RESUME PRAKTIKUM
PEMROGRAMAN BERORIENTASI OBJEK (RB)
PERTEMUAN 1 - 4

Oleh:

Akhmad Fadilla Akbar (121140130)

Program Studi Teknik Informatika

Institut Teknologi Sumatera

2022

Daftar Isi

Pertemuan 1

Pengenalan dan Dasar Pemrograman Python I

A. Pengenalan Bahasa Phyton

Python adalah bahasa pemrograman tingkat tinggi yang pertama kali dirilis pada tahun 1991 oleh Guido van Rossum. Python didesain dengan sintaks yang mudah dipahami dan lebih mudah untuk dibaca dan ditulis oleh manusia. Python juga dikembangkan dengan fokus pada produktivitas dan kemampuan untuk menyelesaikan tugas dalam waktu yang lebih singkat.

Beberapa karakteristik penting dari Python adalah:

- Bahasa pemrograman yang mendukung pemrograman berorientasi objek, fungsional, dan prosedural
- Mudah dibaca, ditulis, dan dipelajari karena sintaks yang sederhana dan mudah dimengerti
- Sangat fleksibel dan dapat digunakan untuk berbagai keperluan, seperti pengembangan web, ilmu data, kecerdasan buatan, game, dan banyak lagi
- Memiliki komunitas pengembang yang besar dan aktif yang menyediakan banyak modul dan paket yang berguna
- Gratis dan open source, artinya dapat digunakan dan didistribusikan secara bebas
- Python dapat dijalankan di berbagai platform, termasuk Windows, MacOS, dan Linux. Selain itu, Python juga dapat dijalankan di lingkungan pengembangan terpadu (IDE) seperti PyCharm, Visual Studio Code, dan IDLE.

B. Dasar Pemgroggraman Python

a) Sintaks Dasar

Sintaks dasar Python sangat mudah dipahami dan diingat. Beberapa contoh sintaks dasar Python adalah:

- Menampilkan teks ke layar: print("Hello, World!")
- Variabel: nama = "John"
- If statement: if x > 10: print("x lebih besar dari 10")
- Looping: for i in range(10): print(i)

b) Modul

Python memiliki banyak modul dan paket yang dapat membantu pengembangan program. Beberapa modul dan paket populer di antaranya adalah:

- NumPy: modul untuk pemrosesan data numerik
- Pandas: modul untuk analisis data dan pemrosesan data tabular
- Matplotlib: modul untuk visualisasi data
- TensorFlow: paket untuk pembelajaran mesin
- Django: kerangka kerja web untuk pengembangan aplikasi web

Dalam keseluruhan, Python adalah bahasa pemrograman yang populer, mudah dipelajari, dan dapat digunakan untuk berbagai keperluan. Python juga memiliki komunitas pengembang yang aktif dan menyediakan banyak modul dan paket yang berguna.

```python
# user memasukkan jumlah bilangan
n = int(input("Masukkan jumlah bilangan: "))

# Inisialisasi variabel untuk menyimpan total bilangan
total = 0

# user memasukkan bilangan sebanyak n kali
for i in range(n):
    bil = int(input("Masukkan bilangan ke-{}: ".format(i+1)))
    total += bil

# menghitung rata-rata dari bilangan yang dimasukkan
rata = total / n

# menampilkan hasil rata-rata
print("Rata-rata dari bilangan yang dimasukkan adalah:", rata)
```

Program meminta user untuk memasukkan jumlah bilangan yang akan dihitung rata-ratanya. Program kemudian melakukan loop sebanyak n kali untuk meminta user memasukkan bilangan yang akan dihitung rata-ratanya. Setiap bilangan dimasukkan, maka variabel total akan bertambah dengan nilai bilangan tersebut. Setelah semua bilangan dimasukkan, program menghitung rata-rata dari bilangan tersebut dengan membagi total dengan n. Hasil rata-rata kemudian ditampilkan di layar.

Pertemuan 2

Objek dan Kelas Dalam Python (Konstruktor, Setter, dan Getter)

A. Kelas

Kelas adalah sebuah blueprint atau cetakan yang digunakan untuk membuat objek. Dalam bahasa pemrograman Python, semua hal dianggap sebagai objek, termasuk tipe data seperti angka, string, dan list. Dengan menggunakan kelas kita dapat membuat sebuah objek dengan leluasa. Kelas memiliki dan mendefinisikan atribut/property dan metode untuk objek yang akan dibuat.

```
class Mobil:
    def __init__(self, merek, warna):
        self.merek = merek
        self.warna = warna

    def info_mobil(self):
        print("Merek:", self.merek)
        print("Warna:", self.warna)

mobil1 = Mobil("Toyota", "Putih")
mobil1.info_mobil()

mobil2 = Mobil("Honda", "Hitam")
mobil2.info_mobil()
```

Dalam contoh di atas, kita membuat sebuah kelas bernama "Mobil". Kita menggunakan metode khusus bernama "init" untuk menginisialisasi atribut "merek" dan "warna". Kemudian, kita membuat metode "info_mobil" untuk mencetak informasi mobil.

Setelah itu, kita membuat dua objek baru, "mobil1" dan "mobil2", yang dibuat berdasarkan kelas "Mobil". Kita memberikan nilai "Toyota" dan "Putih" untuk objek "mobil1" dan nilai "Honda" dan "Hitam" untuk objek "mobil2". Akhirnya, kita memanggil metode "info_mobil" pada setiap objek untuk mencetak informasi mobil yang sesuai.

Dalam Python, objek dan kelas sangat fleksibel dan dapat disesuaikan dengan kebutuhan pengguna. Konsep ini sangat penting dalam membangun program yang kompleks dan fleksibel.

B. Objek

Objek adalah entitas dalam program yang memiliki properti dan perilaku tertentu. Semua tipe data dalam Python, seperti angka, string, daftar, tupel, dll., dianggap sebagai objek. Faktanya, kelas itu sendiri juga dianggap sebagai objek di Python.

C. Magic Method

Magic method pada Python adalah metode khusus yang memiliki nama yang diawali dan diakhiri dengan dua garis bawah (__). Metode ini juga dikenal sebagai metode khusus atau metode ajaib. Magic method digunakan untuk mengatur perilaku kelas pada Python.

Magic method pada Python digunakan untuk mengimplementasikan berbagai fungsi bawaan pada kelas, seperti pengurangan, penambahan, perbandingan, dan sebagainya. Dalam Python, ada banyak magic method yang tersedia, dan mereka dapat diimplementasikan untuk menyesuaikan perilaku objek dengan kebutuhan kita.

D. Destruktor

Destructor pada Python adalah sebuah method khusus dalam sebuah kelas yang digunakan untuk membersihkan objek ketika objek tersebut tidak digunakan atau dihapus dari memori. Method ini dijalankan secara otomatis ketika objek dihapus dari memori, atau ketika objek sudah tidak digunakan lagi. Destructor pada Python memiliki nama "del".

Destructor biasanya digunakan untuk menghapus sumber daya yang digunakan oleh objek, seperti file, soket, koneksi database, dan sebagainya. Dalam bahasa pemrograman lain, destructor juga dikenal sebagai finalizer.

```python
class Mobil:
    def __init__(self, merk, tahun):
        self.merk = merk
        self.tahun = tahun
        print(f"{self.merk} ({self.tahun}) telah dibuat.")

    def __del__(self):
        print(f"{self.merk} ({self.tahun}) telah dihapus dari memori.")

mobil1 = Mobil("Toyota", 2015)
mobil2 = Mobil("Honda", 2018)

del mobil1
```

Pada contoh di atas, kita membuat sebuah kelas "Mobil" yang memiliki dua atribut, "merk" dan "tahun". Ketika objek baru dibuat, constructor (init) akan dijalankan dan mencetak pesan "Toyota (2015) telah dibuat." atau "Honda (2018) telah dibuat." di layar. Ketika objek "mobil1" dihapus dengan menggunakan perintah "del", destructor (del) akan dijalankan dan mencetak pesan "Toyota (2015) telah dihapus dari memori." di layar. Dengan menggunakan destructor, kita dapat memastikan bahwa sumber daya yang digunakan oleh objek akan dihapus dari memori ketika objek tersebut sudah tidak digunakan lagi, sehingga menghemat penggunaan memori dan mencegah terjadinya kebocoran memori (memory leak).

E. Setter and Getter

Setter dan getter adalah metode atau fungsi yang digunakan untuk mengakses dan memodifikasi nilai atribut dalam sebuah kelas di Python. Getter merupakan method yang digunakan untuk mengambil nilai dari sebuah atribut. Sedangkan setter merupakan method yang digunakan untuk memperbarui nilai dari sebuah atribut. Untuk mengimplementasikan getter dan setter dalam Python, kita dapat menggunakan property decorators. Property decorators adalah decorator yang digunakan untuk mendefinisikan getter, setter, dan deleter dalam sebuah kelas.

F. Decorator

Decorator pada Python adalah fungsi yang digunakan untuk mengubah fungsi atau metode yang sudah ada, dengan menambahkan fungsionalitas tambahan ke dalamnya. Decorator ini diletakkan di atas fungsi atau metode yang ingin diubah. Pada Python, decorator didefinisikan menggunakan simbol "@" dan diikuti dengan nama decorator. Decorator ini diletakkan di atas definisi fungsi atau metode yang ingin diubah. Saat fungsi atau metode tersebut dipanggil, decorator akan dijalankan terlebih dahulu sebelum fungsi atau metode yang sesungguhnya dijalankan.

Pertemuan 3

Abstraksi dan Enskapsulasi (Visibilitas Fungsi dan Variabel, Relasi Antar Kelas)

A. Abstraksi

Abstraksi pada Python merujuk pada teknik pengkodean yang memungkinkan kita untuk memusatkan perhatian pada informasi penting dan mengabaikan detail yang tidak penting. Abstraksi adalah salah satu konsep dasar dalam pemrograman berorientasi objek (OOP) dan dapat membantu dalam membangun kode yang bersih, terstruktur, dan mudah dimengerti.

Dalam OOP, abstraksi dapat dicapai melalui konsep kelas dan objek. Kita dapat membuat kelas yang merepresentasikan objek nyata, dan mengabaikan detail implementasi objek tersebut. Kita hanya perlu memperhatikan properti atau atribut objek yang relevan dan metode atau fungsi yang dapat dijalankan pada objek tersebut.

Selain itu dalam Python, abstraksi juga dapat dicapai melalui konsep modul. Modul adalah file yang berisi definisi kelas, fungsi, dan konstanta yang dapat digunakan di file lain. Dengan membuat modul, kita dapat mengurangi kompleksitas kode dan memudahkan pengembangan aplikasi yang kompleks.

B. Enskapsulasi

Enkapsulasi pada Python adalah teknik pengkodean yang memungkinkan kita untuk menyembunyikan detail implementasi suatu objek dari pengguna luar, dan hanya mengekspos fungsionalitas objek yang relevan melalui metode atau atribut yang ditentukan. Tujuan dari enkapsulasi adalah untuk meningkatkan keamanan dan menjaga kekonsistenan data dalam sebuah program.

Dalam Python, enkapsulasi dapat dicapai melalui penggunaan aksesori atau metode setter dan getter pada atribut kelas. Metode setter digunakan untuk mengatur nilai dari sebuah atribut, sedangkan metode getter digunakan untuk mengambil nilai dari sebuah atribut. Dengan menggunakan metode setter dan getter, kita dapat memastikan bahwa nilai atribut yang diakses dan diubah oleh pengguna selalu valid dan sesuai dengan aturan yang ditetapkan.

A. Public Access Modifier

Pada umumnya ketika kita mendeklarasikan suatu variabel atau method, maka itulah public access modifier. Setiap class, variable dan method yang dibuat secara default merupakan public.

```python
class Mobil:
    def __init__(self, merk, tahun):
        self.merk = merk
        self.tahun = tahun
        print(f"{self.merk} ({self.tahun}) telah dibuat.")

    def __del__(self):
        print(f"{self.merk} ({self.tahun}) telah dihapus dari memori.")
```

B. Protected Access Modifier

Jika suatu variabel dan method dideklarasikan secara protected, maka variable dan method tersebut hanya dapat diakses oleh kelas turunan darinya. Cara mendeklarasikannya dengan menambahkan 1 underscore (_) sebelum variable atau method.

```
class Mobil:
    def __init__(self, merk, tahun):
        self._merk = merk
        self._tahun = tahun
        print(f"{self._merk} ({self._tahun}) telah dibuat.")

    def __del__(self):
        print(f"{self._merk} ({self._tahun}) telah dihapus dari memori.")
```

C. Private Access Modifier

Jika suatu variabel dan method dideklarasikan secara private, maka variable dan method tersebut hanya dapat diakses di dalam kelas itu sendiri, private access merupakan yang paling aman. Dalam mendeklarasikannya, hanya perlu menambahkan double underscore ( ) sebelum nama variable dan methodnya.

```
class Mobil:
    def __init__(self, merk, tahun):
        self.__merk = merk
        self.__tahun = tahun
        print(f"{self.__merk} ({self.__tahun}) telah dibuat.")

    def __del__(self):
        print(f"{self.__merk} ({self.__tahun}) telah dihapus dari memori.")
```

D. Setter and 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. Dikarenakan property dengan access modifier private hanya dapat diakses dari dalam kelas tersebut, dengan metode inilah kita dapat mengaksesnya dari luar kelas. Contoh program ada di halaman 12.

Pertemuan 4

Pewarisan dan Polimorfisme (Overloading, Overriding, dynamic Cast)

A. Inheritance (Pewarisan)

Inheritance atau pewarisan merupakan salah satu konsep dasar dalam pemrograman berorientasi objek (PBO) di Python, dimana sebuah class dapat memperoleh sifat dan perilaku dari class lain. Class yang mewarisi sifat dan perilaku disebut sebagai subclass, sedangkan class yang memberikan sifat dan perilaku disebut sebagai superclass.

Di Python, inheritance dapat diimplementasikan dengan cara mendefinisikan sebuah class baru dengan menyebutkan nama class yang ingin diwarisi sebagai argumen dari class yang baru. Proses ini disebut sebagai subclassing. Selain itu, subclass juga dapat menambahkan sifat atau perilaku yang unik dengan mendefinisikan atribut atau method baru, atau meng-override atribut atau method yang diwarisi dari superclass.

```python
# parent
class Kendaraan:
    pass

# child
class Mobil(Kendaraan):
    pass
```

```python
# superclass
class Kendaraan:
    def __init__(self, merk, tahun):
        self.merk = merk
        self.tahun = tahun

    def info_kendaraan(self):
        print(f"Kendaraan {self.merk} tahun {self.tahun}.")

# subclass
class Mobil(Kendaraan):
    def __init__(self, merk, tahun, jenis_bahan_bakar):
        super().__init__(merk, tahun)
        self.jenis_bahan_bakar = jenis_bahan_bakar

    def info_kendaraan(self):
        print(f"Mobil {self.merk} tahun {self.tahun}, bahan bakar {self.jenis_bahan_bakar}.")

# membuat objek
mobil1 = Mobil("Toyota", 2015, "Bensin")
mobil1.info_kendaraan()
```

B. Plymorphism

Polymorphism merupakan konsep dalam pemrograman berorientasi objek (OOP) yang memungkinkan suatu objek untuk memiliki banyak bentuk atau perilaku. Dalam Python, polymorphism dapat diimplementasikan melalui konsep pewarisan (inheritance) dan penggunaan fungsi atau method yang memiliki nama yang sama, tetapi berbeda dalam implementasinya pada setiap class.

```python
# superclass
class Hewan:
    def __init__(self, nama):
        self.nama = nama
    def bersuara(self):
        pass
# subclass
class Kucing(Hewan):
    def bersuara(self):
        print(f"{self.nama} : Meow")
# subclass
class Anjing(Hewan):
    def bersuara(self):
        print(f"{self.nama} : Guk Guk")
# fungsi yang menggunakan objek hewan
def dengarkan_suara(hewan):
    hewan.bersuara()
# membuat objek
kucing1 = Kucing("Kitty")
anjing1 = Anjing("Snoopy")

# memanggil fungsi
dengarkan_suara(kucing1)
dengarkan_suara(anjing1)
```

C. 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.

```python
# superclass
class Hewan:
    def __init__(self, nama):
        self.nama = nama

    def bersuara(self):
        print("Hewan bersuara")

# subclass
class Kucing(Hewan):
    def bersuara(self):
        print(f"{self.nama} : Meow")

# objek kucing
kucing1 = Kucing("Kitty")

# memanggil method bersuara
kucing1.bersuara()
```

D. Overloading

Metode overloading mengizinkan sebuah class untuk memiliki sekumpulan fungsi dengan nama yang sama dan argumen yang berbeda. Akan tetapi, Python tidak mengizinkan pendeklarasian fungsi (baik pada class ataupun tidak) dengan nama yang sama. Hal itu menyebabkan implementasi overloading pada python menjadi “tricky”. Secara umum overloading memiliki beberapa signature, yaitu jumlah argumen, tipe argumen, tipe keluaran dan urutan argumen. Akan tetapi, seperti yang telah dijelaskan python tidak menangani overloading secara khusus. Karena python adalah Bahasa pemrograman yang bersifat duck typing (dynamic typing), overloading secara tidak langsung dapat diterapkan.

E. Multiple Inheritance

Multiple Inheritance adalah fitur di Python yang memungkinkan sebuah kelas untuk mengambil sifat dari lebih dari satu kelas induk. Artinya, kita dapat mewarisi atribut dan metode dari beberapa kelas induk ke dalam satu kelas turunan. Dalam multiple inheritance, kelas turunan memperoleh sifat dan metode dari dua atau lebih kelas induk. Ini dapat dilakukan dengan membuat kelas turunan baru dari beberapa kelas induk. Dalam kasus ini, kelas turunan memiliki akses ke seluruh sifat dan metode yang diwarisi dari semua kelas induk.

```
class A:
    def method_A(self):
        print("Ini adalah method dari kelas A")

class B:
    def method_B(self):
        print("Ini adalah method dari kelas B")

class C(A, B):
    def method_C(self):
        print("Ini adalah method dari kelas C")

obj = C()
obj.method_A()
obj.method_B()
obj.method_C()
```

F. Method Resolution Order di Python

Method Resolution Order (MRO) adalah urutan yang digunakan oleh Python untuk mencari metode dalam kelas induk saat metode yang sama didefinisikan dalam kelas turunan. Dalam kasus multiple inheritance, MRO menentukan urutan pencarian metode dari kelas-kelas induk.

G. 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:

a) Implisit

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

b) 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.

H. Casting

a) Downcasting
   Parent class mengakses atribut yang ada pada kelas bawah (child class).
b) Upcasting
   Child class mengakses atribut yang ada pada kelas atas (parent class).
c) Type casting
   Konversi tipe kelas agar memiliki sifat/perilaku tertentu yang secara default tidak dimiliki kelas tersebut. Karena Python merupakan bahasa pemrograman berorientasi objek, maka semua variabel atau instansi di Python pada dasarnya merupakan objek (kelas) yang sifat/perilakunya dapat dimanipulasi jika dibutuhkan (umumnya melalui magic method).