

**RESUME**  
**PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK**



Oleh :  
M. Akbar Purnama Putra (121140065)

**PROGRAM STUDI TEKNIK INFORMATIKA**  
**JURUSAN TEKNIK ELEKTRO INFORMATIKA DAN SISTEM FISIS**  
**INSTITUT TEKNOLOGI SUMATERA**  
**LAMPUNG SELATAN**  
**2023**

# BAB I

## DASAR PEMROGRAMAN PHYTON

### A. Sintaks Dasar

- **Variabel dan Tipe Data**

Variabel merupakan 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:

```
1 x = 7 #integer
2 y = "Akbar" #string
3 z = 3.14 #float
4 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:

```
1 x = 7
2 y = 3
3 print ("Hasil : ", x+y)
4 #ouput = Hasil : 10
```

### ➤ *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	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

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)
```

Output :

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

- **Tipe Data Bentukan**

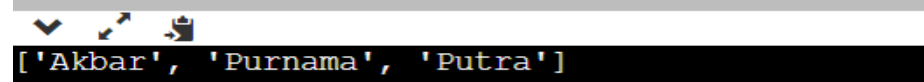
Ada 4 jenis tipe data bentuk:

- **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 = ["Akbar", "Purnama", "Putra"]
2 print(thislist)
```

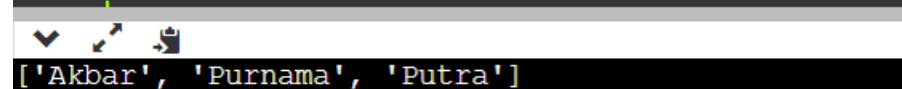


- **Tuple**

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 thistuple = ["Akbar", "Purnama", "Putra"]
2 print(thistuple)
```

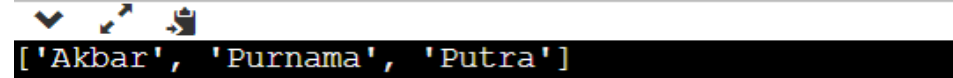


➤ **Set**

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 = ["Akbar", "Purnama", "Putra"]
2 print(thisset)
```

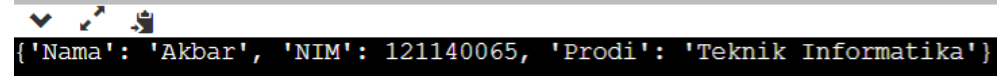


➤ **dictionary**

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

Contoh:

```
1 thisdict = {
2     "Nama" : "Akbar",
3     "NIM" : 121140065,
4     "Prodi": "Teknik Informatika"
5 }
6 print(thisdict)
```



● **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 mahasiswa = ["Akbar", "Purnama", "Putra"]
2 for x in mahasiswa :
3     print(x)
```

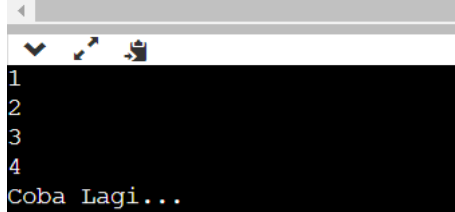


### ➤ While

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 < 5:
3     print(i)
4     i += 1
5 else:
6     print("Coba Lagi...")
```



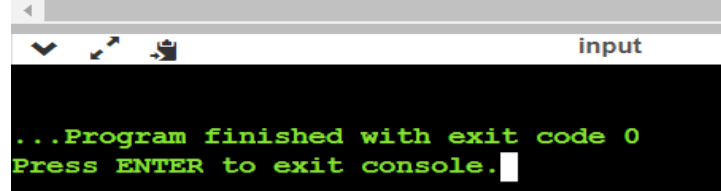
### ● Fungsi

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:

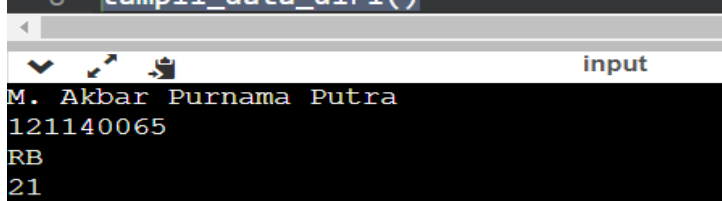
Sebelum dipanggil:

```
1 def tampil_data_diri():
2     print("M. Akbar Purnama Putra")
3     print("121140065")
4     print("RB")
5     print("21")
```



Pemanggilan fungsi tersebut:

```
1 def tampil_data_diri():
2     print("M. Akbar Purnama Putra")
3     print("121140065")
4     print("RB")
5     print("21")
6 tampil_data_diri()
```



## 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:

```
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

Abstraksi adalah cara untuk menyembunyikan informasi yang tidak dibutuhkan, sedangkan enkapsulasi adalah cara menyembunyikan atribut suatu entitas serta metode untuk melindungi informasi dari luar. 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*

Public access modifier pada Python dapat diimplementasikan dengan cara membuat variabel atau fungsi tanpa menggunakan tanda underscore (\_) pada awal nama variabel atau fungsi. Berikut adalah contoh program Python sederhana yang menggunakan public access modifier:

Contoh:

```
1 class Mobil:
2     def __init__(self, merk, tahun):
3         self.merk = merk # public variable
4         self.tahun = tahun # public variable
5
6     def info(self):
7         print("Merk mobil:", self.merk)
8         print("Tahun produksi:", self.tahun)
9
10 mobil1 = Mobil("Toyota", 2020)
11 mobil1.info()
```

Output:

```
Merk mobil: Toyota
Tahun produksi: 2020
```

Pada contoh program di atas, variabel merk dan tahun di dalam kelas Mobil didefinisikan sebagai public variable, sehingga dapat diakses dari luar kelas. Selain itu, fungsi info pada kelas Mobil juga dapat diakses dari luar kelas dan digunakan untuk menampilkan informasi tentang objek mobil1.

## 2) *Protected Access Modifier*

Protected member adalah anggota kelas yang tidak dapat diakses di luar kelas tetapi dapat diakses dari dalam kelas dan subkelasnya. Protected access modifier pada Python dapat diimplementasikan dengan menambahkan tanda underscore (\_) pada awal nama variabel atau fungsi, namun variabel atau fungsi tersebut masih dapat diakses dari kelas turunan.

Contoh:

```
1 class Mobil:
2     def __init__(self, merk, tahun):
3         self._merk = merk # protected variable
4         self._tahun = tahun # protected variable
5
6     def _info(self):
7         print("Merk mobil:", self._merk)
8         print("Tahun produksi:", self._tahun)
9
10 class SUV(Mobil):
11     def __init__(self, merk, tahun, kapasitas):
12         super().__init__(merk, tahun)
13         self._kapasitas = kapasitas # protected variable
14
15     def info(self):
16         self._info()
17         print("Kapasitas mesin:", self._kapasitas)
18
19 suv1 = SUV("Toyota", 2020, 2000)
20 suv1.info()
```

Output :

```
Merk mobil: Toyota
Tahun produksi: 2020
Kapasitas mesin: 2000
```

Pada contoh program di atas, variabel merk dan tahun di dalam kelas Mobil didefinisikan sebagai protected variable dengan menambahkan tanda underscore ( ) pada awal nama variabel, sehingga dapat diakses dari kelas turunan seperti kelas SUV. Selain itu, fungsi \_info pada kelas Mobil juga didefinisikan sebagai protected function dengan menambahkan tanda underscore ( ) pada awal nama fungsi, sehingga dapat diakses dari kelas turunan. Variabel kapasitas pada kelas SUV juga didefinisikan sebagai protected variable dengan tanda underscore ( ) pada awal nama variabel. Selain itu, kelas SUV juga memiliki fungsi info yang memanggil fungsi \_info dari kelas Mobil dan menambahkan informasi tentang kapasitas mesin. Saat dijalankan, program akan menampilkan informasi tentang objek suv1 yang terdiri dari merk mobil, tahun produksi, dan kapasitas mesin.

### 3) *Private Access Modifier*

Private access modifier pada Python dapat diimplementasikan dengan menambahkan tanda underscore (\_\_) pada awal nama variabel atau fungsi.

Contoh :

```
1 class Mobil:
2     def __init__(self, merk, tahun):
3         self.__merk = merk # private variable
4         self.__tahun = tahun # private variable
5
6     def __info(self):
7         print("Merk mobil:", self.__merk)
8         print("Tahun produksi:", self.__tahun)
9
10 mobil1 = Mobil("Toyota", 2020)
11 mobil1.__info()
```

Output:

```
Merk mobil: Toyota
Tahun produksi: 2020
```

Pada contoh program di atas, variabel merk dan tahun di dalam kelas Mobil didefinisikan sebagai private variable dengan menambahkan tanda underscore (\_\_) pada awal nama variabel, sehingga tidak dapat diakses dari luar kelas. Selain itu, fungsi \_\_info pada kelas Mobil juga didefinisikan sebagai private function dengan menambahkan tanda underscore (\_\_) pada awal nama fungsi, sehingga tidak dapat diakses dari luar kelas. Namun, jika tetap ingin mengakses variabel atau fungsi private dari luar kelas, hal tersebut masih memungkinkan dilakukan, namun tidak dianjurkan.

### 4) *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. Dikarenakan property dengan access modifier private hanya dapat diakses dari dalam kelas tersebut, dengan metode inilah kita dapat mengaksesnya dari luar kelas. Terdapat 2 cara untuk membuat setter dan getter, yaitu dengan tanpa decorator dan dengan decorator.

- Tanpa Decorator

```

1 class Mobil:
2     def __init__(self, merk, tahun):
3         self._merk = merk # protected variable
4         self._tahun = tahun # protected variable
5
6     def set_merk(self, merk):
7         self._merk = merk
8
9     def get_merk(self):
10        return self._merk
11
12    def set_tahun(self, tahun):
13        self._tahun = tahun
14
15    def get_tahun(self):
16        return self._tahun
17
18 mobil1 = Mobil("Toyota", 2020)
19 print(mobil1.get_merk())
20 mobil1.set_merk("Honda")
21 print(mobil1.get_merk())

```

Output :

```

Merk mobil: Toyota
Tahun produksi: 2020
Kapasitas mesin: 2000

```

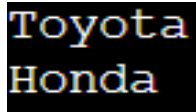
- Menggunakan Decorator

```

1 class Mobil:
2     def __init__(self, merk, tahun):
3         self._merk = merk # protected variable
4         self._tahun = tahun # protected variable
5
6     @property
7     def merk(self):
8         return self._merk
9
10    @merk.setter
11    def merk(self, merk):
12        self._merk = merk
13
14    @property
15    def tahun(self):
16        return self._tahun
17
18    @tahun.setter
19    def tahun(self, tahun):
20        self._tahun = tahun
21
22 mobil1 = Mobil("Toyota", 2020)
23 print(mobil1.merk)
24 mobil1.merk = "Honda"
25 print(mobil1.merk)

```

Output :



## 5) Object

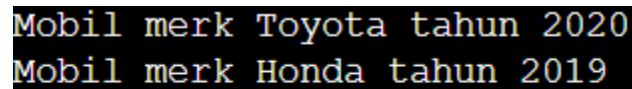
- Membuat Instance Object

Untuk membuat instance object pada program Python, Anda harus membuat objek dari suatu kelas terlebih dahulu. Dalam Python, pembuatan objek kelas dilakukan dengan cara memanggil kelas tersebut dan menyimpan hasilnya ke dalam variabel.

Contoh :

```
1 class Mobil:
2     def __init__(self, merk, tahun):
3         self.merk = merk
4         self.tahun = tahun
5
6     def info(self):
7         print("Mobil merk {} tahun {}".format(self.merk, self.tahun))
8
9 mobil1 = Mobil("Toyota", 2020)
10 mobil2 = Mobil("Honda", 2019)
11
12 mobil1.info()
13 mobil2.info()
```

Output :



Pada program di atas, kelas Mobil memiliki konstruktor yang menerima dua parameter yaitu merk dan tahun. Setiap kali objek dari kelas Mobil dibuat, konstruktor akan dipanggil untuk menginisialisasi nilai dari atribut merk dan tahun. Setelah objek mobil1 dan mobil2 dibuat dengan Mobil("Toyota", 2020) dan Mobil("Honda", 2019), masing-masing objek tersebut memiliki nilai dari atribut merk dan tahun yang berbeda. Selanjutnya, pada saat memanggil metode info, objek akan menampilkan informasi mobil dengan menggunakan nilai dari atribut merk dan tahun.

## 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 Data:
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 Mahasiswa(Data):
10     pass
11
12 mhs1 = Mahasiswa("M. Akbar Purnama Putra", 121140065)
13 mhs1.CetakProfile()
```

Output :

```
M. Akbar Purnama Putra dengan NIM 121140065
```



### 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 Data:
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 Mahasiswa(Data):
10     def __init__(self, nama):
11         super().__init__(nama, 121140065)
12
13 class Mahasiswi(Data):
14     def __init__(self, nama):
15         super().__init__(nama, 113140076)
16
17 mhs1 = Mahasiswa("M. Akbar Purnama Putra")
18 mhs1.info()
19
20 mhs2 = Mahasiswi("Livy Renata")
21 mhs2.info()
```

Output :

```
M. Akbar Purnama Putra dengan NIM 121140065
Livy Renata dengan NIM 113140076
```

### b. Polymorphism

Polymorphism adalah konsep dalam pemrograman berorientasi objek yang memungkinkan objek untuk memiliki banyak bentuk atau tampilan. Dalam Python, polymorphism dapat dicapai dengan beberapa cara, termasuk penggunaan fungsi, metode, dan kelas. Salah satu contoh penggunaan polimorfisme di Python adalah dengan menggunakan method `__len__()` yang digunakan untuk mengembalikan panjang dari sebuah objek.

Contoh :

```
1 string = "Hello, World!"
2 list = [1, 2, 3, 4, 5]
3 dictionary = {"a": 1, "b": 2, "c": 3}
4
5 print(len(string)) # Output: 13
6 print(len(list)) # Output: 5
7 print(len(dictionary)) # Output: 3
```

### c. Override/Overriding

Override atau Overriding adalah konsep dalam pemrograman berorientasi objek di mana sebuah kelas turunan dapat menimpa atau mengganti implementasi dari metode yang sudah ada di kelas dasar atau induknya. Dalam Python, kita dapat melakukan Override dengan mendefinisikan ulang metode yang ingin kita timpa dalam kelas turunan.

Contoh :

```
1 class Shape:
2     def area(self):
3         return 0
4
5 class Square(Shape):
6     def __init__(self, length):
7         self.length = length
8
9     def area(self):
10        return self.length * self.length
11
12 class Circle(Shape):
13     def __init__(self, radius):
14         self.radius = radius
15
16     def area(self):
17        return 3.14 * self.radius * self.radius
18
19 square = Square(5)
20 circle = Circle(7)
21
22 print(square.area())
```

Output :

```
25
153.86
```

Pada program di atas, Shape adalah kelas dasar yang memiliki metode area() yang mengembalikan nilai nol. Square dan Circle adalah kelas turunan dari Shape yang

mendefinisikan ulang metode `area()` dengan implementasi yang berbeda. Dalam contoh ini, kita menimpa metode `area()` pada `Shape` dengan metode `area()` pada `Square` dan `Circle`. Ketika kita memanggil metode `area()` pada objek `square` atau `circle`, Python akan mencari implementasi metode `area()` pada kelas turunan terlebih dahulu. Jika metode `area()` tidak ditemukan di kelas turunan, Python akan mencari di kelas dasar. Dalam contoh di atas, metode `area()` pada kelas turunan ditemukan dan digunakan, sehingga keluaran dari program sesuai dengan implementasi di kelas turunan.

#### d. Overloading

Overloading adalah konsep dalam pemrograman berorientasi objek di mana sebuah fungsi atau metode dapat memiliki beberapa definisi yang berbeda, tergantung pada tipe dan/atau jumlah parameter yang diberikan ke dalamnya. 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”.

Contoh :

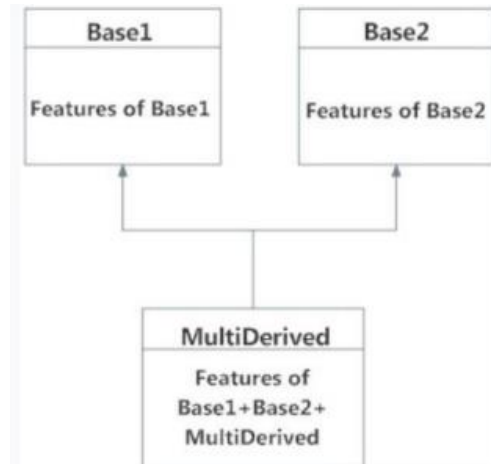
```
1 class Calculation:
2     def add(self, a, b):
3         return a + b
4
5     def add(self, a, b, c):
6         return a + b + c
7
8 calculation = Calculation()
9
10 print(calculation.add(1, 2, 3))
11 # Output: 6
```

#### e. Multiple Inheritance

Python mendukung pewarisan ke banyak kelas. Kelas dapat mewarisi dari banyak orang tua. Bentuk syntax multiple inheritance adalah sebagai berikut:

```
1 class Base1:
2     pass
3 class Base2:
4     pass
5 class MultiDerived(Base1, Base2):
6     pass
```

Dimana class MultiDerived merupakan kelas yang berasal dari clas base1 dan class base2. Adapun bentuk diagramnya adalah sebagai berikut :

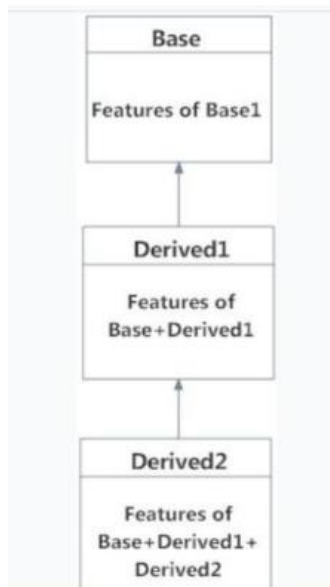


Kita juga dapat mewarisi dari kelas turunan atau disebut warisan bertingkat. Pewarisan ini dapat dilakukan hingga ke dalaman berapa pun. Dalam kelas turunan dari kelas dasar dan turunannya dapat diwarisi ke dalam kelas turunan yang baru.

Contoh :

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

Visualisasi warisan bertingkat dalam bentuk diagram :

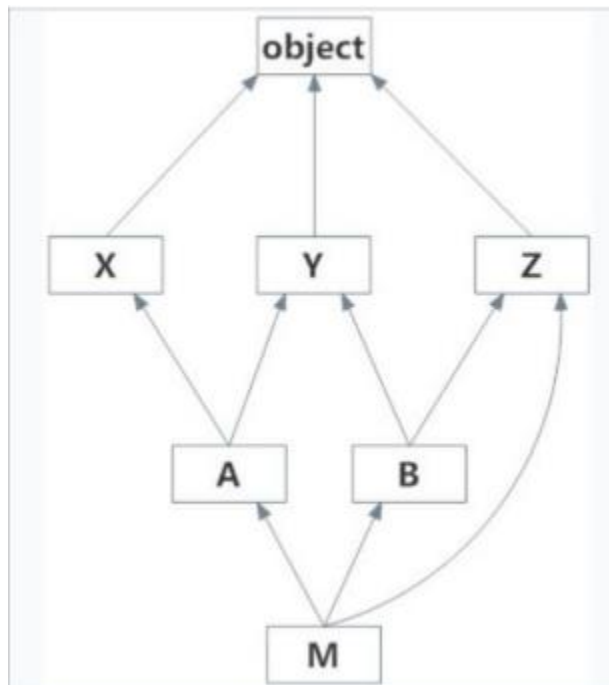


#### f. Method Resolution Order di Python

MRO adalah urutan pencarian metode dalam hirarki class. Hal ini terutama berguna dalam multiple inheritance. Urutan MRO dalam python yaitu bawah-atas dan kiri-kanan. Artinya, method dicari pertama kali di kelas objek. jika tidak ditemukan, pencarian berlanjut ke super class. Jika terdapat banyak superclass (multiple inheritance), pencarian dilakukan di kelas yang paling kiri dan dilanjutkan ke kelas sebelah kanan.

```
1 class RA:
2     def method(self):
3         print("RA.method() dipanggil")
4
5 class RB:
6     def method(self):
7         print("RB.method() dipanggil")
8
9 class RC(RA, RB):
10     pass
11
12 class RD(RB, RA):
13     pass
14
15 rc=RC()
16 rc.method()
17
18 rd=RD()
19 rd.method()
```

Visualisasi diagram :



#### 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:

- **Implisit**

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

Contoh :

```
1  num_int = 123
2  num_flo = 1.23
3
4  num_new = num_int + num_flo
5
6  print("datatype of num_int:",type(num_int))
7  print("datatype of num_flo:",type(num_flo))
8
9  print("Value of num_new:",num_new)
10 print("datatype of num_new:",type(num_new))
```

Output :

```
datatype of num_int: <class 'int'>
datatype of num_flo: <class 'float'>
Value of num_new: 124.23
datatype of num_new: <class 'float'>
```

- **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.

Contoh :

```
num_int = 123
num_str = "456"

print("Data type of num_int:",type(num_int))
print("Data type of num_str before Type Casting:",type(num_str))

num_str = int(num_str)
print("Data type of num_str after Type Casting:",type(num_str))

num_sum = num_int + num_str

print("Sum of num_int and num_str:",num_sum)
print("Data type of the sum:",type(num_sum))
```

Output :

```
Data type of num_int: <class 'int'>
Data type of num_str before Type Casting: <class 'str'>
Data type of num_str after Type Casting: <class 'int'>
Sum of num_int and num_str: 579
Data type of the sum: <class 'int'>
```

## h. Casting

- **Downcasting:** Downcasting adalah konversi dari tipe data yang lebih umum (superclass) menjadi tipe data yang lebih khusus (subclass).

Contoh :

```
class Manusia:
    def __init__(self, namadepan, namabelakang):
        self.namadepan = namadepan
        self.namabelakang = namabelakang

    def biodata(self):
        print(f"{self.namadepan} {self.namabelakang} ({self.pekerjaan})")

class Pekerja(Manusia):
    def __init__(self, namadepan, namabelakang, pekerjaan):
        super().__init__(namadepan, namabelakang)
        self.pekerjaan = pekerjaan

Andhika = Pekerja("Andhika", "Wibawa", "Mahasiswa")
Andhika.biodata()
```

Output :

```
Dhika@DESKTOP-R8DD3EM /e/P
$ D:/Apps/CommonFiles/Pytho
Andhika Wibawa (Mahasiswa)
```

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

```
class Manusia:
    namabelakang = "Putra"

    def __init__(self, namadepan, namabelakang):
        self.namadepan = namadepan
        self.namabelakang = namabelakang

    def biodata(self):
        print(f"{self.namadepan} {self.namabelakang} ({self.pekerjaan})")

class Pekerja(Manusia):
    def __init__(self, namadepan, namabelakang, pekerjaan):
        # super() adalah alias untuk kelas parent (Manusia)
        super().__init__(namadepan, namabelakang)
        self.pekerjaan = pekerjaan

    def biodata(self):
        print(f"{self.namadepan} {super().namabelakang} ({self.pekerjaan})")

Andhika = Pekerja("Andhika", "Wibawa", "Mahasiswa")
Andhika.biodata()
```

Hasil :

```
Dhika@DESKTOP-R8DD3EM /e/
$ D:/Apps/CommonFiles/Pyth
Andhika Putra (Mahasiswa)
```

- i. **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).

Contoh 1 (kelas berperilaku seperti string dan integer):

```
class Mahasiswa:

    def __init__(self, nama, nim, matkul):
        self.__nama = nama
        self.__nim = nim
        self.__matkul = matkul

    def __str__(self):
        return f"{self.__nama} ({self.__nim}) merupakan mahasiswa kelas {self.__matkul}"

    def __int__(self):
        return self.__nim

Rasyid = Mahasiswa("Rasyid", 1234, "PBO RA")
print(Rasyid) # Rasyid (1234) merupakan mahasiswa kelas PBO RA
print(int(Rasyid) == 1234) # True
```

Contoh 2 (kelas berperilaku seperti list/iterable):

```
class Mahasiswa:
    list_mahasiswa = []

    def __init__(self, *args):
        for mhs in args:
            Mahasiswa.list_mahasiswa.append(mhs)

    def __iter__(self):
        return iter(Mahasiswa.list_mahasiswa)

    def __len__(self):
        return len(Mahasiswa.list_mahasiswa)

Asprak = Mahasiswa("Andhika", "Laras", "Ihza", "Ammar")

for mahasiswa in Asprak:
    print(mahasiswa)
# Andhika Laras Ihza Ammar

print(len(Asprak))
# 4
```



## References

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

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

*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/>

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

<https://jagongoding.com/python/menengah/oop/destructor/%20orphism-serta-Abstrak-Class.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-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](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](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](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](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/>

<https://towardsdatascience.com/python-class-inheritance-62fdb33ede47>

<https://www.codesdope.com/course/python-method-overriding-and-mro/>

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

<https://www.educative.io/edpresso/what-is-mro-in-python>

<https://data-flair.training/blogs/python-multiple-inheritance/>

<https://towardsdatascience.com/duck-typing-python-7aeac97e11f8>