

**RESUME MODUL
LAPORAN PRAKTIKUM
PEMROGRAMAN BERORIENTASI OBJEK**



Disusun oleh:

Nama : Gabriel Fico Darius
NIM : 121140069
Kelas : PBO RB

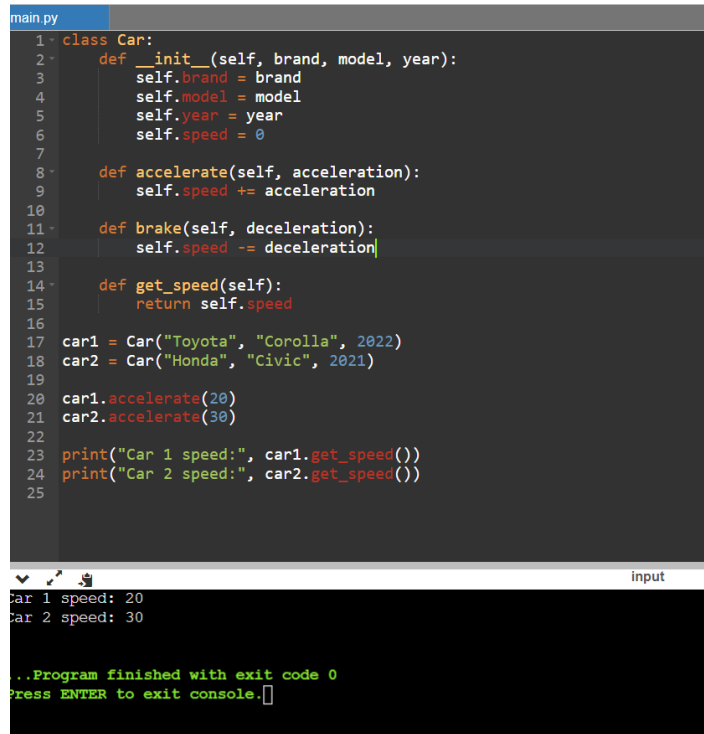
**PROGRAM STUDI TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI SUMATERA
LAMPUNG SELATAN**

2023

Modul 2

1. Kelas

Kelas atau *class* pada python ialah sebuah blueprint (cetakan) dari objek ataupun instance yang ingin dibuat. Kelas sendiri dapat mendesain objek secara leluasa. Kelas berisi dan mendefinisikan atribut dan metode untuk objeknya nanti. Kelas sendiri tidak bisa langsung digunakan, karena harus diimplementasikan sebuah objek terlebih dahulu, baru bisa disebut Instansiasi. Berikut merupakan contoh penggunaan kelas :

The image shows a screenshot of a code editor window titled 'main.py' and a terminal window below it. The code in the editor defines a 'Car' class with attributes 'brand', 'model', 'year', and 'speed'. It includes methods for initializing the object, accelerating, braking, and getting the speed. Two car objects are created: 'car1' (Toyota Corolla, 2022) and 'car2' (Honda Civic, 2021). 'car1' is accelerated by 20, and 'car2' is accelerated by 30. The speed of each car is then printed. The terminal output shows the execution of the code, displaying the speed of each car and a message indicating the program finished successfully.

```
1 class Car:
2     def __init__(self, brand, model, year):
3         self.brand = brand
4         self.model = model
5         self.year = year
6         self.speed = 0
7
8     def accelerate(self, acceleration):
9         self.speed += acceleration
10
11    def brake(self, deceleration):
12        self.speed -= deceleration
13
14    def get_speed(self):
15        return self.speed
16
17    car1 = Car("Toyota", "Corolla", 2022)
18    car2 = Car("Honda", "Civic", 2021)
19
20    car1.accelerate(20)
21    car2.accelerate(30)
22
23    print("Car 1 speed:", car1.get_speed())
24    print("Car 2 speed:", car2.get_speed())
25
```

```
Car 1 speed: 20
Car 2 speed: 30

..Program finished with exit code 0
Press ENTER to exit console.
```

Pada program ini mendefinisikan sebuah kelas bernama Car, yang memiliki empat atribut: brand, model, year, dan speed. `__init__` adalah metode khusus yang dipanggil saat objek baru dari kelas dibuat, dan digunakan untuk menginisialisasi atribut.

a. Atribut/Property

Atribut adalah variabel yang terkait dengan sebuah objek. Setiap objek memiliki atribut yang berbeda-beda, tergantung pada kelas dari objek tersebut. Atribut bisa berupa data atau objek lain yang terkait dengan objek induk. Atribut biasanya didefinisikan dalam kelas dengan menggunakan metode `__init__` atau di luar metode tersebut, dan dapat diakses dan dimanipulasi melalui objek dari kelas tersebut. Atribut juga dapat didefinisikan pada objek secara dinamis, bahkan setelah objek telah dibuat. Contoh atribut pada sebuah kelas adalah nama dan umur pada kelas Orang berikut :

```
main.py
1 class Orang:
2     def __init__(self, nama, usia):
3         self.nama = nama
4         self.usia = usia
5
6     def say_hello(self):
7         print("Halo, nama saya adalah", self.nama, "dan saya berusia ", self.usia, "tahun.")
8
9 Orang1 = Orang("John", 25)
10 print(Orang1.nama)
11 Orang1.usia = 30
12 print(Orang1.usia)
```

input

John
30

...Program finished with exit code 0
Press ENTER to exit console.

b. Method

Method adalah fungsi yang terkait dengan objek tertentu. Method biasanya didefinisikan di dalam kelas dan digunakan untuk melakukan tindakan tertentu pada objek yang sesuai dengan kelas tersebut. Method pada dasarnya adalah fungsi yang dapat diakses melalui objek kelas. Ketika sebuah objek dipanggil untuk menggunakan sebuah method, method tersebut dapat mengakses data yang terkait dengan objek tersebut dan dapat mengubah atau memanipulasi data tersebut sesuai dengan implementasi method yang didefinisikan. Untuk memanggil method pada sebuah objek, kita hanya perlu menuliskan nama objek, diikuti dengan tanda titik, diikuti dengan nama method, dan diakhiri dengan tanda kurung. Sebagai contoh:

```
main.py
1 class Orang:
2     def __init__(self, nama, usia):
3         self.nama = nama
4         self.usia = usia
5
6     def say_hello(self):
7         print("Halo, nama saya adalah", self.nama, "dan saya berusia ", self.usia, "tahun.")
8
9 Orang1 = Orang("John", 25)
10 Orang1.say_hello()
11
```

input

Halo, nama saya adalah John dan saya berusia 25 tahun.

...Program finished with exit code 0
Press ENTER to exit console.

2. Objek

Objek adalah suatu entitas yang dapat merepresentasikan data, fungsi, atau kombinasi keduanya. Setiap objek memiliki jenis (type) tertentu dan metode serta atribut yang terkait dengannya. Untuk membuat sebuah objek yang mewakili sebuah kelas, kita dapat membuatnya dengan cara memanggil nama dari kelas yang diinginkan, ditambah dengan tanda kurung (). Contoh :

```
ini_objek = kelas_yang_dipanggil()
```

3. Magic Method

Magic method berfungsi untuk mengimplementasikan perilaku khusus dalam sebuah kelas yang terkait dengan operator, fungsi, atau peristiwa tertentu. Misalnya, method "add" digunakan untuk mengimplementasikan perilaku operator "+" pada objek kelas tertentu, sedangkan "str" digunakan untuk mengimplementasikan perilaku fungsi "str()" pada objek kelas tertentu. Tujuan utama dari penggunaan magic method adalah untuk mengubah sifat (behavior) bawaan dari suatu objek. Contoh penggunaan magic method dalam Python adalah sebagai berikut:

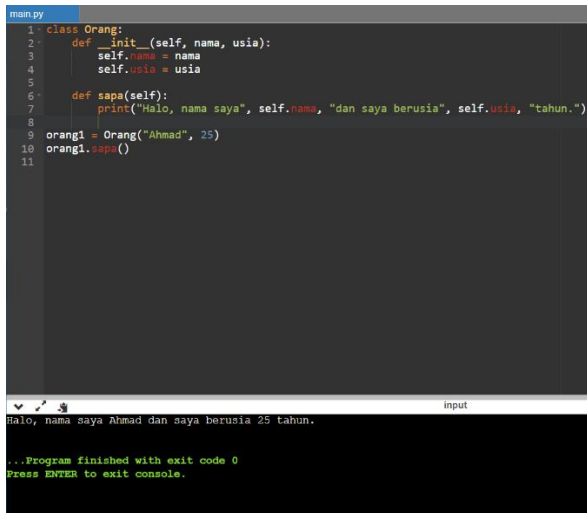
```
main.py
1 class Contoh:
2     def __init__(self, x, y):
3         self.x = x
4         self.y = y
5
6     def __add__(self, other):
7         return Contoh(self.x + other.x, self.y + other.y)
8
9     def __str__(self):
10        return "({}, {})".format(self.x, self.y)
11
12 a = Contoh(1, 2)
13 b = Contoh(3, 4)
14 c = a + b
15 print(c)
16
```

```
(4, 6)
```

```
...Program finished with exit code 0
Press ENTER to exit console.
```

4. Konstruktor

Konstruktor adalah method yang “pasti” dijalankan secara otomatis pada saat sebuah objek dibuat untuk mewakili kelas tersebut. Seperti dengan method lain, konstruktor dapat melakukan operasi seperti melakukan print. Selain operasi method dasar, konstruktor dapat menerima argumen yang diberikan ketika objek dibuat. Argumen ini akan diproses di dalam kelas nantinya. Berikut adalah contoh sederhana penggunaan konstruktor :



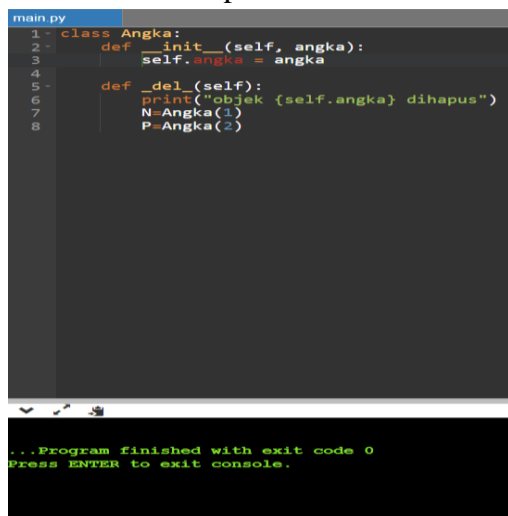
```
main.py
1 class Orang:
2     def __init__(self, nama, usia):
3         self.nama = nama
4         self.usia = usia
5
6     def sapa(self):
7         print("Halo, nama saya", self.nama, "dan saya berusia", self.usia, "tahun.")
8
9 orang1 = Orang("Ahmad", 25)
10 orang1.sapa()
11
```

```
Halo, nama saya Ahmad dan saya berusia 25 tahun.
...Program finished with exit code 0
Press ENTER to exit console.
```

Pada contoh di atas, konstruktor pada kelas "Orang" menerima dua parameter "nama" dan "usia" untuk menginisialisasi atribut "nama" dan "usia" pada objek yang dibuat dari kelas tersebut. Setelah objek "orang1" dibuat, metode "sapa" dipanggil untuk mencetak nilai atribut "nama" dan "usia" dari objek tersebut.

5. Destruktor

Destruktor adalah fungsi yang dipanggil ketika user menghapus objek. Fungsi ini bekerja secara otomatis, jadi tidak perlu dilakukan pemanggilan. Tujuan adanya destruktur adalah melakukan final cleaning up atau “bersih-bersih” terakhir sebelum sebuah objek benar-benar dihapus dari memori.



```
main.py
1 class Angka:
2     def __init__(self, angka):
3         self.angka = angka
4
5     def __del__(self):
6         print("Objek {self.angka} dihapus")
7
8 N=Angka(1)
P=Angka(2)
```

```
...Program finished with exit code 0
Press ENTER to exit console.
```

6. Setter dan Getter

Getter adalah metode yang digunakan untuk mengambil nilai atribut objek, sedangkan setter adalah metode yang digunakan untuk memodifikasi nilai atribut objek. Getter dan setter membantu dalam implementasi konsep encapsulation (enkapsulasi) dalam pemrograman berorientasi objek, yaitu cara untuk menyembunyikan atau melindungi implementasi internal objek dari penggunaan yang salah atau tidak terduga dari luar kelas.

```
main.py
1 class Orang:
2     def __init__(self, nama):
3         self.__nama = nama # atribut bersifat private
4
5     @property
6     def nama(self):
7         return self.__nama
8
9     @nama.setter
10    def nama(self, nama_baru):
11        self.__nama = nama_baru
12
13    # membuat objek Orang
14    orang1 = Orang("Ahmad")
15
16    # mengambil nilai atribut nama dengan getter
17    print(orang1.nama) # Output: "Ahmad"
18
19    # mengubah nilai atribut nama dengan setter
20    orang1.nama = "Budi"
21
22    # mengambil nilai atribut nama setelah diubah dengan getter
23    print(orang1.nama) # Output: "Budi"
24
```

```
Ahmad
Budi

...Program finished with exit code 0
Press ENTER to exit console.
```

7. Decorator

Selain menggunakan fungsi setter dan getter tambahan property decorator juga dapat dimanfaatkan untuk mendapatkan hasil serupa. Bedanya, melalui property decorator ini tidak perlu membuat fungsi lagi dengan nama yang berbeda-beda (cukup menggunakan 1 buah nama/variabel). Contoh penggunaan decorator:

```
main.py
1 # Definisikan decorator
2 def cek_angka_positif(func):
3     def wrapper(angka):
4         if angka <= 0:
5             print("Angka harus positif!")
6         else:
7             func(angka)
8     return wrapper
9
10 # Gunakan decorator untuk memeriksa input pada fungsi
11 @cek_angka_positif
12 def hitung_kuadrat(angka):
13     print(f"Kuadrat dari {angka} adalah {angka ** 2}")
14
15 # Panggil fungsi hitung_kuadrat
16 hitung_kuadrat(5) # Output: Kuadrat dari 5 adalah 25
17
18 hitung_kuadrat(-3) # Output: Angka harus positif!
19
```

```
Kuadrat dari 5 adalah 25
Angka harus positif!

...Program finished with exit code 0
Press ENTER to exit console.
```

Modul 3

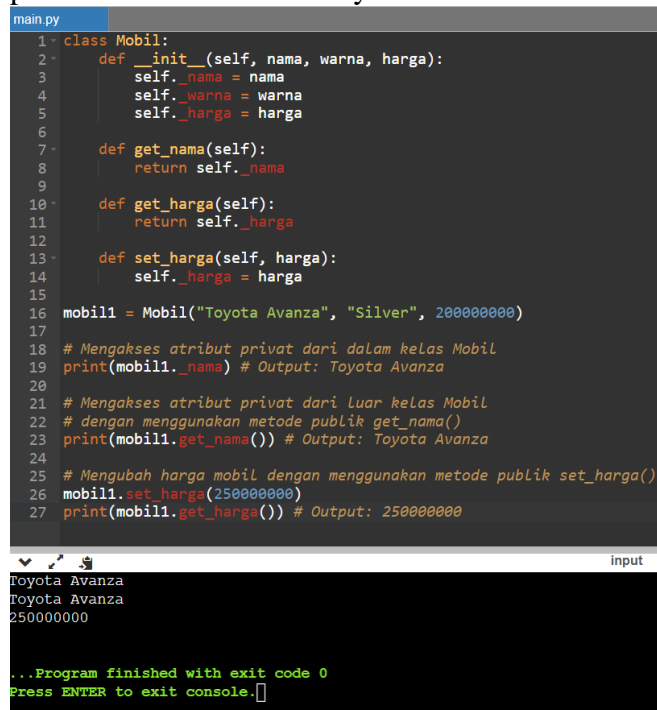
1. Abstraksi

Abstraksi adalah konsep OOP dimana model yang dibuat hanya memperlihatkan atribut yang esensial dan menyembunyikan detail-detail yang tidak penting dari user. gunanya untuk mengurangi kompleksitas. Dalam Python, abstraksi biasanya diimplementasikan dengan menggunakan kelas abstrak atau interface. Kelas abstrak adalah kelas yang tidak dapat diinisialisasi atau diinstantiasi, tetapi digunakan sebagai kerangka kerja untuk membuat kelas turunan yang lebih spesifik. Kelas abstrak dapat memiliki metode abstrak yang harus diimplementasikan oleh kelas turunannya.

Interface adalah kelas abstrak yang hanya memiliki metode abstrak, tetapi tidak memiliki implementasi apapun. Interface dapat digunakan sebagai spesifikasi antarmuka yang harus diikuti oleh kelas lain yang mengimplementasikan interface tersebut.

2. Enkapsulasi (*encapsulation*)

Dalam Python, enkapsulasi dapat dicapai dengan memberikan nama yang diawali dengan garis bawah ("_") pada atribut atau metode yang ingin dibuat privat. Atribut atau metode yang diberi nama dengan garis bawah hanya dapat diakses secara langsung dari dalam kelas tersebut, sedangkan akses dari luar kelas harus dilakukan melalui metode publik yang telah ditentukan. Berikut adalah contoh sederhana implementasi enkapsulasi pada sebuah kelas dalam Python:



```
main.py
1 class Mobil:
2     def __init__(self, nama, warna, harga):
3         self._nama = nama
4         self._warna = warna
5         self._harga = harga
6
7     def get_nama(self):
8         return self._nama
9
10    def get_harga(self):
11        return self._harga
12
13    def set_harga(self, harga):
14        self._harga = harga
15
16 mobil1 = Mobil("Toyota Avanza", "Silver", 200000000)
17
18 # Mengakses atribut privat dari dalam kelas Mobil
19 print(mobil1._nama) # Output: Toyota Avanza
20
21 # Mengakses atribut privat dari Luar kelas Mobil
22 # dengan menggunakan metode publik get_nama()
23 print(mobil1.get_nama()) # Output: Toyota Avanza
24
25 # Mengubah harga mobil dengan menggunakan metode publik set_harga()
26 mobil1.set_harga(250000000)
27 print(mobil1.get_harga()) # Output: 250000000
```

input

```
Toyota Avanza
Toyota Avanza
250000000

...Program finished with exit code 0
Press ENTER to exit console.[]
```

Pada contoh di atas, atribut nama, warna, dan harga diawali dengan garis bawah, sehingga dianggap sebagai atribut privat. Atribut ini hanya dapat diakses secara langsung dari dalam kelas Mobil. Untuk mengakses atau mengubah atribut dari luar kelas, kita harus menggunakan metode publik `get_nama()`, `get_harga()`, dan `set_harga()`. Dengan demikian, enkapsulasi membantu kita untuk membatasi akses dan melindungi atribut yang sensitif dari modifikasi yang tidak terkontrol.

Dalam pemrograman Python, aksesibilitas dari suatu atribut atau metode dalam suatu kelas dapat diatur dengan menggunakan tiga level aksesibilitas, yaitu:

a. Public Access Modifier

Atribut atau metode yang didefinisikan dengan level aksesibilitas public dapat diakses secara bebas dari dalam dan luar kelas. atribut atau metode yang tidak diawali dengan garis bawah atau dua garis bawah dianggap sebagai atribut atau metode dengan level aksesibilitas public.

```
main.py
1 class Mobil:
2     def __init__(self, nama, warna, harga):
3         self.nama = nama # public attribute
4         self.warna = warna # public attribute
5         self.harga = harga # public attribute
6
7     def get_info(self):
8         return f"{self.nama} berwarna {self.warna} dengan harga {self.harga}"
9
10 mobil1 = Mobil("Toyota Avanza", "Silver", 200000000)
11
12 print(mobil1.nama) # Output: Toyota Avanza
13 print(mobil1.warna) # Output: Silver
14 print(mobil1.harga) # Output: 200000000
15 print(mobil1.get_info()) # Output: Toyota Avanza berwarna Silver dengan harga 200000000
16
```

input

```
Toyota Avanza
Silver
200000000
Toyota Avanza berwarna Silver dengan harga 200000000

...Program finished with exit code 0
Press ENTER to exit console.
```

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. Contoh :

```
main.py
1 class Mobil:
2     def __init__(self, nama, warna, harga):
3         self._nama = nama # protected attribute
4         self._warna = warna # protected attribute
5         self._harga = harga # protected attribute
6
7     def get_info(self):
8         return f"{self._nama} berwarna {self._warna} dengan harga {self._harga}"
9
10 class SUV(Mobil):
11     def __init__(self, nama, warna, harga, kapasitas):
12         super().__init__(nama, warna, harga)
13         self._kapasitas = kapasitas # protected attribute
14
15     def get_info(self):
16         return f"{self._nama} berwarna {self._warna} dengan harga {self._harga} dan kapasitas {self._kapasitas} cc"
17
18 suv1 = SUV("Mitsubishi Pajero Sport", "Hitam", 450000000, 2500)
19
20 print(suv1._nama) # Output: Mitsubishi Pajero Sport
21 print(suv1._warna) # Output: Hitam
22 print(suv1._harga) # Output: 450000000
23 print(suv1.get_info()) # Output: Mitsubishi Pajero Sport berwarna Hitam dengan harga 450000000 dan kapasitas 2500 cc
```

input

```
Mitsubishi Pajero Sport
Hitam
450000000
Mitsubishi Pajero Sport berwarna Hitam dengan harga 450000000 dan kapasitas 2500 cc

...Program finished with exit code 0
Press ENTER to exit console.
```


c. Private Access

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. Contoh :

```
main.py
1 class Mobil:
2     #private variable
3     __merk = None
4     __merk = None
5
6     #constructor
7     def __init__(self, merk, warna):
8         self.__merk = merk
9         self.__warna = warna
10
11     #private classmethod
12     def __tampilMobil(self):
13         print("Merk Mobil : ", self.__merk)
14         print("Warna Mobil : ", self.__warna)
```

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.

```
main.py
1 class Manusia:
2     def __init__(self, nama, umur):
3         self.nama = nama
4         self.umur = umur
5
6     def cetakProfile(self):
7         print(f"{self.nama} berumur {self.umur}")
8
9 class Mahasiswa(Manusia):
10     pass
11
12 mhs1 = Mahasiswa("Daris", 19)
13 mhs1.cetakProfile()
```

```
Daris berumur 19

...Program finished with exit code 0
Press ENTER to exit console.
```

2. Polymorphism

Polymorphism berarti banyak (poly) dan bentuk (morphism), dalam Pemrograman Berbasis Objek konsep ini memungkinkan digunakannya suatu interface yang sama untuk memerintah objek agar melakukan aksi atau tindakan yang mungkin secara prinsip sama namun secara proses berbeda. Polymorphism merupakan kemampuan suatu method untuk bekerja dengan lebih dari satu tipe argumen. Contoh sederhana penggunaan polymorphism dalam Python dengan menggunakan metode yang sama pada kelas yang berbeda:

```
main.py
1 class Hewan:
2     def suara(self):
3         pass
4
5 class Anjing(Hewan):
6     def suara(self):
7         print("Guk guk")
8
9 class Kucing(Hewan):
10     def suara(self):
11         print("Meong")
12
13 # membuat objek dari kelas Anjing dan Kucing
14 dog = Anjing()
15 cat = Kucing()
16
17 # memanggil metode suara dari masing-masing objek
18 dog.suara() # Output: Guk guk
19 cat.suara() # Output: Meong
20
```

```
Guk guk
Meong

...Program finished with exit code 0
Press ENTER to exit console.
```

3. 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.

```
main.py
1 class Hewan:
2     def __init__(self, nama):
3         self.nama = nama
4
5     def suara(self):
6         print("")
7
8 class Anjing(Hewan):
9     def suara(self):
10        print("Guk guk")
11
12 class Kucing(Hewan):
13     def suara(self):
14        print("Meong")
15
16 # membuat objek dari kelas Anjing dan Kucing
17 dog = Anjing("Anjing")
18 cat = Kucing("Kucing")
19
20 # memanggil metode suara dari masing-masing objek
21 dog.suara() # Output: Guk guk
22 cat.suara() # Output: Meong
23
```

```
Guk guk
Meong

...Program finished with exit code 0
Press ENTER to exit console.
```

Dalam contoh di atas, terdapat tiga kelas yaitu Hewan sebagai superclass dan Anjing dan Kucing sebagai subclass. Kedua subclass Anjing dan Kucing memiliki metode suara() yang telah dioverride dengan implementasi yang berbeda dari metode yang diwarisi dari superclass-nya. Sehingga objek dog dan cat dapat menggunakan metode suara() dari subclass masing-masing dengan perilaku yang berbeda.

4. Overloading

Overloading adalah konsep dalam pemrograman berorientasi objek di mana sebuah fungsi atau metode dapat memiliki beberapa definisi dengan nama yang sama, tetapi memiliki parameter yang berbeda. Dalam Python, overloading tidak didukung secara eksplisit, namun dapat dicapai dengan menggunakan *args dan **kwargs dalam parameter fungsi. Contoh sederhana penggunaan overloading dalam Python menggunakan *args:

```
main.py
1 class Kalkulator:
2     def add(self, *args):
3         sum = 0
4         for arg in args:
5             sum += arg
6         return sum
7
8 # membuat objek dari kelas Kalkulator
9 kalkulator = Kalkulator()
10
11 # memanggil metode add dengan berbagai jumlah argumen
12 print(kalkulator.add(1, 2, 3)) # Output: 6
13 print(kalkulator.add(1, 2, 3, 4)) # Output: 10
14 print(kalkulator.add(1, 2, 3, 4, 5)) # Output: 15
15
```

```
6
10
15

...Program finished with exit code 0
Press ENTER to exit console.
```

5. Multiple Inheritance

Multiple inheritance adalah konsep dalam pemrograman berorientasi objek di mana sebuah kelas dapat mewarisi sifat-sifat atau metode dari dua atau lebih kelas (superclass) sekaligus. Dalam Python, multiple inheritance didukung dengan cara mewarisi dua atau lebih kelas dalam satu kelas turunan (subclass). Contoh sederhana penggunaan multiple inheritance :

```
main.py
1 class Pekerjaan:
2     def __init__(self, pekerjaan):
3         self.pekerjaan = pekerjaan
4
5 class Karyawan:
6     def __init__(self, nama, gaji):
7         self.nama = nama
8         self.gaji = gaji
9
10 class Pegawai(Pekerjaan, Karyawan):
11     def __init__(self, nama, gaji, pekerjaan):
12         Pekerjaan.__init__(self, pekerjaan)
13         Karyawan.__init__(self, nama, gaji)
14
15     def info(self):
16         print("Nama:", self.nama)
17         print("Pekerjaan:", self.pekerjaan)
18         print("Gaji:", self.gaji)
19
20 # membuat objek dari kelas Pegawai
21 pegawai = Pegawai("Daris", 5000000, "Marketing")
22
23 # memanggil metode info dari objek pegawai
24 pegawai.info()
25
```

```
Nama: Daris
Pekerjaan: Marketing
Gaji: 5000000

...Program finished with exit code 0
Press ENTER to exit console.
```

Dalam contoh di atas, terdapat tiga kelas yaitu Pekerjaan, Karyawan, dan Pegawai. Kelas Pegawai mewarisi sifat-sifat dari kelas Pekerjaan dan Karyawan. Dalam metode `__init__()`, konstruktor dari kedua kelas induk (Pekerjaan dan Karyawan) dipanggil secara terpisah dengan menggunakan `Pekerjaan.__init__(self, pekerjaan)` dan `Karyawan.__init__(self, nama, gaji)`. Kemudian, metode `info()` dipanggil pada objek pegawai, yang menampilkan informasi mengenai nama, pekerjaan, dan gaji pegawai tersebut.