

BAB I

DASAR PEMROGRAMAN PHYTON

A. Sintaks Dasar

- Variabel dan Tipe Data

Variabel adalah tempat untuk menyimpan suatu nilai atau data. Dalam pemrograman, tipe data adalah konsep penting. Variabel dapat menyimpan data dari tipe yang berbeda, dan tipe yang berbeda dapat melakukan hal yang berbeda. Berikut tipe data primitif.

Tipe Data	Jenis	Nilai
bool	Boolean	True atau false
int	Bilangan bulat	Seluruh bilangan bulat
float	Bilangan real	Seluruh bilangan real
string	Teks	Kumpulan karakter

Contoh:

```
x = 4 #integer
y = "string" #string
z = 3.14 #float
a = True #boolean
```

- Operator

Operator digunakan untuk melakukan operasi pada variabel dan nilai.

Python memiliki sejumlah operator, yaitu :

➤ Operator Aritmatika

Operator aritmatika digunakan dengan nilai numerik untuk melakukan operasi matematika umum:

Operator	Name	Example
+	Addition	$x + y$
-	Subtraction	$x - y$
*	Multiplication	$x * y$
/	Division	x / y
%	Modulus	$x \% y$
**	Exponentiation	$x ** y$
//	Floor division	$x // y$

Contoh:

```
x = 5
y = 2
print("Hasil : ", x+y)
```

➤ Operator Perbandingan

Operator perbandingan digunakan untuk membandingkan dua nilai

Operator	Name	Example
==	Equal	$x == y$
!=	Not equal	$x != y$
>	Greater than	$x > y$
<	Less than	$x < y$
>=	Greater than or equal to	$x >= y$
<=	Less than or equal to	$x <= y$

Contoh:

```
x = int(input("Masukan nilai pertama: "))
y = int(input("Masukan nilai kedua: "))
lebihKecil = x < y
print("Hasil : ", lebihKecil)
```

➤ Operator Logika

Operator logika digunakan untuk menggabungkan pernyataan bersyarat

Operator	Description	Example
and	Returns True if both statements are true	<code>x < 5 and x < 10</code>
or	Returns True if one of the statements is true	<code>x < 5 or x < 4</code>
not	Reverse the result, returns False if the result is true	<code>not(x < 5 and x < 10)</code>

Contoh:

```
2 x = int(input("Masukan nilai pertama: "))
3 y = int(input("Masukan nilai kedua: "))
4
5 operator1 = y and y<3
6 print("Hasil : ", operator1)
7
8 operator2 = x or y>3
9 print("Hasil : ", operator2)
10
11 operator3 = x and y>3 or x and y < 2
12 print("Hasil : ", operator3)
13
```

input

```
Masukan nilai pertama: 6
Masukan nilai kedua: 2
Hasil : True
Hasil : 6
Hasil : False
```

- Tipe Data Bentukan

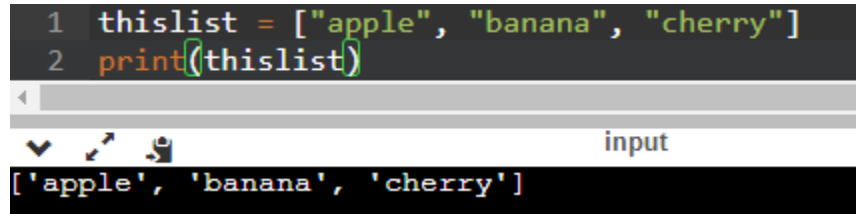
Ada 4 jenis tipe data bentuk:

➤ List

List digunakan untuk menyimpan beberapa item dalam satu variabel. List adalah salah satu dari 4 tipe data bawaan di Python yang digunakan untuk menyimpan kumpulan data, 3 lainnya adalah Tuple, Set, dan Kamus, semuanya dengan kualitas dan penggunaan yang berbeda. List dibuat menggunakan tanda kurung siku serta Sebuah kumpulan data yang terurut, dapat diubah, dan memungkinkan ada anggota yang sama.

Contoh:

```
1 thislist = ["apple", "banana", "cherry"]
2 print(thislist)
```

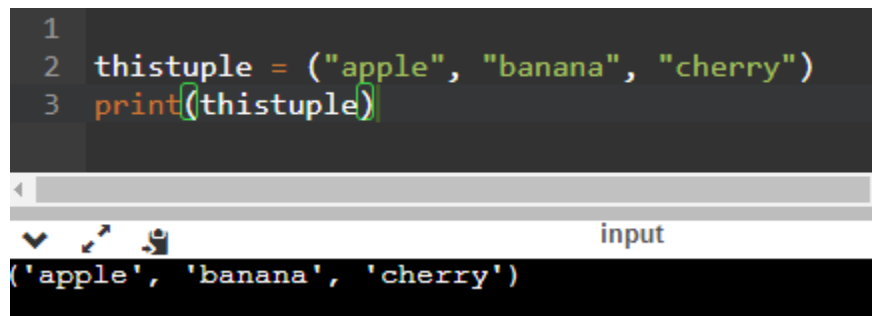


➤ Tuple

Tuple digunakan untuk menyimpan banyak item dalam satu variabel. Tuple adalah koleksi yang dipesan dan tidak dapat diubah dan ditulis dengan tanda kurung bulat.

Contoh:

```
1
2 thistuple = ("apple", "banana", "cherry")
3 print(thistuple)
```

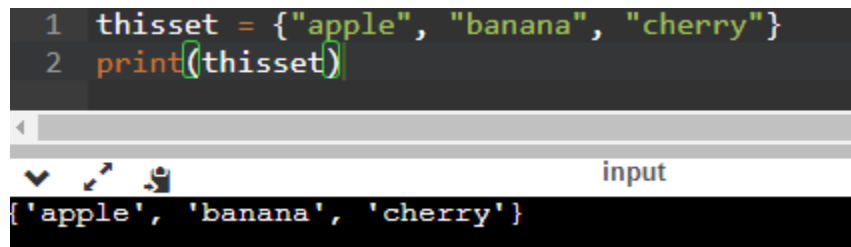


➤ Set

Set digunakan untuk menyimpan beberapa item dalam satu variabel. Himpunan adalah koleksi yang tidak terurut, tidak dapat diubah, dan tidak terindeks. Set item tidak dapat diubah, tetapi Anda dapat menghapus item dan menambahkan item baru.

Contoh:

```
1 thisset = {"apple", "banana", "cherry"}
2 print(thisset)
```



➤ dictionary

Sebuah kumpulan data yang tidak berurutan, dapat diubah, tidak memungkinkan ada anggota yang sama dan ditulis dengan kurung kurawal.

Contoh:

```
1 thisdict = {  
2     "brand": "Ford",  
3     "model": "Mustang",  
4     "year": 1964  
5 }  
6 print(thisdict)
```

input

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

- Perulangan

Terdapat 2 jenis perulangan yaitu for dan while

- For

Dengan perulangan for kita dapat menjalankan serangkaian pernyataan, satu kali untuk setiap item dalam daftar, tuple, set, dll. Perulangan for tidak memerlukan variabel pengindeksan untuk disetel sebelumnya.

Contoh:

```
1 fruits = ["apple", "banana", "cherry"]  
2 for x in fruits:  
3     print(x)
```

input

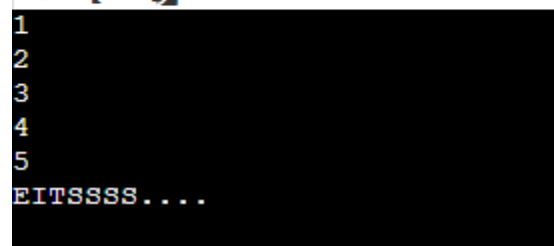
```
apple  
banana  
cherry
```

➤ While

Dengan while loop kita dapat mengeksekusi satu set pernyataan selama kondisinya benar. ingatlah untuk menaikkan i, atau loop akan berlanjut selamanya. While loop membutuhkan variabel yang relevan untuk siap

Contoh:

```
1 i = 1
2 while i < 6:
3     print(i)
4     i += 1
5 else:
6     print("EITSSSS....")
```



● Fungsi

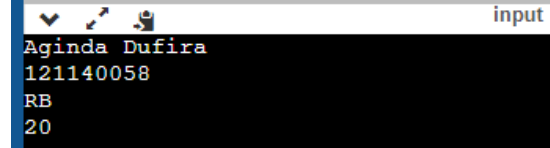
Fungsi adalah blok kode yang hanya berjalan saat dipanggil. Anda dapat meneruskan data, yang dikenal sebagai parameter, ke dalam suatu fungsi. Suatu fungsi dapat mengembalikan data sebagai hasilnya.

Contoh:

```
1
2 def tampil_data_diri(self):
3     print("Aginda Dufira")
4     print("121140058")
5     print("RB")
6     print("20")
```

Pemanggilan fungsi tersebut menggunakan

```
2 def tampil_data_diri():
3     print("Aginda Dufira")
4     print("121140058")
5     print("RB")
6     print("20")
7 tampil_data_diri()
```



```
1 def transfer(self):
2     nominal = int(input("Masukkan nominal yang ingin ditransfer: "))
3     no_rek_tujuan = int(input("Masukkan no rekening tujuan: "))
4     pelanggan_tujuan = None
5     for pelanggan in AkunBank.list_pelanggan:
6         if pelanggan.no_pelanggan == no_rek_tujuan:
7             pelanggan_tujuan = pelanggan
8             break
9     if pelanggan_tujuan is None:
10        print("No rekening tujuan tidak dikenal! Kembali ke menu utama .")
11    else:
12        if nominal > self.__jumlah_saldo:
13            print("Nominal saldo yang Anda punya tidak cukup!")
14        else:
15            self.__jumlah_saldo -= nominal
16            pelanggan_tujuan.__jumlah_saldo += nominal
17            print(f"Transfer Rp {nominal} ke {pelanggan_tujuan.nama_pelanggan} sukses!")
18
```

BAB II

OBJEK DAN KELAS DALAM PYTHON (KONSTRUKTOR, SETTER, DAN GETTER)

A. Kelas

Kelas digunakan untuk menentukan bentuk objek dan menggabungkan representasi data dan metode untuk memanipulasi data tersebut menjadi satu paket yang rapi. Data dan metode dalam kelas disebut anggota kelas. Kelas adalah deskripsi rinci, definisi, dan template dari apa yang akan menjadi objek. Tapi itu bukan objek itu sendiri. Juga, apa yang kita sebut, kelas adalah blok bangunan yang mengarah ke Pemrograman Berorientasi Objek. Ini adalah tipe data yang ditentukan pengguna, yang menyimpan anggota data dan fungsi anggotanya sendiri, yang dapat diakses dan digunakan dengan membuat turunan dari kelas tersebut. Ini adalah cetak biru dari objek apa pun. Setelah kita menulis kelas dan mendefinisikannya, kita dapat menggunakannya untuk membuat objek berdasarkan kelas itu sebanyak yang kita inginkan.

Contoh:

```
11
12 class Bike:
13     name = ""
14     gear = 0
15
```

B. Objek

Objek adalah contoh dari kelas. Semua anggota data dan fungsi anggota kelas dapat diakses dengan bantuan objek. Saat sebuah kelas didefinisikan, tidak ada memori yang dialokasikan, tetapi memori dialokasikan saat dibuat instance-nya (yaitu objek dibuat). Misalnya, mengingat objek untuk kelas Rekening adalah Rekening SBI, Rekening ICICI, dll.

Contoh:

```
1
2 # create class
3 class Bike:
4     name = ""
5     gear = 0
6
7 # create objects of class
8 bike1 = Bike()
9
```


C. Magic Method

Magic method dalam python adalah method-method spesial yang namanya diawali dan diakhiri dengan dobel *underscore*. Magic method sendiri tidak dirancang untuk kita panggil secara langsung, akan tetapi ia akan dipanggil oleh sistem secara internal pada saat-saat tertentu seperti misalnya saat membuat sebuah objek

Contoh:

```
class Angka:
    def __init__(self, angka):
        self.angka = angka

    def __add__(self, objek):
        return self.angka + objek.angka
```

D. Konstruktor

Konstruktor adalah metode yang dipanggil saat objek dibuat. Metode ini didefinisikan dalam kelas dan dapat digunakan untuk menginisialisasi variabel dasar. Jika Anda membuat empat objek, konstruktor kelas dipanggil empat kali. Setiap kelas memiliki konstruktor, tetapi tidak diharuskan untuk mendefinisikannya secara eksplisit. Di Python, konstruktor adalah metode khusus yang dipanggil saat objek dibuat. Tujuan dari konstruktor python adalah untuk menetapkan nilai ke anggota data di dalam kelas saat objek diinisialisasi. Nama metode konstruktor selalu `__init__`.

Contoh:

```
1 class Human:
2     #konstruktor
3     def __init__(self):
4         self.legs = 2
5         self.arms = 2
6
7 bob = Human()
8 print(bob.legs)
```

2

E. Destruktor

Destruktor pada python adalah sebuah fungsi yang akan dipanggil ketika suatu objek dihancurkan (destroyed) [1], atau dalam bahasa yang lebih sederhana: ketika suatu objek di-delete. Jadi bisa kita katakan bahwa destruktur adalah kebalikan dari konstruktor [2]. Jika konstruktor dipanggil saat ketika sebuah objek dibuat / diinstantiasi, maka destruktur akan dipanggil ketika sebuah objek dihapus atau ketika program selesai berjalan.

Contoh:

```
1 class Vehicle:
2     def __init__(self):
3         print('Vehicle created.')
4
5     def __del__(self):
6         print('Destructor called, vehicle deleted.')
7
8 car = Vehicle() # Di sinilah objek dibuat dan konstruktor dipanggil
9 del car # disinilah fungsi destruktur dipanggil
```

input

Vehicle created.
Destructor called, vehicle deleted.

BAB III

ABSTRAKSI DAN ENKAPSULASI (VISIBILITAS FUNGSI DAN VARIABEL RELASI ANTAR KELAS)

A. Abstraksi

Semacam namanya, abstract class merupakan class- class yang mempunyai data abstrak serta metode- metode dari sekumpulan informasi. Abstract Class tidak dapat diganti dan berlaku pula selaku kerangka dalam penciptaan subclass- subclassnya(berfungsi semacam Superclass yang dibahas di konsep Inheritance). Suatu Abstract Class mempunyai data serta tata cara yang bisa diturunkan ke subclassnya, serta segala subclass hendak menjajaki apa saja tata cara yang hendak diturunkan oleh Abstract Class.

Selaku contoh, mobil, sepeda motor, becak, adalah kendaraan. Kendaraan(Abstract Class) mempunyai syarat- syarat di mana sesuatu objek bisa dikatakan kendaraan(method and information). Wujud kendaraan semacam mobil, motor, dsb. merupakan hasil penyempitan dari kendaraan serta bertabiat lebih khusus(Subclass).

Dalam Python, abstraksi digunakan untuk menyembunyikan data/kelas yang tidak relevan untuk mengurangi kerumitan. Ini juga meningkatkan efisiensi aplikasi. Selanjutnya, kita akan belajar bagaimana kita bisa mencapai abstraksi menggunakan program Python.

Contoh:

```
1 from abc import ABC, abstractmethod
2 class Car(ABC):
3     def mileage(self):
4         pass
5
6 class Tesla(Car):
7     def mileage(self):
8         print("The mileage is 30kmph")
9 class Suzuki(Car):
10    def mileage(self):
11        print("The mileage is 25kmph ")
12 class Duster(Car):
13    def mileage(self):
14        print("The mileage is 24kmph ")
15
16 class Renault(Car):
17    def mileage(self):
18        print("The mileage is 27kmph ")
19
20 # Driver code
21 t= Tesla ()
22 t.mileage()
23
```

Output:

```
The mileage is 30kmph
```

B. Enkapsulasi

Enkapsulasi adalah salah satu konsep kunci dari bahasa berorientasi objek seperti Python, Java, dll. Enkapsulasi digunakan untuk membatasi akses ke metode dan variabel. Dalam enkapsulasi, kode dan data dibungkus bersama dalam satu unit agar tidak dimodifikasi secara tidak sengaja.

Enkapsulasi adalah mekanisme membungkus data (variabel) dan kode yang bekerja pada data (metode) bersama sebagai satu kesatuan. Dalam enkapsulasi, variabel suatu kelas akan disembunyikan dari kelas lain, dan hanya dapat diakses melalui metode kelas mereka saat ini. terdapat 3 jenis access modifier yang terdapat dalam python, yaitu public access, protected access, dan private access. Yaitu sebagai berikut:

1) Public Access Modifier

Anggota data publik dapat diakses di dalam dan di luar kelas. Semua variabel anggota kelas secara default bersifat publik

Contoh:

```
1 class Employee:
2     def __init__(self, name, salary):
3         # anggota data publik
4         self.name = name
5         self.salary = salary
6
7     # metode instance publik
8     def show(self):
9         # mengakses anggota data publik
10        print("Name: ", self.name, 'Salary:', self.salary)
11
12    emp = Employee('Jessa', 10000)
13
14    # mengakses anggota data publik
15    print("Name: ", emp.name, 'Salary:', emp.salary)
16
17    # memanggil metode publik dari kelas
18    emp.show()
```

Output:

```
Name:  Jessa Salary: 10000
Name:  Jessa Salary: 10000
```

2) Protected Access Modifier

Protected member adalah anggota kelas yang tidak dapat diakses di luar kelas tetapi dapat diakses dari dalam kelas dan subclassesnya. Untuk melakukannya dengan Python, cukup ikuti konvensi dengan mengawali nama anggota dengan satu garis bawah “_”. Meskipun variabel yang dilindungi dapat diakses di luar kelas dan juga di kelas turunan (dimodifikasi juga di kelas turunan), sudah menjadi kebiasaan (konvensi bukan aturan) untuk tidak mengakses badan kelas yang dilindungi.

Contoh:

```
1 class Company:
2     def __init__(self):
3         # Protected member
4         self._project = "NLP"
5
6 class Employee(Company):
7     def __init__(self, name):
8         self.name = name
9         Company.__init__(self)
10
11    def show(self):
12        print("Employee name :", self.name)
13        print("Working on project :", self._project)
14
15 c = Employee("Jessa")
16 c.show()
17
18 print('Project:', c._project)
```

jh

Output:

```
Employee name : Jessa
Working on project : NLP
Project: NLP
```

3) Private Access Modifier

Private member mirip dengan anggota yang dilindungi, perbedaannya adalah anggota kelas yang dideklarasikan pribadi tidak boleh diakses di luar kelas atau oleh kelas dasar mana pun. Di Python, tidak ada variabel instance Private yang tidak dapat diakses kecuali di dalam kelas. Namun, untuk menentukan awalan anggota pribadi, nama anggota dengan garis bawah ganda “__”.

Contoh:

```

1 class Employee:
2     class Employee:
3         # constructor
4     def __init__(self, name, salary):
5         self.name = name
6         # private member
7         self.__salary = salary
8
9 emp = Employee('Jessa', 100050)
10
11 # mengakses private data members
12 print('Salary:', emp.__salary)
13
14 def __init__(self, name, salary):
15     self.name = name
16     # private member
17     self.__salary = salary
18
19 emp = Employee('Jessa', 10000)
20
21 # mengakses private data members
22 print('Salary:', emp.__salary)

```

Output:

```
AttributeError: 'Employee' object has no attribute '__salary'
```

Dalam contoh di atas, gaji adalah variabel pribadi. Seperti yang Anda ketahui, kami tidak dapat mengakses variabel pribadi dari luar kelas itu.

C. 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. Terdapat 2 cara untuk membuat setter dan getter, yaitu dengan tanpa decorator dan dengan decorator.

1) Decorator

```
1 class student:
2     def __init__(self, total):
3         self._total = total
4
5     def average(self):
6         return self._total / 5.0
7
8     @property
9     def total(self):
10        return self._total
11
12    @total.setter
13    def total(self, t):
14        if t < 0 or t > 500:
15            print("Invalid Total and can't Change")
16        else:
17            self._total = t
18
19
20 o = student(450)
21 print("Total : ", o.total)
22 print("Average : ", o.average())
23 o.total = 550
24 print("Total : ", o.total)
25 print("Average : ", o.average())
```

Output:

```
Total : 450
Average : 90.0
Invalid Total and can't Change
Total : 450
Average : 90.0
```

Keterangan program tersebut adalah total properti: Properti ini bertindak sebagai pengambil untuk atribut `_total`. Ini mengembalikan nilai `_total`. `total.setter`: Ini bertindak sebagai penyetel untuk atribut `_total`. Ini menetapkan nilai `_total` ke nilai yang diteruskan. Namun, ia memeriksa apakah nilai yang diteruskan kurang dari 0 atau lebih besar dari 500. Jika ya, ia mencetak "Invalid Total and can't Change."

2) Tanpa Decorator

```
1 class Student:
2     def __init__(self, total):
3         self._total = total
4
5     def average(self):
6         return self._total / 5.0
7
8     def get_total(self):
9         return self._total
10
11    def set_total(self, t):
12        if t < 0 or t > 500:
13            print("Invalid Total and can't Change")
14        else:
15            self._total = t
16
17    total = property(get_total, set_total)
18
19
20 o = Student(450)
21 print("Total : ", o.total)
22 print("Average : ", o.average())
23 o.total = 550
24 print("Total : ", o.total)
25 print("Average : ", o.average())
26
```

Output:

```
Total : 450
Average : 90.0
Invalid Total and can't Change
Total : 450
Average : 90.0
```


BAB IV

PEWARISAN DAN POLIMORFISME (OVERLOADING, OVERRIDING, DYNAMIC CAST)

A. 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.

Syntax dasar inheritance:

```
1 class Parent:
2     pass
3
4 class Child(Parent):
5     pass
```

Contoh :

```
1 class Mahasiswa:
2     def __init__(self, nama, nim):
3         self.nama = nama
4         self.nim = nim
5
6     def cetakProfile(self):
7         print(f"{self.nama} dengan NIM {self.nim}")
8
9 class Biodata(Mahasiswa):
10     pass
11
12 bdt1 = Biodata("Aginda", 121140058)
13 bdt1.cetakProfile()
```

Output :

```
Aginda dengan NIM 121140058
```

a. Inheritance Identik

Inheritance identik merupakan pewarisan yang menambahkan constructor pada class child sehingga class child memiliki constructornya sendiri tanpa menghilangkan constructor pada class parentnya. Metode ini ditandai dengan adanya class child yang menggunakan constructor dan menggunakan kata kunci super().

Contoh :

```
1 class Mahasiswa:
2     def __init__(self, nama, nim):
3         self.nama = nama
4         self.nim = nim
5
6     def info(self):
7         print(f"{self.nama} dengan NIM {self.nim}")
8
9 class Biodata(Mahasiswa):
10     def __init__(self, nama):
11         super().__init__(nama, "121140058")
12
13 class Angkatan(Mahasiswa):
14     def __init__(self, nama):
15         super().__init__(nama, "1111222")
16
17 bdt1 = Biodata("Aginda Dufira")
18 bdt1.info()
19
20 agt1 = Angkatan("Jisoo")
21 agt1.info()
22
```

Output :

```
Aginda Dufira dengan NIM 121140058
Jisoo dengan NIM 1111222
```

B. Polymorphism

Kata polimorfisme berarti memiliki banyak bentuk. Dalam pemrograman, polimorfisme berarti nama fungsi yang sama (tetapi tanda tangan berbeda) digunakan untuk tipe yang berbeda. Perbedaan utamanya adalah tipe data dan jumlah argumen yang digunakan dalam fungsi.

Contoh:

```
1 class Cat:
2     def __init__(self, name, age):
3         self.name = name
4         self.age = age
5
6     def info(self):
7         print(f"I am a cat. My name is {self.name}. I am {self.age} years old.")
8
9     def make_sound(self):
10        print("Meow")
11
12 class Dog:
13     def __init__(self, name, age):
14         self.name = name
15         self.age = age
16
17     def info(self):
18         print(f"I am a dog. My name is {self.name}. I am {self.age} years old.")
19
20     def make_sound(self):
21         print("Bark")
22
23 cat1 = Cat("Kitty", 2.5)
24 dog1 = Dog("Fluffy", 4)
25
26 for animal in (cat1, dog1):
27     animal.make_sound()
28     animal.info()
29     animal.make_sound()
```

Output:

```
Meow
I am a cat. My name is Kitty. I am 2.5 years old.
Meow
Bark
I am a dog. My name is Fluffy. I am 4 years old.
Bark
```

C. Override/overriding

Metode overriding adalah kemampuan bahasa pemrograman berorientasi objek apa pun yang memungkinkan subkelas atau kelas anak untuk menyediakan implementasi khusus dari metode yang sudah disediakan oleh salah satu kelas super atau kelas induknya. Ketika sebuah metode dalam subkelas memiliki nama yang sama, parameter atau tanda tangan yang sama, dan tipe pengembalian (atau subtype) yang sama dengan metode di kelas supernya, maka metode di subkelas dikatakan menimpa metode di kelas super. Kelas.

Contoh:

```
1 class Parent():
2     def __init__(self):
3         self.value = "Inside Parent"
4
5     def show(self):
6         print(self.value)
7
8 class Child(Parent):
9     def __init__(self):
10        self.value = "Inside Child"
11
12    def show(self):
13        print(self.value)
14
15 obj1 = Parent()
16 obj2 = Child()
17
18 obj1.show()
19 obj2.show()
```

Output:

```
Inside Parent
Inside Child
```

D. Overloading

Dua atau lebih metode memiliki nama yang sama tetapi jumlah parameter yang berbeda atau tipe parameter yang berbeda, atau keduanya. Metode ini disebut metode kelebihan beban dan ini disebut metode overloading.

Contoh:

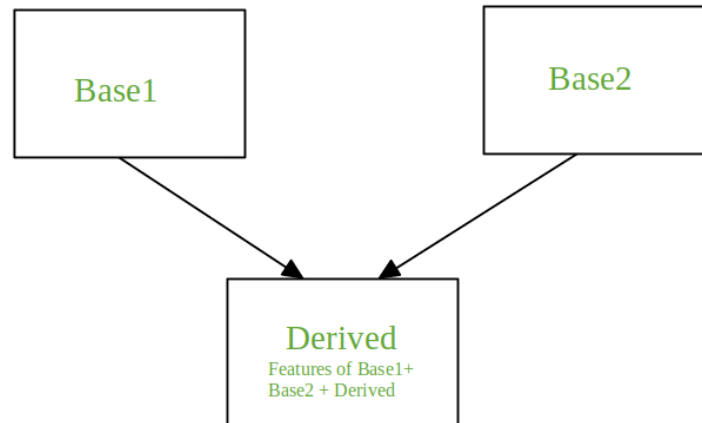
```
1 class Rectangle:
2     def __init__(self, width, height):
3         self.width = width
4         self.height = height
5
6     def area(self):
7         return self.width * self.height
8
9     def perimeter(self):
10        return 2 * (self.width + self.height)
11
12    def diagonal(self, *, diagonal_length=None):
13        if diagonal_length is None:
14            return (self.width ** 2 + self.height ** 2) ** 0.5
15        else:
16            return diagonal_length
17
18 rect1 = Rectangle(5, 10)
19 print(rect1.diagonal())
20 print(rect1.diagonal(diagonal_length=15))
```

Output:

```
11.180339887498949
15
```

E. Multiple Inheritance

Ketika sebuah kelas diturunkan dari lebih dari satu kelas dasar, itu disebut Multiple Inheritance. Kelas turunan mewarisi semua fitur dari kasus dasar.

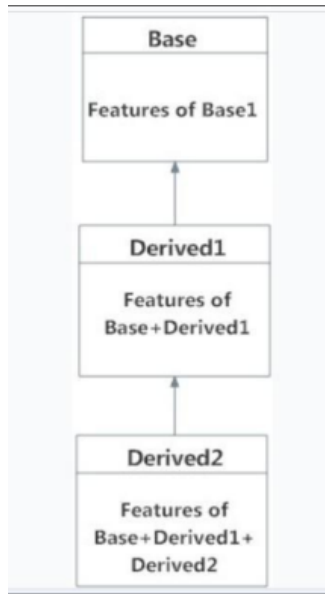


Sintak:

```
1 Class Base1:
2     Body of the class
3
4 Class Base2:
5     Body of the class
6
7 Class Derived(Base1, Base2):
8     Body of the class
```

Di bagian selanjutnya, kita akan melihat masalah yang dihadapi selama pewarisan berganda dan cara mengatasinya dengan bantuan contoh.

```
1 class Base:
2     pass
3
4 class Derived1 (Base):
5     pass
6
7 class Derived2 (Derived1):
8     pass
9
```

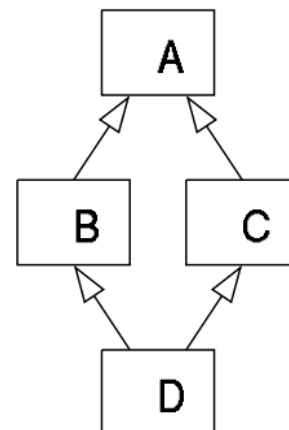


F. Method Resolution Order

MRO itu menunjukkan cara bahasa pemrograman menyelesaikan metode atau atribut. Python mendukung kelas yang diwarisi dari kelas lain. Kelas yang diwariskan disebut Parent atau Superclass, sedangkan kelas yang mewarisi disebut Child atau Subclass. Dalam python, urutan resolusi metode menentukan urutan pencarian kelas dasar saat menjalankan suatu metode. Pertama, metode atau atribut dicari di dalam kelas dan kemudian mengikuti urutan yang kami tentukan saat mewarisi. Urutan ini juga disebut Linearisasi kelas dan seperangkat aturan disebut MRO (Method Resolution Order). Saat mewarisi dari kelas lain, juru bahasa memerlukan cara untuk menyelesaikan metode yang dipanggil melalui sebuah instance. Jadi kita membutuhkan urutan resolusi metode

```

2 class A:
3     pass
4
5 class B:
6     pass
7
8 class C(A, B):
9     pass
10
11 class D(B, A):
12     pass
13
14 class E(C,D):
15     pass
  
```



Contoh:

```
1 class A:
2     def foo(self):
3         print("A")
4
5 class B(A):
6     def foo(self):
7         print("B")
8         super().foo()
9
10 class C(A):
11     def foo(self):
12         print("C")
13         super().foo()
14
15 class D(B, C):
16     def foo(self):
17         print("D")
18         super().foo()
19
20 # Membuat objek dari kelas D
21 d = D()
22 # Memanggil metode foo pada objek d
23 d.foo()
```

```
D
B
C
A
```

G. Dynamic Cast

Type Casting adalah metode untuk mengubah tipe data variabel menjadi tipe data tertentu untuk operasi yang diperlukan untuk dilakukan oleh pengguna. Pada artikel ini, kita akan melihat berbagai teknik untuk typecasting. Ada dua jenis Type Casting di Python:

1) Implisit

Dalam metode ini, Python mengubah tipe data menjadi tipe data lain secara otomatis. Dalam proses ini, pengguna tidak harus terlibat dalam proses ini.

Contoh:

```
1 x = 10
2 y = 3.5
3
4 z = x + y # Type casting implicit x menjadi float
5
6 print(z)
```



13.5

2) Eksplisit

Dalam metode ini, Python membutuhkan keterlibatan pengguna untuk mengubah tipe data variabel menjadi tipe data tertentu agar operasi yang diperlukan.

contoh:

```
1 x = "10.5"
2 y = int(float(x))
3
4 print(y)
```

10

H. Casting

1) Downcasting

Downcasting berarti typecasting objek induk ke objek anak. Downcasting tidak bisa implisit.

```
1 class Person:
2     def __init__(self, name):
3         self.name = name
4
5     def say_hello(self):
6         print(f"Hello, my name is {self.name}.")
7
8
9 class Student(Person):
10     def __init__(self, name, student_id):
11         super().__init__(name)
12         self.student_id = student_id
13
14     def say_hello(self):
15         super().say_hello()
16         print(f"My student ID is {self.student_id}.")
17
18
19 person = Person("John")
20 student = Student("Jane", "12345")
21
22 # Downcasting Person object to Student object
23 person_as_student = student.__class__(person.name, "99999")
24
25 # Calling method from Student object
26 person_as_student.say_hello()
27
```

Output:

```
Hello, my name is John.
My student ID is 99999.
```


2) Upcasting

Upcasting adalah typecasting dari objek anak ke objek induk. Upcasting dapat dilakukan secara implisit.

```
1 class Person:
2     def __init__(self, name):
3         self.name = name
4
5     def say_hello(self):
6         print(f"Hello, my name is {self.name}.")
7
8
9 class Student(Person):
10    def __init__(self, name, student_id):
11        super().__init__(name)
12        self.student_id = student_id
13
14    def say_hello(self):
15        super().say_hello()
16        print(f"My student ID is {self.student_id}.")
17
18
19 person = Person("John")
20 student = Student("Jane", "12345")
21
22 # Upcasting Student object to Person object
23 student_as_person = person.__class__(student.name)
24
25 # Calling method from Person object
26 student_as_person.say_hello()
27
```

Output:

```
Hello, my name is Jane.
```

3) Type cast

Konversi tipe kelas agar memiliki sifat/perilaku tertentu yang secara default tidak dimiliki kelas tersebut.

```

1 class Mahasiswa:
2     def __init__(self, nama, nim, ipk):
3         self.nama = nama
4         self.nim = nim
5         self.ipk = ipk
6
7     def __str__(self):
8         return f"{self.nama} ({self.nim}) - IPK: {self.ipk}"
9
10    def to_dict(self):
11        return {
12            "nama": self.nama,
13            "nim": self.nim,
14            "ipk": self.ipk
15        }
16
17    # Data awal
18    mahasiswa = Mahasiswa("AGINDA", "121140058", 4)
19
20    # Typecast ke dictionary
21    mahasiswa_dict = mahasiswa.to_dict()
22
23    # Print hasil typecast
24    print(mahasiswa_dict)

```

Output:

```
{'nama': 'AGINDA', 'nim': '121140058', 'ipk': 4}
```

DAFTAR PUSTAKA

(2023, March 14). YouTube. Retrieved April 2, 2023, from

<https://idmetafora.com/news/read/496/Memahami-OOP-Encapsulation-Inheritance-Polym>
m

Abstraction in Python. (n.d.). Javatpoint. Retrieved April 2, 2023, from

<https://www.javatpoint.com/abstraction-in-python>

Constructors in Python. (2023, March 1). GeeksforGeeks. Retrieved April 2, 2023, from

<https://www.geeksforgeeks.org/constructors-in-python/>

Difference Between Object And Class. (2022, August 25). GeeksforGeeks. Retrieved April 2,

2023, from <https://www.geeksforgeeks.org/difference-between-class-and-object/>

Encapsulation in Python. (2022, November 24). GeeksforGeeks. Retrieved April 2, 2023, from

<https://www.geeksforgeeks.org/encapsulation-in-python/>

Encapsulation in Python [Guide] – PYNative. (2021, August 28). PYNative. Retrieved April 2,

2023, from <https://pynative.com/python-encapsulation/>

Home. (n.d.). YouTube. Retrieved April 2, 2023, from

[https://jagongoding.com/python/menengah/oop/destructor/%20orphism-serta-Abstrak-Cl](https://jagongoding.com/python/menengah/oop/destructor/%20orphism-serta-Abstrak-Class.html)
ass.html

Kumar, R. (2023, January 12). *Constructors in Python: Definition, Types, and Rules*. Shiksha.

Retrieved April 2, 2023, from

[https://www.shiksha.com/online-courses/articles/constructors-in-python-definition-types-](https://www.shiksha.com/online-courses/articles/constructors-in-python-definition-types-and-rules/)
and-rules/

Kumar, R. (2023, January 12). *Constructors in Python: Definition, Types, and Rules*. Shiksha.

Retrieved April 2, 2023, from

<https://www.shiksha.com/online-courses/articles/constructors-in-python-definition-types-and-rules/>

Objective-C Classes & Objects. (n.d.). Tutorialspoint. Retrieved April 2, 2023, from

https://www.tutorialspoint.com/objective_c/objective_c_classes_objects.htm

Python: Belajar Magic Method. (2021, June 1). Jago Ngoding. Retrieved April 2, 2023, from

<https://jagongoding.com/python/menengah/oop/magic-method/>

Python Classes and Objects (With Examples). (n.d.). Programiz. Retrieved April 2, 2023, from

<https://www.programiz.com/python-programming/class>

Python Data Types. (n.d.). W3Schools. Retrieved April 2, 2023, from

https://www.w3schools.com/python/python_datatypes.asp

Python: Destructor. (2021, June 3). Jago Ngoding. Retrieved April 2, 2023, from

<https://jagongoding.com/python/menengah/oop/destructor/>

Python Operators. (n.d.). W3Schools. Retrieved April 2, 2023, from

https://www.w3schools.com/python/python_operators.asp

Python Variables. (n.d.). W3Schools. Retrieved April 2, 2023, from

https://www.w3schools.com/python/python_variables.asp

What is a constructor in Python? (n.d.). Python Tutorial. Retrieved April 2, 2023, from

<https://pythonbasics.org/constructor/>