TUGAS PRAKTIKUM
PEMROGRAMAN BERORIENTASI OBJEK RB
RESUME

Oleh :

Salma Zurida (121140064)

Program Studi Teknik Informatika

Institut Teknologi Sumatera

2022

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 sintaks nya 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 digunakkan untuk iterasi pada urutan berupa list, tuple, atau string. Sintaks dasar pada perulangan for di python:

MODUL 2

1. Kelas

   Kelas (class) adalah salah satu konsep dasar dalam pemrograman berorientasi objek yang memungkinkan pengguna untuk membuat tipe data baru dengan atribut dan metode tertentu. Dalam python, kelas dibuat menggunakan kata kunci “class”, diikuti dengan nama kelas dan blok kode yang berisi definisi atribut dan metode

```python
class Mobil:
    def __init__(self, dibuat, model, tahun):
        self.dibuat = dibuat
        self.model = model
        self.tahun = tahun

    def start(self):
        print("Mobil berjalan.")

    def stop(self):
        print("Mobil berhenti.")
```

   Pada contoh berikut, kita membuat sebuah kelas bernama Mobil dengan atribut dibuat, model, dan tahun. Atribut tersebut diinisialisasi saat objek dibuat dengan menggunakan metode __init__. Selain atribut, kita juga dapat menambahkan metode ke dalam kelas. Pada contoh di atas, kita menambahkan dua metode, yaitu start() dan stop(). Metode tersebut digunakan untuk memberi instruksi pada objek untuk memulai dan menghentikan mesin mobil. Setelah kelas dibuat, membuat objek dari kelas tersebut menggunakan sintaks berikut :

```python
mobil1 = Mobil("Toyota", "Corlla", 2021)
```

   Dengan membuat objek dari kelas, kita dapat mengakses atribut metode yang telah didefinisikan dalam kelas tersebut

```
print(mobil1.dibuat)
print(mobil1.model)
print(mobil1.tahun)

mobil1.start()
mobil1.stop()
```

Hasil Output :

```
Toyota
Corolla
2021
Mobil berjalan.
Mobil berhenti.
```

2. Atribut

Atribut pada program Python adalah variabel yang didefinisikan di dalam sebuah kelas dan dapat diakses dari objek yang dibuat dari kelas tersebut. Atribut menyimpan data atau informasi mengenai objek, seperti nama, alamat, umur, dan sebagainya. Dalam Python, terdapat dua jenis atribut yang dapat didefinisikan di dalam sebuah kelas, yaitu:

a. Atribut Instance

Atribut instance adalah atribut yang nilainya unik pada setiap objek yang dibuat dari sebuah kelas. Atribut ini didefinisikan di dalam metode init() dan dapat diakses melalui objek yang dibuat.

```
class Mahasiswa:
    def __init__(self, nama, nim):
        self.nama = nama
        self.nim = nim

mahasiswa1 = Mahasiswa("Agus", "123456")

print(mahasiswa1.nama)
print(mahasiswa1.nim)
```

Hasil Output :

```
Agus
123456
```

b. Atribut Class

Atribut class adalah atribut yang nilainya sama untuk semua objek yang dibuat dari sebuah kelas. Atribut ini didefinisikan di luar metode init() dan dapat diakses melalui nama kelas.

```python
class Mobil:
    jumlah_mobil = 0

    def __init__(self, merk, warna):
        self.merk = merk
        self.warna = warna
        Mobil.jumlah_mobil += 1

mobil1 = Mobil("Toyota", "Hitam")
mobil2 = Mobil("Honda", "Putih")
print(mobil1.jumlah_mobil)
print(mobil2.jumlah_mobil)
```

Hasil Output :

```
2
2
```

Atribut class dapat diakses melalui objek atau nama kelas, tetapi jika diubah nilainya melalui objek, maka hanya objek tersebut yang nilainya berubah, sementara objek lain dan kelas tetap memiliki nilai yang sama.

3. Method

Method pada program Python adalah fungsi yang didefinisikan di dalam sebuah kelas dan beroperasi pada objek yang dibuat dari kelas tersebut. Method digunakan untuk melakukan operasi atau manipulasi data pada objek, dan dapat mengakses atribut dan method lain yang dimiliki oleh objek atau kelas tersebut. Dalam Python, terdapat tiga jenis method yang dapat didefinisikan di dalam sebuah kelas, yaitu:

a. Method Instance

Method instance adalah method yang hanya dapat diakses melalui objek yang dibuat dari kelas. Method ini biasanya digunakan untuk melakukan operasi pada atribut objek tersebut.

```python
class Kalkulator:
    def __init__(self, bilangan1, bilangan2):
        self.bilangan1 = bilangan1
        self.bilangan2 = bilangan2

    def tambah(self):
        return self.bilangan1 + self.bilangan2

    def kurang(self):
        return self.bilangan1 - self.bilangan2

kalkulator1 = Kalkulator(10, 5)
print(kalkulator1.tambah())
print(kalkulator1.kurang())
```

Hasil Output :

```
15
5
```

b. Method Class

Method class adalah method yang dapat diakses melalui nama kelas dan tidak membutuhkan objek untuk dioperasikan. Method ini biasanya digunakan untuk melakukan operasi pada atribut kelas atau membuat objek baru.

```python
class Mahasiswa:
    jumlah_mahasiswa = 0

    def __init__(self, nama, nim):
        self.nama = nama
        self.nim = nim
        Mahasiswa.jumlah_mahasiswa += 1

    @classmethod
    def jumlah(cls):
        return cls.jumlah_mahasiswa

mahasiswa1 = Mahasiswa("Agus", "123456")
mahasiswa2 = Mahasiswa("Budi", "654321")
print(Mahasiswa.jumlah())
```

Hasil Output :

```
2
```

c. Method Static

Method static adalah method yang tidak membutuhkan objek atau kelas untuk dioperasikan. Method ini biasanya digunakan untuk melakukan operasi yang tidak berkaitan denfan objek atau kelas.

```
class Utility:
    @staticmethod
    def kali(angka1, angka2):
        return angka1 * angka2

print(Utility.kali(5, 3))
```

Hasil Output :

```
15
```

Dalam penggunaan method pada program Python, sebaiknya kita mengikuti prinsip enkapsulasi atau penyembunyian informasi dengan membatasi akses langsung ke atribut objek melalui method-metode yang telah didefinisikan. Hal ini dapat membantu menjaga keamanan dan integritas data dari objek. Selain itu, kita juga dapat menggunakan decorator seperti @property dan @setter untuk membuat method yang terlihat seperti atribut, tetapi sebenarnya melakukan operasi tertentu saat atribut tersebut diakses atau diubah.

4. Objek

Objek dalam program Python adalah instansi dari sebuah kelas. Objek dianggap sebagai unit dasar dari program, dan kelas sebagai blueprint atau template untuk membuat objek-objek tersebut. Setiap objek memiliki atribut dan method yang ditentukan oleh kelas yang digunakan untuk membuat objek tersebut. Atribut adalah data yang terkait dengan objek, sedangkan method adalah fungsi yang terkait dengan objek dan digunakan untuk melakukan operasi atau manipulasi data pada objek tersebut. Untuk membuat objek di program Python, kita perlu menggunakan konstruktor atau method init(). Konstruktor digunakan untuk menginisialisasi nilai awal pada atribut objek ketika objek dibuat dari kelas.

```python
class Mahasiswa:
    def __init__(self, nama, nim, jurusan):
        self.nama = nama
        self.nim = nim
        self.jurusan = jurusan

    def tampilkan(self):
        print("Nama:", self.nama)
        print("NIM:", self.nim)
        print("Jurusan:", self.jurusan)

mahasiswa1 = Mahasiswa("Agus", "123456", "Teknik Informatika")
mahasiswa2 = Mahasiswa("Budi", "654321", "Sistem Informasi")
```

Pada contoh di atas, kita membuat dua objek mahasiswa1 dan mahasiswa2 dari kelas Mahasiswa dengan atribut nama, nim, dan jurusan yang berbeda-beda. Objek-objek tersebut dapat diakses melalui nama objek dan dapat memanggil method tampilkan() untuk menampilkan informasi yang terkait dengan objek tersebut.

Dalam penggunaan objek pada program Python, kita dapat menggunakan prinsip enkapsulasi atau penyembunyian informasi dengan membatasi akses langsung ke atribut objek. Hal ini dapat membantu menjaga keamanan dan integritas data dari objek. Kita juga dapat menggunakan decorator seperti @property dan @setter untuk membuat method yang terlihat seperti atribut, tetapi sebenarnya melakukan operasi tertentu saat atribut tersebut diakses atau diubah.

5. Magic Method

Magic method di program Python adalah method khusus yang dimulai dan diakhiri dengan dua garis bawah (underscores) atau disebut juga dengan dunder (double underscore). Method ini sering juga disebut sebagai method khusus atau method ajaib karena digunakan untuk mengubah perilaku kelas atau objek pada saat tertentu. Beberapa contoh magic method yang sering digunakan di Python:

- init(): method ini digunakan untuk menginisialisasi objek ketika kelas dibuat.

- str(): method ini digunakan untuk mengembalikan representasi string dari objek.
- len(): method ini digunakan untuk mengembalikan panjang (length) dari objek.
- eq(): method ini digunakan untuk menentukan apakah dua objek sama atau tidak.
- add(): method ini digunakan untuk menambahkan dua objek.

6. Konstruktor

Konstruktor adalah method khusus pada kelas di program Python yang digunakan untuk menginisialisasi nilai awal pada atribut objek ketika objek tersebut dibuat dari kelas. Konstruktor biasanya memiliki nama init() dan dijalankan secara otomatis pada saat pembuatan objek. Pada saat objek dibuat, nilai-nilai atribut yang telah ditentukan di dalam konstruktor akan diassign ke objek tersebut. Konstruktor dapat menerima parameter sesuai dengan kebutuhan dalam menginisialisasi objek.

```python
class Mahasiswa:
    def __init__(self, nama, nim, jurusan):
        self.nama = nama
        self.nim = nim
        self.jurusan = jurusan

mahasiswa1 = Mahasiswa("Agus", "123456", "Teknik Informatika")
```

Pada contoh di atas, kita menggunakan konstruktor untuk membuat objek mahasiswa1 dari kelas Mahasiswa dengan atribut nama, nim, dan jurusan yang telah ditentukan saat pembuatan objek. Dengan menggunakan konstruktor, kita dapat dengan mudah menginisialisasi nilai awal pada atribut objek dengan cepat dan efisien.

Selain itu, konstruktor juga dapat digunakan untuk melakukan validasi pada nilai-nilai yang diterima sebagai parameter. Jika nilai-nilai yang diterima tidak sesuai dengan kebutuhan, konstruktor dapat memunculkan pesan error atau melakukan operasi tertentu untuk mengatasi masalah tersebut.

7. Destruktor

Destruktor adalah method khusus pada kelas di program Python yang digunakan untuk membersihkan memori atau sumber daya lainnya yang digunakan oleh objek ketika objek tersebut dihapus atau tidak digunakan lagi. Destruktor biasanya memiliki nama del() dan dijalankan secara otomatis ketika objek dihapus dari memori.

Destruktor dapat digunakan untuk menghapus sumber daya yang dialokasikan secara dinamis oleh objek, seperti file, koneksi database, atau objek yang kompleks lainnya. Destruktor juga dapat digunakan untuk melakukan operasi tertentu sebelum objek dihapus dari memori, seperti menyimpan data ke database atau melakukan proses lain yang diperlukan.

Namun, perlu diingat bahwa penggunaan destruktor sebaiknya dibatasi dan hanya digunakan jika memang diperlukan. Python memiliki manajemen memori yang efektif dan melakukan penghapusan objek secara otomatis ketika objek tersebut tidak lagi digunakan oleh program.

```python
class Mahasiswa:
    def __init__(self, nama, nim, jurusan):
        self.nama = nama
        self.nim = nim
        self.jurusan = jurusan

    def __del__(self):
        print("Objek Mahasiswa telah dihapus dari memori")

mahasiswa1 = Mahasiswa("Agus", "123456", "Teknik Informatika")
del mahasiswa1
```

Pada contoh di atas, kita menggunakan destruktor untuk mencetak pesan ketika objek mahasiswa1 dihapus dari memori dengan menggunakan keyword del. Ketika objek mahasiswa1 dihapus dari memori, destruktor del() akan dijalankan secara otomatis dan mencetak pesan "Objek Mahasiswa telah dihapus dari memori".

Dalam penggunaannya, perlu diperhatikan bahwa penggunaan destruktor harus dilakukan dengan hati-hati dan hanya digunakan ketika memang diperlukan, karena penggunaan destruktor yang tidak tepat dapat mempengaruhi kinerja program secara keseluruhan.

8. Setter dan Getter

Setter dan getter adalah method khusus pada kelas di program Python yang digunakan untuk mengatur dan mengambil nilai pada atribut objek. Setter digunakan untuk mengatur nilai atribut, sedangkan getter digunakan untuk mengambil nilai atribut. Penggunaan setter dan getter sangat membantu dalam menjaga keamanan dan kekonsistenan data pada objek.

Setter biasanya memiliki nama yang sama dengan nama atribut yang ingin diatur nilainya, dengan tambahan awalan "set_". Setter digunakan untuk memvalidasi nilai yang akan diatur ke atribut objek dan memastikan bahwa nilai yang dimasukkan memenuhi persyaratan tertentu, seperti batasan nilai atau tipe data yang diizinkan.

Getter biasanya memiliki nama yang sama dengan nama atribut yang ingin diambil nilainya, dengan tambahan awalan "get_". Getter digunakan untuk mengambil nilai dari atribut objek dan mengembalikan nilai tersebut ke pengguna atau program.

```python
class Mahasiswa:
    def __init__(self, nama, nim, jurusan):
        self.nama = nama
        self.nim = nim
        self.jurusan = jurusan

    def set_nama(self, nama):
        if len(nama) >= 3 and nama.isalpha():
            self.nama = nama
        else:
            print("Nama tidak valid")

    def get_nama(self):
        return self.nama

mahasiswa1 = Mahasiswa("Agus", "123456", "Teknik Informatika")
mahasiswa1.set_nama("Budi")
print(mahasiswa1.get_nama())
```

Pada contoh di atas, kita menggunakan setter dan getter untuk mengatur dan mengambil nilai pada atribut nama pada objek mahasiswa1 dari kelas Mahasiswa. Setter set_nama() digunakan untuk memvalidasi nilai nama yang akan diatur ke objek mahasiswa1 dan memastikan bahwa nilai tersebut memenuhi persyaratan tertentu. Getter get_nama() digunakan untuk mengambil nilai dari atribut nama pada objek mahasiswa1.

Dalam penggunaannya, penggunaan setter dan getter sangat membantu dalam menjaga keamanan dan kekonsistenan data pada objek, sehingga mengurangi kesalahan dan kerusakan yang mungkin terjadi pada program.

9. Decorator

Decorator adalah fitur di program Python yang digunakan untuk memodifikasi fungsi atau method pada kelas dengan menambahkan fungsi lain pada fungsi atau method yang sudah ada. Decorator digunakan untuk mempermudah penggunaan fungsi atau method dengan menambahkan fitur tambahan tanpa harus mengubah struktur atau isi fungsi atau method yang sudah ada.

Decorator pada dasarnya adalah fungsi yang mengambil fungsi lain sebagai argumen dan mengembalikan fungsi baru dengan menambahkan fitur tambahan pada fungsi asli. Decorator dapat digunakan untuk berbagai macam keperluan, seperti logging, caching, authentication, dan sebagainya.

MODUL 3

1. Abstraksi

   Abstraksi dalam bahasa pemrograman Python mengacu pada proses menyembunyikan detail implementasi dari suatu objek atau fungsi dan hanya mengekspos antarmuka atau fitur yang penting bagi pengguna.

   Dalam praktiknya, abstraksi dapat dicapai melalui penggunaan kelas, fungsi, dan modul. Misalnya, ketika Anda menggunakan kelas dalam Python, Anda dapat menyembunyikan detail implementasi dari atribut dan metode kelas, dan hanya mengekspos fitur-fitur yang penting bagi pengguna. Ini berarti bahwa ketika pengguna menginstansiasi kelas, mereka tidak perlu tahu detail implementasi kelas tersebut. Sebaliknya, mereka hanya perlu mengetahui antarmuka publik kelas yang didefinisikan oleh atribut dan metode yang diungkapkan.

   Penerapan abstraksi juga dapat ditemukan dalam fungsi, di mana Anda dapat membatasi akses pengguna ke detail implementasi fungsi dan hanya mengekspos fitur-fitur yang relevan. Anda dapat menggunakan

parameter fungsi untuk menerima input dari pengguna dan menghasilkan output yang diinginkan, sementara menyembunyikan detail implementasi di balik fungsi tersebut.

Modul adalah cara lain untuk mencapai abstraksi dalam Python. Dalam modul, Anda dapat menyembunyikan detail implementasi dari variabel dan fungsi, dan hanya mengekspos fitur-fitur yang relevan melalui antarmuka publik modul.

Dalam semua kasus, tujuan utama dari abstraksi adalah untuk mempermudah penggunaan objek, fungsi, atau modul, dan membuatnya lebih mudah dipahami dan digunakan oleh pengguna.

```python
class Shape:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def area(self):
        pass

    def perimeter(self):
        pass

class Circle(Shape):
    def __init__(self, x, y, radius):
        super().__init__(x, y)
        self.radius = radius

    def area(self):
        return 3.14 * self.radius ** 2

    def perimeter(self):
        return 2 * 3.14 * self.radius
```

```python
class Rectangle(Shape):
    def __init__(self, x, y, width, height):
        super().__init__(x, y)
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

    def perimeter(self):
        return 2 * (self.width + self.height)
```

Dalam contoh di atas, kelas Shape adalah kelas abstrak yang memiliki dua metode abstrak area() dan perimeter(). Kelas ini tidak dapat diinstansiasi secara langsung, tetapi digunakan sebagai superclass untuk kelas konkrit seperti Circle dan Rectangle.

Kelas Circle dan Rectangle menerapkan metode area() dan perimeter() untuk bentuk geometris yang spesifik. Namun, detail implementasi dari kedua metode ini disembunyikan dari penggunaan langsung dan hanya diakses melalui antarmuka yang ditentukan oleh kelas Shape.

Dengan cara ini, pengguna hanya perlu mengetahui antarmuka publik yang ditentukan oleh kelas Shape, dan dapat mengabaikan detail implementasi yang rumit dan tidak relevan. Misalnya, pengguna dapat dengan mudah menghitung luas dan keliling suatu bentuk geometris dengan menggunakan metode area() dan perimeter() tanpa perlu tahu detail implementasi dari kedua metode tersebut.

2. Enkapsulasi

Enkapsulasi adalah konsep dalam bahasa pemrograman Python yang memungkinkan penggunaan pembatasan akses ke atribut atau metode kelas. Dengan menggunakan enkapsulasi, atribut dan metode dapat disembunyikan dari penggunaan langsung, dan hanya dapat diakses melalui antarmuka yang ditentukan oleh kelas.

Enkapsulasi dapat dicapai dalam Python dengan menggunakan konvensi pemrograman, yaitu memberikan awalan garis bawah (underscore) pada atribut atau metode yang ingin disembunyikan dari penggunaan langsung. Konvensi ini mengindikasikan bahwa atribut atau metode bersifat "private" dan tidak seharusnya diakses secara langsung dari luar kelas.

Namun, enkapsulasi di Python masih relatif lemah dibandingkan dengan bahasa pemrograman lainnya, seperti Java atau C++. Pengguna

masih dapat mengakses atribut dan metode yang ditandai dengan garis bawah dengan cara yang tidak seharusnya, seperti dengan menggunakan nama yang sama di luar kelas.

Meskipun begitu, enkapsulasi masih berguna dalam membantu mencegah perubahan yang tidak diinginkan pada atribut atau metode kelas dari luar kelas dan memudahkan penggunaan kelas dengan hanya mengekspos antarmuka yang relevan dan aman bagi pengguna.

MODUL 4

1. Inheritance (Pewarisan)

   Inheritance atau pewarisan adalah konsep dalam pemrograman berorientasi objek di mana sebuah kelas dapat mewarisi sifat dan perilaku dari kelas lain. Dalam inheritance, kelas yang mewarisi disebut kelas anak atau subclass, sedangkan kelas yang diwarisi disebut kelas induk atau superclass.

   Dalam Python, inheritance dapat dilakukan dengan membuat kelas anak dan menentukan superclass-nya dengan menggunakan syntax class ChildClass(ParentClass):. Dengan menggunakan inheritance, kita dapat menghindari duplikasi kode dan meningkatkan modularitas kode, sehingga memudahkan pengembangan dan pemeliharaan program.

   Contoh penggunaan inheritance pada program Python adalah sebagai berikut. Kita memiliki kelas Person sebagai superclass yang memiliki dua atribut yaitu name dan age. Kemudian kita membuat dua kelas anak, yaitu Student dan Teacher, yang mewarisi atribut dan metode dari kelas Person.

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def info(self):
        print(f"Name: {self.name}, Age: {self.age}")

class Student(Person):
    def __init__(self, name, age, student_id):
        super().__init__(name, age)
        self.student_id = student_id

    def info(self):
        super().info()
        print(f"Student ID: {self.student_id}")
```

```python
class Teacher(Person):
    def __init__(self, name, age, subject):
        super().__init__(name, age)
        self.subject = subject

    def info(self):
        super().info()
        print(f"Subject: {self.subject}")

person = Person("John", 30)
person.info()

student = Student("Jane", 20, "123456")
student.info()

teacher = Teacher("Mary", 40, "Math")
teacher.info()
```

Hasil Output :

```
Name: John, Age: 30
Name: Jane, Age: 20
Student ID: 123456
Name: Mary, Age: 40
Subject: Math
```

Dalam contoh di atas, kelas Student dan Teacher mewarisi atribut dan metode dari kelas Person. Kelas Student memiliki atribut tambahan student_id, sedangkan kelas Teacher memiliki atribut tambahan subject. Kedua kelas anak juga memiliki metode info() yang mengoverride metode info() dari kelas Person. Metode info() ini memanggil metode info() dari superclass menggunakan syntax super().info() untuk menampilkan informasi atribut dari superclass, kemudian

menambahkan informasi atribut kelas anak. Kita dapat melihat bahwa kelas Student dan Teacher mewarisi atribut dan metode dari kelas Person, sehingga dapat digunakan untuk mengakses informasi dasar tentang seseorang seperti nama dan umur. Namun, kedua kelas anak juga memiliki atribut dan metode khusus mereka sendiri yang berguna untuk tugas-tugas yang berbeda.

2. Polymorphism

Polymorphism adalah konsep dalam pemrograman berorientasi objek di mana suatu objek dapat memiliki banyak bentuk atau perilaku yang berbeda dalam konteks yang berbeda pula. Dalam Python, Polymorphism dapat dicapai dengan menggunakan dua teknik, yaitu function overloading dan method overriding.

Sebagai contoh, kita akan membuat class Binatang dan class Kucing. Class Kucing merupakan turunan dari class Binatang. Class Binatang memiliki method suara(), sedangkan class Kucing memiliki method yang sama namun isinya berbeda.

```
class Binatang:
    def suara(self):
        print("Hewan ini membuat suara")

class Kucing(Binatang):
    def suara(self):
        print("Meow")

hewan1 = Binatang()
hewan2 = Kucing()

hewan1.suara()
hewan2.suara()
```

Dalam contoh ini, objek hewan1 adalah objek dari class Binatang, sehingga ketika kita memanggil method suara() pada objek tersebut, outputnya adalah "Hewan ini membuat suara". Namun, objek hewan2 adalah objek dari class Kucing, sehingga ketika kita memanggil method suara() pada objek tersebut, outputnya adalah "Meow".

Polymorphism memungkinkan kita untuk menggunakan objek dengan cara yang berbeda-beda, tergantung pada jenis objek yang digunakan. Dalam contoh di atas, method suara() dapat digunakan pada objek Binatang dan Kucing, namun menghasilkan output yang berbeda.

3. Overloading

Overloading merupakan teknik di mana kita dapat mendefinisikan beberapa fungsi dengan nama yang sama namun dengan jumlah atau tipe argumen yang berbeda. Namun, Python tidak mendukung teknik ini secara eksplisit karena Python menggunakan konsep duck typing, yang memungkinkan objek untuk memiliki perilaku yang berbeda bergantung pada konteks penggunaannya.

Sebagai contoh, kita akan membuat class PersegiPanjang dengan dua method luas(), yang satu menerima dua parameter dan yang lain menerima satu parameter. Dalam hal ini, kedua method memiliki nama yang sama, tetapi jumlah parameter yang diterima berbeda.

```
class PersegiPanjang:
    def luas(self, panjang, lebar):
        return panjang * lebar

    def luas(self, sisi):
        return sisi * sisi

persegiPanjang = PersegiPanjang()

print("Luas Persegi Panjang: ", persegiPanjang.luas(4, 5))
print("Luas Persegi: ", persegiPanjang.luas(4))
```

Dalam contoh ini, class PersegiPanjang memiliki dua method luas() yang masing-masing menerima dua parameter dan satu parameter. Saat dipanggil, program akan menentukan method mana yang harus digunakan berdasarkan jumlah parameter yang diberikan. Jika method luas() diberikan dua parameter, maka method pertama akan digunakan, dan jika method luas() diberikan satu parameter, maka method kedua yang akan digunakan.

Overloading memungkinkan kita untuk membuat code yang lebih efisien dan mudah dibaca. Dalam contoh di atas, kita hanya perlu menggunakan satu nama method untuk melakukan perhitungan luas baik untuk persegi panjang maupun persegi.

4. Overriding

Overriding adalah teknik di mana kita dapat mendefinisikan kembali metode dari superclass pada kelas anak dengan perilaku yang berbeda. Dalam Python, method overriding dilakukan dengan mendefinisikan kembali metode pada kelas anak dengan nama yang sama dengan metode pada superclass.

Sebagai contoh, kita akan membuat class Binatang dan class Kucing. Class Kucing merupakan turunan dari class Binatang. Class Binatang memiliki method suara(), sedangkan class Kucing meng-overwrite method tersebut dengan perilaku yang berbeda.

```
class Binatang:
    def suara(self):
        print("Hewan ini membuat suara")

class Kucing(Binatang):
    def suara(self):
        print("Meow")

hewan1 = Binatang()
hewan2 = Kucing()

hewan1.suara()
hewan2.suara()
```

Dalam contoh ini, class Kucing meng-overwrite method suara() dari class Binatang dengan perilaku yang berbeda, yaitu mencetak "Meow". Ketika kita memanggil method suara() pada objek hewan1, yang merupakan objek dari class Binatang, maka outputnya adalah "Hewan ini membuat suara". Namun, ketika kita memanggil method suara() pada objek hewan2, yang merupakan objek dari class Kucing, maka outputnya adalah "Meow".

Overriding memungkinkan subclass untuk memiliki perilaku yang berbeda dari superclass, sehingga memperkaya fungsionalitas class. Dalam contoh di atas, class Kucing dapat memiliki perilaku khusus yang berbeda dari class Binatang, meskipun keduanya merupakan hewan dan memiliki kemampuan untuk membuat suara.

5. Multiple Inheritance

Multiple inheritance adalah konsep dalam pemrograman berorientasi objek di mana sebuah class dapat mengambil sifat-sifat dari lebih dari satu class induk. Dalam multiple inheritance, sebuah class turunan dapat memiliki beberapa class induk dengan cara yang sama seperti dalam hierarki kelas tunggal.

Dalam pemrograman, multiple inheritance dapat membantu meningkatkan fleksibilitas dan kegunaan sebuah class dengan memungkinkannya untuk memiliki sifat-sifat dari beberapa class yang berbeda. Namun, multiple inheritance juga dapat menjadi sumber kompleksitas dan ambiguitas dalam desain class.

Sebagai contoh, kita akan membuat class kucing, anjing, dan burung, yang akan diwarisi oleh class Mamalia dan class Vertebrata:

```python
class Mamalia:
    def beranak(self):
        print("Beranak hidup")

class Vertebrata:
    def bertulangBelakang(self):
        print("Bertulang belakang")

class Kucing(Mamalia, Vertebrata):
    def __init__(self):
        Mamalia.__init__(self)
        Vertebrata.__init__(self)
        print("Kucing adalah hewan mamalia dan vertebrata")

class Anjing(Mamalia, Vertebrata):
    def __init__(self):
        Mamalia.__init__(self)
        Vertebrata.__init__(self)
        print("Anjing adalah hewan mamalia dan vertebrata")

class Burung(Vertebrata):
    def __init__(self):
        Vertebrata.__init__(self)
        print("Burung adalah hewan vertebrata")
```

```python
kucing = Kucing()
kucing.beranak()
kucing.bertulangBelakang()

anjing = Anjing()
anjing.beranak()
anjing.bertulangBelakang()

burung = Burung()
burung.bertulangBelakang()
```

Dalam contoh ini, class Mamalia dan class Vertebrata mewakili sifat-sifat yang dimiliki oleh kucing, anjing, dan burung. Class Kucing dan class Anjing mewarisi sifat-sifat dari kedua class induk, sementara class Burung hanya mewarisi sifat dari class Vertebrata saja.

Dengan multiple inheritance, kita dapat membuat class-class dengan sifat-sifat yang lebih kompleks dan berguna dalam aplikasi kita. Namun, perlu diingat bahwa desain class yang rumit dapat menyebabkan kebingungan dan kesulitan dalam pemrograman, jadi disarankan untuk merencanakan dan mempertimbangkan dengan baik sebelum menggunakan multiple inheritance.

6. Method Resolution Order di Python

Method Resolution Order (MRO) adalah urutan yang digunakan Python untuk mencari method atau atribut dalam hierarki pewarisan atau inheritance. MRO memungkinkan Python untuk menentukan urutan dalam pencarian method atau atribut pada sebuah class dan superclass, sehingga menghindari konflik dalam pewarisan atau inheritance.

Sebagai contoh, kita akan membuat tiga class yaitu A, B, dan C. Class A adalah superclass dari class B dan C. Class B dan C memiliki method masing-masing dengan nama yang sama, sedangkan class A memiliki method lain yang berbeda.

```
class A:
    def method_a(self):
        print("Ini method A")

class B(A):
    def method_b(self):
        print("Ini method B")

    def method_a(self):
        print("Ini method A dari class B")

class C(A):
    def method_c(self):
        print("Ini method C")

    def method_a(self):
        print("Ini method A dari class C")

objek = B()
objek.method_a()
```

Dalam contoh ini, class B dan C adalah subclass dari class A. Class B memiliki method method_a() dan method_b(), sedangkan class C memiliki method method_a() dan method_c(). Kedua method method_a() memiliki nama yang sama, namun isinya berbeda. Ketika kita membuat objek dari class B dan memanggil method method_a(), maka program akan mencari method yang sesuai dengan urutan MRO, yaitu class B, class A, dan object. Dalam hal ini, method yang

digunakan adalah method_a() dari class B, sehingga output yang dihasilkan adalah "Ini method A dari class B".

MRO memungkinkan Python untuk menentukan urutan dalam pencarian method atau atribut pada sebuah class dan superclass, sehingga menghindari konflik dalam pewarisan atau inheritance. Dalam contoh di atas, MRO memungkinkan objek dari class B untuk menggunakan method method_a() dari class B, meskipun terdapat method yang sama pada class A dan class C.

7. Dynamic Cast

   Dynamic cast adalah sebuah konsep pada pemrograman yang digunakan untuk mengubah tipe data dari sebuah objek secara dinamis. Pada umumnya, konversi tipe data dilakukan secara statis, yaitu pada saat kompilasi program. Namun, dengan menggunakan dynamic cast, konversi tipe data dapat dilakukan pada saat runtime, sehingga lebih fleksibel dan memungkinkan program untuk mengatasi situasi yang tidak pasti.

   Sebagai contoh, kita akan membuat class Binatang dan class Kucing. Class Kucing adalah subclass dari class Binatang. Kita juga akan membuat fungsi yang akan memanggil method suara() dari objek yang diberikan. Namun, objek yang diberikan belum tentu objek dari class Kucing, sehingga kita perlu melakukan dynamic cast untuk memastikan objek yang diberikan adalah objek dari class Kucing.

```python
class Binatang:
    def suara(self):
        print("Hewan ini membuat suara")

class Kucing(Binatang):
    def suara(self):
        print("Meow")

def panggil_suara(objek):
    kucing = None
    try:
        kucing = objek.suara()
    except AttributeError:
        pass

    if isinstance(kucing, Kucing):
        kucing.suara()
    else:
        print("Objek yang diberikan bukanlah kucing")

hewan1 = Binatang()
hewan2 = Kucing()

panggil_suara(hewan1)
panggil_suara(hewan2)
```

Dalam contoh ini, kita membuat fungsi panggil_suara() yang akan memanggil method suara() dari objek yang diberikan. Namun, objek yang diberikan belum tentu objek dari class Kucing, sehingga kita perlu melakukan dynamic cast untuk memastikan objek yang diberikan adalah objek dari class Kucing. Dalam hal ini, kita menggunakan fungsi isinstance() untuk memeriksa apakah objek kucing adalah objek dari class Kucing. Jika objek kucing adalah objek dari class Kucing, maka program akan mencetak "Meow". Jika tidak, maka program akan mencetak "Objek yang diberikan bukanlah kucing".

Dynamic cast memungkinkan program untuk mengubah tipe data dari sebuah objek secara dinamis pada saat runtime, sehingga lebih fleksibel dan memungkinkan program untuk mengatasi situasi yang tidak pasti. Dalam contoh di atas, dynamic cast memungkinkan kita untuk memastikan objek yang diberikan adalah objek dari class Kucing sebelum memanggil method suara().