

RESUME PRAKTIKUM
PEMROGRAMAN BERORIENTASI OBJEK
Minggu 5



Disusun oleh :

Nama : RA Siti Zakiyah

NIM : 121140103

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI PRODUKSI DAN INDUSTRI
INSTITUT TEKNOLOGI SUMATERA
2023

A. DASAR DASAR PEMROGRAMAN

- **Variabel dan Tipe Data dalam Python**

Variabel berasosiasi dengan sebuah nilai tertentu. Python secara otomatis mengasosiasikan tipe data berdasarkan *assignment*. Berikut tabel tipe data yang ada :

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

- **Operator**

Python memiliki beberapa operator yaitu :

- a. Operator Aritmatika

Operator	Nama dan Fungsi	Contoh
+	Penjumlahan, Menjumlahkan dua buah operand	$x + y$
-	Pengurangan, mengurangkan dua buah operand	$x - y$
*	Perkalian, mengalikan dua buah operand	$x * y$
/	Pembagian, membagi dua buah operand	x / y
**	Pemangkatan, memangkatkan bilangan	$x ** y$
//	Pembagian bulat, menghasilkan hasil bagi tanpa koma	$x // y$
%	Modulus, menghasilkan sisa pembagian 2 bilangan	$x \% y$

b. Operator Perbandingan

Operator	Nama dan Fungsi	Contoh
>	Lebih besar dari – Hasilnya True jika nilai sebelah kiri lebih besar dari nilai sebelah kanan	$x > y$
<	Lebih kecil dari – Hasilnya True jika nilai sebelah kiri lebih kecil dari nilai sebelah kanan	$x < y$
==	Sama dengan – Hasilnya True jika nilai sebelah kiri sama dengan nilai sebelah kanan	$x == y$
!=	Tidak sama dengan – Hasilnya True jika nilai sebelah kiri tidak sama dengan nilai sebelah kanan	$x != y$
>=	Lebih besar atau sama dengan – Hasilnya True jika nilai sebelah kiri lebih besar atau sama dengan nilai sebelah kanan	$x >= y$
<=	Lebih kecil atau sama dengan – Hasilnya True jika nilai sebelah kiri lebih kecil atau sama dengan nilai sebelah kanan	$x <= y$

c. Operator Penugasan

Operator	Penjelasan	Contoh
=	Menugaskan nilai yang ada di kanan ke operand di sebelah kiri	$c = a + b$ menugaskan $a + b$ ke c
+=	Menambahkan operand yang di kanan dengan operand yang ada di kiri dan hasilnya ditugaskan ke operand yang di kiri	$c += a$ sama dengan $c = c + a$
-=	Mengurangi operand yang di kanan dengan operand yang ada di kiri dan hasilnya ditugaskan ke operand yang di kiri	$c -= a$ sama dengan $c = c - a$
*=	Mengalikan operand yang di kanan	$c *= a$ sama

	dengan operand yang ada di kiri dan hasilnya ditugaskan ke operand yang di kiri	dengan $c = c * a$
$/=$	Membagi operand yang di kanan dengan operand yang ada di kiri dan hasilnya ditugaskan ke operand yang di kiri	$c /= a$ sama dengan $c = c * a$
$**=$	Memangkatkan operand yang di kanan dengan operand yang ada di kiri dan hasilnya ditugaskan ke operand yang di kiri	$c **= a$ sama dengan $c = c ** a$
$//=$	Melakukan pembagian bulat operand di kanan terhadap operand di kiri dan hasilnya disimpan di operand yang di kiri	$c //= a$ sama dengan $c = c // a$
$%=$	Melakukan operasi sisa bagi operand di $c \%= a$ sama dengan kanan dengan operand di kiri dan hasilnya disimpan di operand yang di kiri	$c = c \% a$

d. Operator Logika

Operator	Penjelasan	Contoh
and	Hasilnya adalah True jika kedua operandnya bernilai benar	$x \text{ and } y$
or	Hasilnya adalah True jika salah satu atau kedua operandnya bernilai benar	$x \text{ or } y$
not	Hasilnya adalah True jika operandnya bernilai salah (kebalikan nilai)	$\text{not } x$

e. Operator Bitwise

Operator	Nama	Contoh
&	Bitwise AND	$x \& y = 0 (0000\ 0000)$
	Bitwise OR	$x y = 14 (0000\ 1110)$
\sim	Bitwise NOT	$\sim x = -11 (1111\ 0101)$
\wedge	Bitwise XOR	$x \wedge y = 14 (0000\ 1110)$

>>	Bitwise right shift	$x>> 2 = 2 (0\ 000\ 0010)$
<<	Bitwise left shift	$x<< 2 = 40 (0010\ 1000)$

f. Operator Identitas

Operator	Penjelasan	Contoh
is	True jika kedua operand identik (menunjuk ke objek yang sama)	$x \text{ is True}$
is not	True jika kedua operand tidak identik (tidak merujuk ke objek yang sama)	$x \text{ is not True}$

g. Operator Keanggotaan

Operator	Penjelasan	Contoh
in	True jika nilai/variabel ditemukan di dalam data	$5 \text{ in } x$
not in	True jika nilai/variabel tidak ada di dalam data	$5 \text{ not in } x$

- **Tipe Data Bentukan**

Tipe Data Dasar	Contoh nilai	Penjelasan
List	[1, 2, 3, 4, 5] atau ['apple', 'banana', 'cherry'] atau ['xyz', 768, 2.23]	Data untaian yang menyimpan berbagai tipe data dan isinya bisa diubah-ubah
Tuple	('xyz', 1, 3.14)	Data untaian yang menyimpan berbagai tipe data tapi isinya tidak bisa diubah
Dictionary	{ 'firstName': 'Joko', 'lastName': 'Widodo' }	Data untaian yang menyimpan berbagai tipe data berupa pasangan penunjuk dan nilai
Set	{ 'apple', 'banana', 'cherry' }	Data untaian yang menyimpan berbagai tipe

		data dan elemen datanya harus unik
--	--	------------------------------------

- **Percabangan**

Percabangan dibuat saat terdapat sebuah kondisi. contohnya seperti berikut.

- a. Percabangan IF

```

1 x = int(input("Masukan nilai x : "))
2 if(x>0):
3     print(str(x)+" bilangan positif")
4

```

- b. Percabangan IF-ELSE

```

2 x = int(input("Masukan nilai x : "))
3 if(x>0):
4     print(str(x)+" bilangan positif")
5
6 else:
7     print(str(x)+" bilangan negatif")
8

```

- c. Percabangan IF-ELSE-IF

```

1
2 x = int(input("Masukan nilai x : "))
3 if(x>0):
4     print(str(x)+" bilangan positif")
5
6 elif (x < 0):
7     print(str(x)+" bilangan negatif")
8
9 else:
10    print(str(x)+" adalah nol")
11

```

- **Perulangan**

Terdapat dua perulangan dalam python, yaitu perulangan `for` dan perulangan `while`. Perintah `for` biasanya digunakan untuk mengulangi kode yang sudah diketahui banyak perulangannya. Perintah `while` untuk perulangan yang memiliki syarat dan tidak tentu berapa banyak perulangannya.

Contoh perulangan `for`

```
11  
12 for i in range(5):  
13     print(i)  
14
```

Contoh perulangan `while`

```
1  
2 passw = input("Masukkan password: ")  
3 real_passw = "235711"  
4  
5 while passw != real_passw:  
6     print("Password tidak sesuai :(")  
7     passw = input("Masukkan password: ")  
8  
9 print("OK :)")
```

- **Fungsi dalam Python**

Setiap fungsi dalam python ditandai dengan keyword `def`. Tipe kembalian dan tipe argumen fungsi dalam Python tidak perlu didefinisikan secara eksplisit.

```
4 def faktorial(n):  
5     if (n <= 1):  
6         return 1  
7     else:  
8         return n * faktorial(n-1)  
9
```

B. OBJEK DAN KELAS DALAM PYTHON

- **Kelas**

Kelas atau *class* suatu rancangan/blueprint dari objek yang akan dibuat yang berisi atribut dan metode untuk objeknya nanti. Satu buah kelas dapat membuat banyak objek. Kelas tidak bisa langsung digunakan, harus diimplementasikan menjadi sebuah objek atau instansiasi.

```

3 - class Mobil:
4     # inisialisasi atribut
5 -     def __init__(self, nama, warna, merek):
6         self.nama = nama
7         self.warna = warna
8         self.merek = merek
9
10    # metode
11    def maju(self):
12        print("Mobil " + self.nama + " ini berjalan maju")
13
14    def mundur(self):
15        print("Mobil " + self.nama + " ini berjalan mundur")
16
17 # object
18 prius = Mobil("Prius", "Hitam", "Toyota") # instansiasi
19 prius.maju()

```

Dalam 1 sebuah kelas terdapat variabel (atribut/properti) dan fungsi (method).

a. Atribut

Atribut merupakan sebuah variabel yang berisi data/informasi dari suatu objek yang didefinisikan di kelas. Terdapat dua jenis atribut, yaitu atribut kelas dan atribut objek.

```

2
3 - class Mobil:
4     jumlah_mobil = 0
5
6 -     def __init__(self):
7         Mobil.jumlah_mobil += 1
8
9 innova = Mobil()
10 print(innova.jumlah_mobil)
11
12 avanza = Mobil()
13 print(avanza.jumlah_mobil)
14
15 fortuner = Mobil()
16 print(fortuner.jumlah_mobil)
17

```

Atribut kelas adalah atribut yang datanya dibagi ke semua objek dari kelas disebut atribut kelas

```
3 class Mobil:
4     def __init__(self, nama):
5         self.nama = nama
6
7 innova = Mobil("Innova")
8 avanza = Mobil("Avanza")
9
10 print(innova.nama) #Innova
11 print(avanza.nama) #Avanza
12
```

Atribut objek adalah atribut yang datanya hanya menjadi milik objek tersebut disebut atribut objek.

b. Method

Method merupakan fungsi yang menyatakan proses-proses yang bisa dilakukan oleh objek. Fungsi merupakan sifat/perilaku dari objek yang didefinisikan di kelas. Method `__init__()` merupakan method konstruktor, method khusus yang digunakan Python untuk menginisialisasi pembuatan objek dari kelas tersebut yang selalu dijalankan di awal saat proses instansiasi. Method tersebut dipanggil secara otomatis saat proses instansiasi, digunakan untuk menetapkan data (nilai) ke atribut objek atau operasi lain saat proses instansiasi, untuk membuat objek dari sebuah kelas, kita bisa memanggil nama kelas diikuti oleh tanda kurung dengan argumen sesuai dengan method `__init__()` pada saat kita mendefinisikannya.

```
2
3 class Orang:
4     def __init__(self, nama, pekerjaan, tahun_lahir):
5         self.nama=nama
6         self.pekerjaan=pekerjaan
7         self.tahun_lahir=tahun_lahir
8
9     def berjalan_kedepan(self):
10        print("Melangkah")
11        print("Posisi berpindah")
12
13 oranggila=Orang("Bela", "Keliling Pasar", 1998)
14
15 oranggila.berjalan_kedepan()
16
```

Output

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

• Objek

Objek adalah sesuatu yang diciptakan dari kelas, sehingga merupakan bentuk “nyata” atau perwujudan dari kelas. Sebuah objek terdiri dari satu kelas. Setiap objek dapat berkomunikasi/berinteraksi dengan objek lainnya

```
14  
15 oranggila.berjalan_kedepan()  
16
```

• Magic Method

Magic method adalah metode yang diawali dan diakhiri dengan double underscore (dunder). Method ini tidak dipanggil secara langsung, tapi dipanggil sistem secara internal ketika melakukan sesuatu seperti menggunakan operator tambah (`__add__`), membuat objek (`__init__`), dan lain-lain.

```
2  
3 class Angka:  
4     def __init__(self, angka):  
5         self.angka=angka  
6  
7     def __add__(self, objek):  
8         return self.angka+objek.angka  
9  
10 N=Angka(1)  
11 P=Angka(2)  
12 print(N+P)  
13
```

Output

```
3  
—  
  
...Program finished with exit code 0  
Press ENTER to exit console.□
```

- **Konstruktor**

Konstruktor adalah method yang “pasti” dijalankan secara otomatis pada saat sebuah objek dibuat untuk mewakili kelas tersebut. Konstruktor dapat melakukan operasi seperti melakukan print dan dapat menerima argumen yang diberikan ketika objek dibuat. Argumen ini akan diproses di dalam kelas nantinya.

```
3 class kelas_yang_dipanggil:
4     #konstruktor
5     def __init__(self, argumen, argumen_juga):
6         self.nama=argumen
7         self.pekerjaan=argumen_juga
8     #buat panggil nama
9     def panggil_nama(self):
10        print("Orang ini bernama", self.nama)
11    #buat panggil pekerjaan
12    def panggil_pekerjaan(self):
13        print("Orang ini berkerja sebagai", self.pekerjaan)
14
15 ini_objek=kelas_yang_dipanggil("Jeki", "Programmer")
16
17 ini_objek.panggil_nama()
18 ini_objek.panggil_pekerjaan()
```

Output

```
Orang ini bernama Jeki
Orang ini berkerja sebagai Programmer

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

- **Destruktor**

Destruktor adalah fungsi yang dipanggil ketika user menghapus objek. Fungsi ini bekerja secara otomatis, jadi tidak perlu dilakukan pemanggilan.

```
2
3 class Angka:
4     def __init__(self, angka):
5         self.angka=angka
6
7     def __del__(self):
8         print("objek{self.angka} dihapus")
9
10 N=Angka(1)
11 P=Angka(2)
```

Output

```
objek{self.angka} dihapus  
objek{self.angka} dihapus  
  
...Program finished with exit code 0  
Press ENTER to exit console.█
```

• Setter dan Getter

Setter dan getter digunakan untuk melakukan enkapsulasi agar tidak terjadi perubahan data secara tidak sengaja. Setter adalah method yang digunakan untuk menetapkan nilai suatu atribut khususnya atribut private dan protected (akan diajarkan lebih jelas pada materi minggu selanjutnya), sedangkan getter digunakan untuk mengambil nilai.

```
2  
3 class Mahasiswa:  
4     def __init__(self, umur=0):  
5         self.umur=umur  
6  
7     #getter  
8     def get_umur(self):  
9         return self.umur  
10  
11    #setter  
12    def set_umur(self, x):  
13        self.umur=x  
14  
15 jeki=Mahasiswa()  
16  
17 jeki.set_umur(20)  
18 print(jeki.get_umur())  
19 print(jeki.umur)  
20
```

Output

```
20  
20  
  
...Program finished with exit code 0  
Press ENTER to exit console.█
```

- **Decorator**

Sama seperti setter dan getter namun ini kita tidak perlu membuat fungsi lagi dengan nama yang berbeda-beda (cukup menggunakan 1 buah nama/variabel) dengan memanfaatkan property decorator.

```
2
3 - class Mahasiswa:
4 -     def __init__(self, nama, umur=20):
5 -         self.nama=nama
6 -         self.umur=umur
7
8     @property
9 -     def umur(self):
10 -         print("Fungsi getter umur dipanggil")
11 -         return self.umur
12
13     @umur.setter
14 -     def umur(self, x):
15 -         print("Fungsi setter umur dipanggil")
16 -         self.umur=x
17
18 jeki=Mahasiswa("Jeki")
19
20 print(jeki.umur)
21 jeki.umur+=3
22 print(jeki.umur)
23
```

C. ABSTRAKSI DAN ENKAPSULASI

- **Abstraksi**

Abstraksi digunakan untuk menyembunyikan detail fungsionalitas sebuah fungsi dari pengguna. Pengguna hanya berinteraksi dengan implementasi dasar dari fungsi tersebut, tetapi detail kerja bagian dalam disembunyikan. Saat mendesain sebuah kode program, kita dapat memisahkan interface dan implementasi sehingga menghasilkan beberapa manfaat, yaitu dapat menyembunyikan data yang tidak relevan untuk mengurangi kompleksitas, menghindarkan terjadinya duplikasi kode, membantu untuk meningkatkan keamanan dari sebuah aplikasi atau program dan memudahkan dalam pengembangan selanjutnya, karena kode di dalam dapat diubah secara individual tanpa menyebabkan error pada pengguna.

- **Enkapsulasi**

Enkapsulasi adalah cara menyembunyikan atribut suatu entitas serta metode untuk melindungi informasi dari luar. Untuk membatasi hak akses terhadap property dan method dalam suatu kelas, terdapat 3 jenis access modifier yang terdapat dalam python, yaitu public access, protected access, dan private access.

- a. Public Access Modifier

Public Access Modifier adalah setiap class, variable dan method yang dibuat secara default.

```
2
3 - class Mahasiswa:
4     global_variabel_jumlah=0
5 -     def __init__(self, nama, semester):
6         self.nama=nama
7         self.semester=semester
8         Mahasiswa.global_variabel_jumlah=Mahasiswa.global_variabel_jumlah+1
9
10    def print_jumlah(self):
11        print("total mahasiswa adalah ", Mahasiswa.global_variabel_jumlah, " Orang")
12
13
14    def print_profil(self):
15        print("Nama : ", self.nama)
16        print("Semester : ", self.semester)
17        print()
```

- b. Protected Access Modifier

Variabel dan method yang dideklarasikan secara protected hanya dapat diakses oleh kelas turunan darinya. Cara mendeklarasikannya dengan menambahkan 1 underscore (_) sebelum variabel atau method.

```
3 - class Mprotected:
4     #protected variabel
5     _merk=None
6     _warna=None
7
8     #konstruktor
9     def __init__(self, merk, warna):
10        self._merk=merk
11        self._warna=warna
12
13     #protected method
14     def _tampilMobil(self):
15         print("Merk mobil : ", self._merk)
16         print("Warna mobil : ", self._warna)
```

c. Private Access Modifier

Variabel dan method yang dideklarasikan secara private hanya dapat diakses di dalam kelas itu sendiri, namun private masih bisa diakses dari luar kelas dengan cara `nama_object_instance._NamaKelas_nama_atribut`. Dalam mendeklarasikannya, hanya perlu menambahkan double underscore (__) sebelum nama variabel dan methodnya.

```
3+ class Mprotected:
4      #private variabel
5      __merk=None
6      __warna=None
7
8      #konstruktor
9      def __init__(self, merk, warna):
10         self.__merk=merk
11         self.__warna=warna
12
13     #private method
14     def _tampilMobil(self):
15         print("Merk mobil : ", self.__merk)
16         print("Warna mobil : ", self.__warna)
17
```

d. Getters and Setters

Tujuan utama menggunakan getter dan setter adalah memastikan enkapsulasi data. Getter dan setter sering digunakan ketika menambahkan logika validasi dari mendapatkan (get) dan menetapkan (set) suatu nilai dan untuk menghindari akses langsung yaitu variabel bersifat private.

Contoh fungsi property :

```
3+ class Mahasiswa:
4      def __init__(self, name=""):
5          self._name = name
6
7      def get_name(self):# fungsi mengambil nilai name
8          return self._name
9
10     def set_name(self, new_name): # fungsi menetapkan nilai name
11         self._name = new_name
12
13     def del_name(self):
14         del self._name
15
16     name = property(get_name, set_name, del_name)
17
18 mahasiswa_if =Mahasiswa()
19 mahasiswa_if.name= "aldi"
20 print(mahasiswa_if.name)
21
```

Contoh Decorator @property

```
3 - class Mahasiswa:
4 -     def __init__(self, name=""):
5 -         self._name = name
6 -
7 -     @property
8 -     def name(self): # fungsi mengambil nilai name
9 -         return self._name
10 -    |
11 -    @name.setter
12 -    def name(self, new_name): # fungsi menetapkan nilai
13 -        self._name = new_name
14 -
15 - mahasiswa_if =Mahasiswa()
16 - mahasiswa_if.name= "alди"
17 - print(mahasiswa_if.name)
18 -
```

Contoh Decorator @classmethod

```
3 - from datetime import date
4 -
5 - class Manusia:
6 -     def __init__(self, nama, umur):
7 -         self.nama = nama
8 -         self.umur = umur
9 -
10 -    # membuat sebuah objek manusia dari tanggal Lahir
11 -    @classmethod
12 -    def dari_tahun_lahir(cls, nama, tahun):
13 -        return cls(nama, date.today().year - tahun)
14 -
15 - kiki = Manusia("kiki", 21)
16 - sandi = Manusia.dari_tahun_lahir("sandi", 1996)
17 - print(kiki.umur) # 21 tahun
18 - print(sandi.umur) # 27 tahun
19 -
```

Contoh Decorator @staticmethod

```
3 - from datetime import date
4 -
5 - class Manusia:
6 -     def __init__(self, nama, umur):
7 -         self.nama = nama
8 -         self.umur = umur
9 -
10 -    # membuat sebuah objek manusia dari tanggal Lahir
11 -    @classmethod
12 -    def dari_tahun_lahir(cls, nama, tahun):
13 -        return cls(nama, date.today().year - tahun)
14 -
15 -    # sebuah static method u/ mengecek apakah sudah dewasa
16 -    @staticmethod
17 -    def apakah_dewasa(umur):
18 -        return umur > 18
19 -
20 - print(Manusia.apakah_dewasa(20)) #True
21 -
```

D. INHERITANCE DAN POLYMORPHISM

- **Inheritance (Pewarisan)**

Inheritance adalah konsep OOP di mana kelas mewarisi atribut dan fungsi dari orang tuanya (atau kakek-nenek, dan seterusnya) tanpa perlu mengimplementasikan kembali atribut dan fungsi.

```
3 - class Hewan:
4 -     def __init__(self, nama, usia):
5 -         self.nama = nama
6 -         self.usia = usia
7 -
8 -     def makan(self):
9 -         print(self.nama, "sedang makan...")
10-
11- class Kucing(Hewan):
12-     def __init__(self, nama, usia, heterochromia):
13-         super().__init__(nama, usia)
14-         self.heterochromia = heterochromia
15-
16 kucing1=Hewan("Puspus", 1)
17 kucing1.makan()
18-
```

Output

```
Puspus sedang makan...

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

- **Polymorphism**

Polymorphism merupakan kemampuan suatu method untuk bekerja dengan lebih dari satu tipe argumen, konsep ini sering disebut dengan method overloading.

```
2
3 - class Tomato:
4 -     def type(self):
5 -         print("Vegetable")
6
7 -     def color(self):
8 -         print("Red")
9
10 - class Apple():
11 -     def type(self):
12 -         print("Fruit")
13
14 -     def color(self):
15 -         print("Red")
16
17 - def func(obj):
18 -     obj.type()
19 -     obj.color()
20
21 obj_tomato=Tomato()
22 obj_apple=Apple()
23 func(obj_tomato)
24 func(obj_apple)
25
```

- **Override/Overriding**

Override/Overriding adalah 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.

```
3  class bangunDatar():
4      def tampil(self):
5          print("Ini bangun datar")
6
7
8  class Persegi(bangunDatar):
9      def tampil(self):
10         print("Ini persegi")
11
12 P1 = Persegi()
13 P1.tampil()
```

Output

```
Ini persegi

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

- **Overloading**

Metode overloading mengizinkan sebuah class untuk memiliki sekumpulan fungsi dengan nama yang sama dan argumen yang berbeda. Akan tetapi, Python tidak mengizinkan pendeklarasian fungsi (baik pada class ataupun tidak) dengan nama yang sama. Hal itu menyebabkan implementasi overloading pada python menjadi “tricky”.

```
3 - class Duck:
4 -     def __init__(self, nama):
5 -         self.nama = nama
6 -
7 -     def quack(self):
8 -         print("Quack!")
9 -
10 -    class Car:
11 -        def __init__(self, model):
12 -            self.model = model
13 -
14 -        def quack(self):
15 -            print("I can't quack, too!")
16 -
17 -    def quacks(obj):
18 -        obj.quack()
19 -
20 - donald=Duck("Donald Duck")
21 - car=Car("Tesla")
22 - quacks(donald)
23 - quacks(car)
```

Output

```
Quack!
I can't quack, too!

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

- **Multiple Inheritance**

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

```

3  class Leluhur:
4      def __init__(self, nama, jk):
5          self.nama = nama
6          self.jk = jk
7      def bekerja(self):
8          teks = "{} menangkap ikan di laut"
9          return teks.format(self.nama)
10
11 class Ayah(Leluhur):
12     def __init__(self, nama, jk):
13         super().__init__(nama, jk)
14     def bekerja(self):
15         teks = "{} bertani di sawah"
16         return teks.format(self.nama)
17
18 class Ibu(Leluhur):
19     def __init__(self, nama, jk):
20         super().__init__(nama, jk)
21     def bekerja(self):
22         teks = "{} memasak di dapur"
23         return teks.format(self.nama)
24
25 class Anak(Ibu, Ayah):
26     def __init__(self, nama, jk):
27         super().__init__(nama, jk)
28
29 anak1 = Anak("Mustafa", "Pria")
30 print(anak1.bekerja())
31

```

Output

```

Mustafa memasak di dapur

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

```

- **Method Resolution Order di Python**

MRO adalah urutan pencarian metode dalam hierarki class. Hal ini terutama berguna dalam multiple inheritance. 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.

```
3 - class A:
4 -     def method(self):
5 -         print("A.method() dipanggil")
6 -
7 - class B:
8 -     def method(self):
9 -         print("B.method() dipanggil")
10 -
11 - class C(A, B):
12 -     pass
13 -
14 - class D(B, A):
15 -     pass
16 -
17 - c=C()
18 - c.method
19 -
20 - d=D()
21 - d.method
```

- **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.

- a. Tipe Implisit

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

```
3 num_int=123
4 num_flo=1.123
5
6 num_new=num_int+num_flo
7
8 print("datatype of num_int:", type(num_int))
9 print("datatype of num_flo:", type(num_flo))
10
11 print("Value of num_new", num_new)
12 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 124.123
datatype of num_new: <class 'float'>

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

b. Eksplisit

Pengguna mengubah tipe data sebuah objek ke tipe data lainnya dengan fungsi yang sudah ada dalam python seperti int(), float(), dan str(). dapat berisiko terjadinya kehilangan data.

```
3 num_int=123
4 num_str="456"
5
6 print("datatype of num_int:", type(num_int))
7 print("datatype of num_str before type casting:", type(num_str))
8
9 num_str=int(num_str)
10 print("datatype of num_str after type casting:", type(num_str))
11
12 num_sum=num_int+num_str
13
14 print("Sum of num_int and num_str", num_sum)
15 print("datatype of the sum:", type(num_sum))
16
```

Output

```
datatype of num_int: <class 'int'>
datatype of num_str before type casting: <class 'str'>
datatype of num_str after type casting: <class 'int'>
Sum of num_int and num_str 579
datatype of the sum: <class 'int'>

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

• Casting

- a. Downcasting: Parent class mengakses atribut yang ada pada kelas bawah (child class).

```

3 - class Manusia:
4 -     def __init__(self, namadepan, namabelakang):
5 -         self.namadepan=namadepan
6 -         self.namabelakang=namabelakang
7 -
8 -     def biodata(self):
9 -         print(f"{self.namadepan} {self.namabelakang} ({self.pekerjaan})")
10-
11- class Pekerja(Manusia):
12-     def __init__(self, namadepan, namabelakang, pekerjaan):
13-         super().__init__(namadepan, namabelakang)
14-         self.pekerjaan=pekerjaan
15-
16 Jeki=Pekerja("Jeki", "Yaa", "Mahasiswa")
17 Jeki.biodata()
18

```

Output

```

Jeki Yaa (Mahasiswa)

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

```

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

```

3 - class Manusia:
4 -     namabelakang="Lestari"
5 -
6 -     def __init__(self, namadepan, namabelakang):
7 -         self.namadepan=namadepan
8 -         self.namabelakang=namabelakang
9 -
10-    def biodata(self):
11-        print(f"{self.namadepan} {self.namabelakang} ({self.pekerjaan})")
12-
13- class Pekerja(Manusia):
14-     def __init__(self, namadepan, namabelakang, pekerjaan):
15-         super().__init__(namadepan, namabelakang)
16-         self.pekerjaan=pekerjaan
17-
18-     def biodata(self):
19-         print(f"{self.namadepan} {super().namabelakang} ({self.pekerjaan})")
20-
21 Jeki=Pekerja("Jeki", "Yaa", "Mahasiswa")
22 Jeki.biodata()
23

```

Output

```

Jeki Lestari (Mahasiswa)

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

```

- c. Type casting: Konversi tipe kelas agar memiliki sifat/perilaku tertentu yang secara default tidak dimiliki kelas tersebut.

```
 3 class Mahasiswa:  
 4  
 5     def __init__(self, nama, nim, matkul):  
 6         self.nama=nama  
 7         self.nim=nim  
 8         self.matkul=matkul  
 9  
10    def __str__(self):  
11        return f'{self.nama} ({self.nim}) merupakan mahasiswa kelas {self.matkul}'  
12  
13    def __int__(self):  
14        return self.nim  
15  
16 Jeki=Mahasiswa("Jeki", 121140103, "PBO RB")  
17 print(Jeki)  
18 print(int(Jeki)==121140103)  
19
```

Output

```
Jeki (121140103) merupakan mahasiswa kelas PBO RB  
True  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```