, kita dapat membatasi akses terhadap property dan method tertentu dalam sebuah kelas agar tidak dapat diakses dari luar kelas tersebut.

Dalam Python, terdapat tiga jenis access modifier yang digunakan untuk membatasi akses terhadap property dan method dalam sebuah kelas, yaitu public access, protected access, dan private access.

```
1. class Mahasiswa:  
2.     jumlah_jari = 10  
3.     __jumlah_tangan = 2  
4.     __jumlah_kaki = 2  
5.  
6.     def angkat_tangan(self):  
7.         pass  
8.     def _angkat_kaki(self):  
9.         pass  
10.    def __sit_up(self):
```

11. `pass`

3. Setter dan Getter

setter adalah sebuah method yang digunakan untuk mengatur sebuah property yang ada di dalam suatu kelas/objek. Sedangkan getter adalah sebuah method yang digunakan untuk mengambil nilai dari suatu property.

➤ Tanpa decorator

```
1. class Mahasiswa:
2.     def __init__(self, usia):
3.         self.__usia = usia
4.
5.     #setter method
6.     def ubah_usia(self, baru):
7.         self.__usia = baru
8.
9.     #getter method
10.    def ambil_usia(self):
11.        return self.__usia
```

➤ Dengan Decorator

```
➤ class Mahasiswa:
➤     def __init__(self, usia):
➤         self.__usia = usia
➤
➤     @property
➤     def usia(self):
➤         return self.__usia
➤
➤     @usia.getter
➤     def usia(self, baru):
➤         self.__usia = baru
```

MODUL 4

1. Inheritance (Pewarisan)

Inheritance adalah salah satu konsep dasar dari Pemrograman Berbasis Objek (OOP). Pada inheritance, kita dapat menurunkan kelas dari kelas lain untuk hirarki kelas yang saling berbagi atribut dan metode.

```
1. class manusia:
2.     def __init__(self, nama, usia):
3.         self.nama = nama
4.         self.usia = usia
5.
6.     def keluaran(self):
7.         print(f"nama anda adalah {self.nama} dan usia anda adalah
           {self.usia}")
8.
9. class mahasiswa(manusia):
10.    pass
11.
12. orang_1 = mahasiswa("andre", 18)
13. orang_1.keluaran()
14.
```

keluaran

```
PS D:\Kuliah\S4\Praktikum PBO\M5> python -u "d:\Kuliah\S4\Praktikum PBO\M5\main.py"
nama anda adalah andre dan usia anda adalah 18
PS D:\Kuliah\S4\Praktikum PBO\M5>
```

- Inheritance Identik

Inheritance identik adalah suatu konsep pewarisan dalam pemrograman yang memungkinkan kelas anak memiliki constructor yang berbeda dengan kelas induk, tetapi tetap mempertahankan constructor kelas induk. Hal ini dilakukan dengan menggunakan kata kunci "super()" dalam constructor kelas anak untuk memanggil .

```

1. class manusia:
2.     def __init__(self, nama, usia):
3.         self.nama = nama
4.         self.usia = usia
5.
6.     def keluaran(self):
7.         print(f"nama anda adalah {self.nama} dan usia anda adalah
           {self.usia}")
8.
9. class mahasiswa(manusia):
10.    def __init__(self, nama, usia, pekerjaan):
11.        super().__init__(nama, usia)
12.        self.pekerjaan = pekerjaan
13.
14.    def output(self):
15.        print(f"nama anda adalah {self.nama} dan usia anda adalah
           {self.usia} pekerjaannya adalah {self.pekerjaan}")
16.
17. orang_1 = mahasiswa("andre", 18, "pelajar")
18. orang_1.keluaran()
19. orang_1.output()

```

Output

```

PS D:\Kuliah\S4\Praktikum PBO\M5> python -u "d:\Kuliah\S4\Praktikum PBO\M5\main.py"
nama anda adalah andre dan usia anda adalah 18
nama anda adalah andre dan usia anda adalah 18 pekerjaannya adalah pelajar
PS D:\Kuliah\S4\Praktikum PBO\M5> 

```

2. Polymorphism

Polymorphism adalah konsep dalam Pemrograman Berbasis Objek yang memungkinkan sebuah interface digunakan untuk menginstruksikan objek untuk melakukan aksi atau tindakan yang prinsipnya sama, namun dilakukan dengan proses yang berbeda. Dalam Python, polymorphism dapat diimplementasikan dengan kemampuan sebuah method untuk bekerja dengan berbagai jenis argumen.

```

1. class Kucing:
2.     def bersuara(self):
3.         print("Meong")
4.
5. class Anjing:
6.     def bersuara(self):
7.         print("Guk Guk")
8.
9. def buatHewanBersuara(hewan):
10.     hewan.bersuara()
11.
12. kucing = Kucing()
13. anjing = Anjing()
14.
15. buatHewanBersuara(kucing)
16. buatHewanBersuara(anjing)

```

output :

```

Meong
Guk Guk
PS D:\Kuliah\S4\Praktikum PBO\M5>

```

3. Override/Overriding

Override/Overriding Pada konsep OOP di python kita dapat menimpa suatu metode yang ada pada parent class dengan mendefinisikan kembali method dengan nama yang sama pada child class. Dengan begitu maka method yang ada parent class tidak berlaku dan yang akan dijalankan adalah method yang terdapat di child class.

```

1. class Hewan:
2.     def suara(self):
3.         print("Hewan membuat suara")
4.
5. class Anjing(Hewan):
6.     def suara(self):
7.         print("Guk Guk")
8.
9. anjing = Anjing()
10. anjing.suara()

```


output :

```
PS D:\Kuliah\S4\Praktikum PBO\M5> python  
Guk Guk  
PS D:\Kuliah\S4\Praktikum PBO\M5> []
```

Dalam contoh ini, kita membuat sebuah kelas Hewan dengan metode suara. Kemudian, kita membuat sebuah kelas anak Anjing yang merupakan turunan dari kelas Hewan dan juga memiliki metode suara.

Namun, dalam kelas Anjing, kita menulis ulang metode suara dan menggantinya dengan perilaku yang berbeda dari metode suara dalam kelas Hewan. Ketika kita membuat objek Anjing dan memanggil metode suara, maka metode suara dari kelas Anjing yang akan dipanggil, bukan metode suara dari kelas Hewan.

4. Overloading

Overloading adalah konsep di mana sebuah fungsi atau metode memiliki beberapa definisi yang berbeda dengan parameter masukan yang berbeda pula. Di Python, meskipun Python tidak secara eksplisit mendukung overloading, namun kita dapat mencapai overloading dengan cara berbeda.

Salah satu cara yang dapat digunakan untuk mencapai overloading di Python adalah dengan menggunakan argumen default. Berikut adalah contoh sederhana dari overloading di Python menggunakan argumen default:

```
1. class Hitung:  
2.     def jumlah(self, a, b, c=None):  
3.         if c is None:  
4.             return a + b  
5.         else:  
6.             return a + b + c  
7.  
8. hitung = Hitung()  
9. print(hitung.jumlah(5, 10))  
10. print(hitung.jumlah(5, 10, 15))
```

Output :

```
15
30
PS D:\Kuliah\S4\Praktikum PBO\M5> █
```

5. Multiple Inheritance

Multiple inheritance adalah konsep di mana sebuah kelas turunan dapat memiliki lebih dari satu kelas induk. Di Python, kita dapat mengimplementasikan multiple inheritance dengan menambahkan beberapa kelas induk dalam definisi kelas turunan.

```
1. class Pekerja:
2.     def bekerja(self):
3.         print("Pekerja bekerja keras")
4.
5. class Programmer:
6.     def coding(self):
7.         print("Programmer membuat kode")
8.
9. class ProgrammerPekerja(Programmer, Pekerja):
10.    pass
11.
12. programmer_pekerja = ProgrammerPekerja()
13. programmer_pekerja.bekerja()
14. programmer_pekerja.coding()
15.
```

Output :

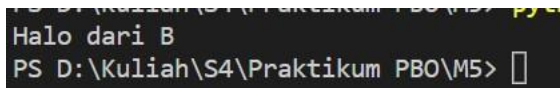
```
PS D:\Kuliah\S4\Praktikum PBO\M5> python -u "
Pekerja bekerja keras
Programmer membuat kode
PS D:\Kuliah\S4\Praktikum PBO\M5> █
```

6. Method Resolution Order di Python

Method Resolution Order (MRO) adalah urutan pencarian yang dilakukan oleh Python ketika mencari metode yang diwarisi oleh kelas turunan. Dalam Python, MRO dihitung menggunakan algoritma C3.

```
1. class A:
2.     def salam(self):
3.         print("Halo dari A")
4.
5. class B(A):
6.     def salam(self):
7.         print("Halo dari B")
8.
9. class C(A):
10.    def salam(self):
11.        print("Halo dari C")
12.
13. class D(B, C):
14.     pass
15.
16. d = D()
17. d.salam()
18.
```

Output :



```
PS D:\Kuliah\S4\Praktikum PBO\M5> python3
Halo dari B
PS D:\Kuliah\S4\Praktikum PBO\M5>
```

7. Dynamic Cast

Dynamic cast atau type conversion adalah proses mengubah nilai dari satu tipe data ke tipe data lainnya seperti dari string ke int atau sebaliknya. Ada 2 tipe konversi yaitu:

- Implisit

Python secara otomatis mengkonversikan tipe data ke tipe data lainnya tanpa ada campur tangan pengguna.

```

1. a = 2
2. b = 1.5
3. c = a + b
4. print(f"nilai a adalah {a} dengan tipe {type(a)}")
5. print(f"nilai b adalah {b} dengan tipe {type(b)}")
6. print(f"nilai c adalah {c} dengan tipe {type(c)}")

```

Output :

```

PS D:\Kuliah\S4\Praktikum PBO\M5> python a.py
nilai a adalah 2 dengan tipe <class 'int'>
nilai b adalah 1.5 dengan tipe <class 'float'>
nilai c adalah 3.5 dengan tipe <class 'float'>
PS D:\Kuliah\S4\Praktikum PBO\M5> 

```

- Eksplisit

Pengguna mengubah tipe data sebuah objek ke tipe data lainnya dengan fungsi yang sudah ada dalam python seperti int(), float(), dan str(). dapat berisiko terjadinya kehilangan data.

```

1. a = "375"
2. print(f"ini adalah bilangan {a} dengan type data {type(a)}")
3. a = int(a)
4. print(f"ini adalah bilangan {a} dengan type data {type(a)}")
5.

```

Output :

```

PS D:\Kuliah\S4\Praktikum PBO\M5> python a.py
ini adalah bilangan 375 dengan type data <class 'str'>
ini adalah bilangan 375 dengan type data <class 'int'>
PS D:\Kuliah\S4\Praktikum PBO\M5> 

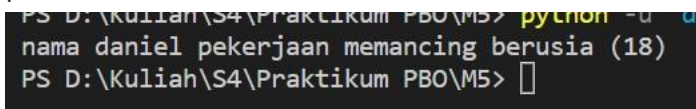
```

8. Casting

- **Downcasting:** Parent class mengakses atribut yang ada pada kelas bawah (child class)

```
1. class orang:
2.     def __init__(self, nama, pekerjaan, usia) -> None:
3.         self.nama = nama
4.         self.pekerjaan = pekerjaan
5.         self.usia = usia
6.
7.     def daftar(self):
8.         print(f"nama {self.nama} pekerjaan {self.pekerjaan}
berusia({self.usia})")
9.
10.manusia_1 = orang("daniel", "memancing", "18")
11.manusia_1.daftar()
12.
```

Output :



```
PS D:\Kuliah\S4\Praktikum PBO\M5> python -u d
nama daniel pekerjaan memancing berusia (18)
PS D:\Kuliah\S4\Praktikum PBO\M5> █
```

- **Upcasting:** Child class mengakses atribut yang ada pada kelas atas (parent class)

```
1. class manusia:
2.     pekerjaan = "engineer"
3.     def __init__(self, nama, pekerjaan):
4.         self.nama = nama
5.         self.pekerjaan = pekerjaan
6.
7.     def daftar(self):
8.         print(f"nama {self.nama} {self.nama_belakang} pekerjaan
{self.pekerjaan} berusia ({self.usia})")
9.
10.class orang(manusia):
11.     def __init__(self, nama, pekerjaan, usia):
12.         super().__init__(nama, pekerjaan)
13.         self.usia = usia
14.     def daftar(self):
```

```

15.         print(f"nama {self.nama} pekerjaan {super().pekerjaan} berusia
              ({self.usia})")
16.
17.manusia_1 = orang("daniel", "memancing", 18)
18.manusia_1.daftar()

```

Output :

```

PS D:\Kuliah\S4\Praktikum PBO\M5> python -u "d:\Kuliah\S4\Praktikum PBO\M5\main.py"
nama daniel pekerjaan enginer berusia (18)
PS D:\Kuliah\S4\Praktikum PBO\M5> 

```

- **type casting** adalah proses mengubah tipe atau sifat dari sebuah objek atau variabel agar memiliki perilaku tertentu yang tidak dimiliki secara default oleh kelas tersebut. Dalam bahasa pemrograman Python, semua variabel atau instance pada dasarnya merupakan objek atau kelas, dan perilakunya dapat dimanipulasi menggunakan metode-metode khusus yang disebut magic method.

Contoh 1 (kelas berperilaku seperti string dan integer)

```

1. class orang:
2.     def __init__(self, nama, pekerjaan, usia):
3.         self.nama = nama
4.         self.pekerjaan = pekerjaan
5.         self.usia = usia
6.     def __str__(self):
7.         return f"nama {self.nama} pekerjaan {self.pekerjaan} berusia
              ({self.usia})"
8.     def __int__(self):
9.         return self.usia
10.
11.manusia_1 = orang("daniel", "memancing", 18)
12.print(manusia_1)
13.print(int(manusia_1) == 18)

```

Output :

```

PS D:\Kuliah\S4\Praktikum PBO\M5> python -u "d:\Kuliah\S4\Praktikum PBO\M5\main.py"
nama daniel pekerjaan memancing berusia (18)
True
PS D:\Kuliah\S4\Praktikum PBO\M5> 

```