

PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK
(RB)



Disusun oleh :

Ahmad Zain Mahmud (121140232)

PROGRAM STUDI TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI SUMATERA

2023

1 MODUL 1

1.1 Pengenalan Bahasa Pemrograman Python

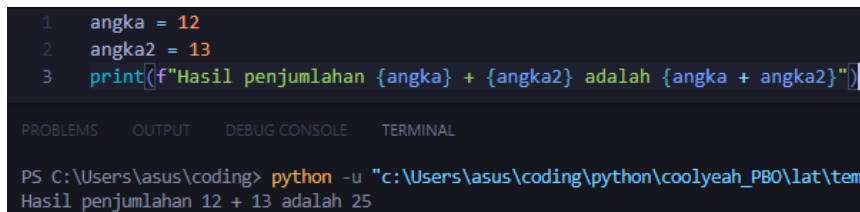
Python adalah bahasa pemrograman tingkat tinggi yang biasa digunakan dalam pengembangan perangkat lunak, komputasi, kecerdasan buatan, pengembangan aplikasi web, dan banyak lagi. Bahasa pemrograman ini dibuat oleh Guido van Rossum pada tahun 1989 dan sejak saat itu menjadi salah satu bahasa pemrograman terpopuler di dunia.

Python adalah bahasa pemrograman tingkat tinggi yang biasa digunakan dalam pengembangan perangkat lunak, komputasi, kecerdasan buatan, pengembangan aplikasi web, dan banyak lagi. Bahasa pemrograman ini dibuat oleh Guido van Rossum pada tahun 1989 dan sejak saat itu menjadi salah satu bahasa pemrograman terpopuler di dunia.

1.2 Dasar Pemrograman Python

1.2.1 Operator Aritmatika

Operator aritmatika adalah operator yang digunakan untuk melakukan operasi matematika seperti penjumlahan, pengurangan, perkalian dan pembagian dll.



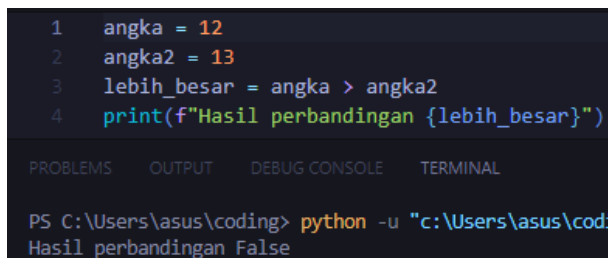
```
1 angka = 12
2 angka2 = 13
3 print(f"Hasil penjumlahan {angka} + {angka2} adalah {angka + angka2}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Users\asus\coding> python -u "c:\Users\asus\coding\python\coolyeah_PBO\lat\tem
Hasil penjumlahan 12 + 13 adalah 25

1.2.2 Operator Perbandingan

Operator perbandingan merupakan operator yang dipakai untuk membandingkan 2 buah nilai.



```
1 angka = 12
2 angka2 = 13
3 lebih_besar = angka > angka2
4 print(f"Hasil perbandingan {lebih_besar}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Users\asus\coding> python -u "c:\Users\asus\codi
Hasil perbandingan False

1.2.3 Tipe data bentukan

Terdapat 4 Tipe data bentukan: List, Tuple, Set dan Dictionary.

- List

```
1 list = [1, 2, 3, 4, 5]
2 list2 = ["pisang", "apel", "jeruk"]
```

- Tuple

```
ex = (1, 2, "belion")
```

- Set

```
x = {"anggur", "leci", "pepaya"}
```

- Dictionary

```
dictionary = {'nama_depan' : 'Ahmad', 'Zain' : 'Mahmud'}
```

1.2.4 Percabangan

Terdapat sejumlah tipe percabangan dalam pemrograman bahasa Python:

- IF

```
1 n = int(input("Masukan bilangan: "))
2 if n < 0:
3     print(n, "bilangan negatif")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\asus\coding> python -u "c:\Users\as
Masukan bilangan: -1
-1 bilangan negatif
```

- IF-ELIF

```
1 n = int(input("Masukan bilangan: "))
2 if n < 0:
3     print(n, "bilangan negatif")
4 else:
5     print(n, "bilangan positif")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\asus\coding> python -u "c:\Users\as
Masukan bilangan: 2
2 bilangan positif
```

- IF-ELIF-ELSE

```
1 n = int(input("Masukan bilangan: "))
2 if n < 0:
3     print(n, "bilangan negatif")
4 elif n == 0:
5     print(n, "bilangan nol")
6 else:
7     print(n, "bilangan positif")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Users\asus\coding> python -u "c:\Users\as
Masukan bilangan: 0
0 bilangan nol
```

1.2.5 Perulangan For

Pada perulangan for biasa digunakan untuk iterasi pada urutan berupa list, tuple, atau string

```
1 # contoh perulangan for pada list
2 x = ["ucup", "gehrman", "klein"]
3 for i in x:
4     print(i)
5
6 # contoh perulangan for
7 print('contoh lain perulangan for')
8 for i in range(5):
9     print(i)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Users\asus\coding> python -u "c:\Users\as
ucup
gehrman
klein
contoh lain perulangan for
0
1
2
3
4
```

2 MODUL 2

2.1 Kelas

Kelas adalah cetak biru atau templat untuk membuat objek tertentu dengan properti dan perilaku yang sama. Kelas digunakan untuk mengelompokkan fungsi dan variabel menjadi satu kesatuan yang disebut objek

```
1 class Hero:
2     def __init__(self, inputName, InputHealth, inputAttack, inputArmor):
3         self.name = inputName
4         self.health = InputHealth
5         self.attack = inputAttack
6         self.armor = inputArmor
7
8 hero = Hero("Eizer", 100, 23, 50)
9 print(hero.name)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Users\asus\coding> python -u "c:\Users\asus\coding\python\pbo_KT\oop#2.py"

Eizer

2.1.1 Atribut/Property

Atribut mengacu pada variabel yang disimpan dalam suatu objek. Atribut ini dapat diberi nilai saat objek dibuat, dan nilainya dapat berubah selama eksekusi program. Atribut juga dapat diakses dan dimodifikasi menggunakan metode kelas objek yang ada. Dalam hal ini, atribut dapat digunakan sebagai ciri atau informasi tentang objek yang berguna dalam proses pemrograman .

```
1 class Hero:
2     def __init__(self, inputName, InputHealth, inputAttack, inputArmor):
3         self.name = inputName
4         self.health = InputHealth
5         self.attack = inputAttack
6         self.armor = inputArmor
7
8 hero = Hero("Eizer", 100, 23, 50)
9 print("Nama hero:",hero.name)
10 print("Health hero:",hero.health)
11 print("Attack hero:",hero.attack)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Users\asus\coding> python -u "c:\Users\asus\coding\python\pbo_KT\oop#2.py"

Nama hero: Eizer
Health hero: 100
Attack hero: 23

2.1.2 Method

Metode adalah fungsi yang didefinisikan dalam kelas yang bertanggung jawab untuk memanipulasi objek kelas itu. Metode dapat dianggap sebagai tindakan atau perilaku dari suatu objek. Metode juga dapat digunakan untuk memanipulasi dan memodifikasi atribut suatu objek atau kelas

```

1 class Hero:
2     def __init__(self, inputName, inputArmor, inputAttack, InputHealth):
3         self.name = inputName
4         self.armor = inputArmor
5         self.health = InputHealth
6         self.attack = inputAttack
7
8     def method(self):
9         print('nama hero adalah ' + self.name)
10
11 hero = Hero("Jajang", 120, 15, 200)
12 hero.method()

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Users\asus\coding> python -u "c:\Users\asus\coding\python\coolyeah_PBO\lat\tempCo
nama hero adalah Jajang

2.2 Objek

Objek adalah turunan atau implementasi dari kelas. Objek memiliki atribut yang mewakili karakteristik atau propertinya, dan metode yang mewakili perilaku atau tindakan yang dapat dilakukan objek.

```
ashborn = Hero("Ashborn", 100, 10)
```

2.3 Magic Method

Magic method adalah metode khusus dalam Python yang memungkinkan kita memanipulasi perilaku objek yang ada, seperti operator aritmatika atau pembandingan, dengan mendefinisikan metode dengan nama khusus yang dimulai dan diakhiri dengan garis bawah ganda, mis. misalnya "__menambahkan__". atau "__eq__". Magic method memungkinkan kita menyesuaikan bahasa Python dengan kebutuhan kita untuk menggunakan objek. Contoh metode sulap yang umum digunakan adalah __init__ untuk menginisialisasi objek saat dibuat, __str__ untuk mengubah representasi objek menjadi string, dan __eq__ untuk membandingkan kemiripan dua objek.

```

1 class Mangga:
2     def __init__(self, jenis, jumlah):
3         self.jenis = jenis
4         self.jumlah = jumlah
5
6     def __str__(self):
7         return f"Mangga {self.jenis}, jumlah {self.jumlah}"
8
9 belanja = Mangga("Arum manis", 10)
10 print(belanja)

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Users\asus\coding> python -u "c:\Users\asus\coding\python\coolyeah
Mangga Arum manis, jumlah 10

2.4 Konstruktor

Konstruktor atau konstruktor adalah metode khusus yang berjalan secara otomatis saat objek dibuat. Konstruktor umumnya digunakan untuk menginisialisasi nilai awal pada atribut suatu objek agar objek tersebut dapat digunakan dengan benar. Dalam bahasa pemrograman Python, konstruktor ditandai dengan metode `init()`, diikuti dengan parameter `self` dan parameter lain yang diperlukan untuk menginisialisasi objek.

```
class Hero:
    # ini adalah konstruktor
    def __init__(self, inputName, InputHealth, inputAttack, inputArmor):
        self.name = inputName
        self.health = InputHealth
        self.attack = inputAttack
        self.armor = inputArmor

    # ini adalah method
    def ex(self):
        print(['ini adalah method'])
```

2.5 Destruktor

Destruktor dalam bahasa pemrograman adalah metode khusus yang dipanggil saat objek dihapus dari memori. Destruktor biasanya digunakan untuk membersihkan sumber daya yang digunakan oleh target. Dalam Python, destruktur diwakili oleh metode `"__del__"`.

```
1 class MyClass:
2     def __init__(self, n):
3         self.n = n
4
5     def __del__(self):
6         print(f"objek {self.n} telah dihapus")
7
8 obj = MyClass(9) # Constructor dipanggil
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\asus\coding> python -u "c:\Users\asus\coding\p
objek 9 telah dihapus
```

3 MODUL 3

3.1 Abstraksi

Abstraksi dalam proses penyederhanaan kompleksitas sistem hanya memperhatikan aspek-aspek penting dari objek dan menyembunyikan detail implementasi dari pengguna. Hal ini dilakukan untuk memudahkan penggunaan dan pengembangan program.

```

1 class Karyawan:
2     def __init__(self, nama, gaji):
3         self.nama = nama
4         self.gaji = gaji
5
6     def setGaji(self, plus):
7         self.gaji += plus
8         return self.gaji
9
10    def info(self):
11        return f>Nama karyawan adalah {self.nama} dengan gaji {self.gaji:,}"
12
13    awan = Karyawan("Awan", 5000000)
14    awan.setGaji(1000000)
15    print(awan.info())

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Users\asus\coding> python -u "c:\Users\asus\coding\python\coolyeah_PBO\lat\tempCodeRu
>Nama karyawan adalah Awan dengan gaji 6,000,000

3.2 Enkapsulasi (Encapsulation)

Enkapsulasi adalah teknik pemrograman berorientasi objek yang digunakan untuk mengatur struktur kelas yang menyembunyikan detail dan alur kerja kelas tersebut. Struktur kelas yang relevan berisi properti dan metode. Dengan konsep enkapsulasi, kita dapat membatasi akses ke properti dan metode tertentu dari suatu kelas sehingga tidak dapat diakses dari luar kelas. Dalam Python, terdapat tiga jenis access modifier, yaitu public access, protected access, dan private access.

```

1 class Karyawan:
2     def __init__(self, nama, gaji):
3         self.nama = nama
4         self.gaji = gaji
5
6     def info(self):
7         return f>Nama karyawan adalah {self.nama} dengan gaji {self.gaji:,}"
8
9    awan = Karyawan("Awan", 5000000)
10    print(awan.info())
11    awan.gaji = 7000000
12    print(awan.info())

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Users\asus\coding> python -u "c:\Users\asus\coding\python\coolyeah_PBO\lat\tempCodeRu
>Nama karyawan adalah Awan dengan gaji 5,000,000
>Nama karyawan adalah Awan dengan gaji 7,000,000

3.3 Setter dan Getter

Setter dan getter adalah metode atau fungsi pemrograman objek yang digunakan untuk mengakses dan mengubah nilai variabel objek (atribut). Setter digunakan untuk mengubah nilai atribut sedangkan getter digunakan untuk mendapatkan nilai atribut.


```

class Karyawan:
    def __init__(self, nama, gaji):
        self.__nama = nama
        self.__gaji = gaji

    @property
    def nama(self):
        pass

    # fungsi setter
    @nama.setter
    def nama(self, nama_baru):
        self.__nama = nama_baru

    # fungsi getter
    @nama.getter
    def nama(self):
        return self.__nama

    def info(self):
        return f"{self.__nama} memiliki gaji {self.__gaji}"

widya = Karyawan("Widya", 6000000)
print(widya.info())
widya.nama = "Wira"
print(widya.info())

```

```

le.py
Widya memiliki gaji 6000000
Wira memiliki gaji 6000000
PS C:\Users\asus\coding>

```

4 MODUL 4

4.1 Inheritance (Pewarisan)

Inheritance adalah konsep OOP (pemrograman berorientasi objek) yang memungkinkan Anda membuat kelas baru dengan mewarisi properti dan metode kelas yang ada. Kelas yang diwariskan disebut superclass atau superclass, sedangkan kelas yang diwariskan disebut subclass atau subclass

```

class Kendaraan:
    def __init__(self, jenis, warna):
        self.jenis = jenis
        self.warna = warna

    def info(self):
        print(f"Kendaraan jenis {self.jenis} dengan warna {self.warna}")

class Mobil(Kendaraan):
    def __init__(self, jenis, warna, merek):
        super().__init__(jenis, warna)
        self.merek = merek

    def info(self):
        print(f"Mobil merek {self.merek} dengan jenis {self.jenis} dan warna {self.warna}")

```

4.2 Polymorphism

Polymorphism adalah kemampuan untuk memperlakukan objek dengan cara yang sama meskipun objek tersebut berasal dari kelas yang berbeda. Di OOP, ini dicapai dengan menggunakan metode dengan nama yang sama di kelas yang

berbeda.

```
class Mobil:
    def __init__(self, merk, warna):
        self.merk = merk
        self.warna = warna

    def info(self):
        print(f"Mobil merk {self.merk} dengan warna {self.warna}")

class Motor:
    def __init__(self, merk, warna):
        self.merk = merk
        self.warna = warna

    def info(self):
        print(f"Motor merk {self.merk} dengan warna {self.warna}")

def obj(objek):
    objek.info()

xjr = Motor("Yamaha", "Putih")
inova = Mobil("Inova", "Hitam")
obj(xjr)
obj(inova)
```

eRunnerFile.py"
Motor merk Yamaha dengan warna Putih
Mobil merk Inova dengan warna Hitam
PS C:\Users\asus\coding> █

4.3 Override/Overriding

Overriding adalah teknik dimana metode yang sudah ada di kelas utama didefinisikan ulang di subclass untuk menggantikan implementasi metode kelas utama di kelas turunan dengan implementasi yang berbeda. Overriding memungkinkan kita membuat perilaku yang berbeda untuk metode yang sama di kelas yang berbeda.

```
class Binatang:
    def suara(self):
        print("tidak ada suara")

class Anjing(Binatang):
    def suara(self):
        print("Guk Guk!!")

class Kucing(Binatang):
    def suara(self):
        print("Meoww")
```

4.4 Overloading

Overloading dalam bahasa pemrograman adalah kemampuan untuk mendefinisikan beberapa metode atau fungsi dengan nama yang sama tetapi dengan jumlah atau jenis parameter yang berbeda.

```

class Point:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def __add__(self, other):
        x = self.x + other.x
        y = self.y + other.y
        return Point(x, y)

p1 = Point(2, 3)
p2 = Point(-1, 2)
p3 = p1 + p2
print(p3.x, p3.y)

```

4.5 Multiple Inheritance

Multiple inheritance adalah suatu konsep dimana suatu kelas dapat diwariskan dari lebih dari satu kelas super atau superclass. Ini memungkinkan kelas anak memiliki properti dari lebih dari satu kelas induk dan memberikan lebih banyak fleksibilitas untuk penggunaan kembali kode.

```

class A:
    def method_A(self):
        print("Ini adalah method dari kelas A")

class B:
    def method_B(self):
        print("Ini adalah method dari kelas B")

class C(A, B):
    pass

```

4.6 Method Resolution Order di Python

Method Resolution Order (MRO) adalah urutan pencarian metode pewarisan berganda dalam bahasa pemrograman Python. MRO menentukan urutan kelas mana yang pertama kali dicari ketika metode dipanggil pada objek yang diturunkan dari kelas yang memiliki beberapa superclass. Untuk mengetahui urutannya, gunakan perintah `help()`.

```
class A:
    def method(self):
        print("Ini adalah method dari kelas A")

class B:
    def method(self):
        print("Ini adalah method dari kelas B")

class C(A, B):
    pass

obj = C()
obj.method()
help(obj)
```

Ini adalah method dari kelas A
Help on C in module __main__ object:

```
class C(A, B)
| Method resolution order:
|   C
|   A
|   B
|   builtins.object
|
| Methods inherited from A:
|
| method(self)
|
| -----
| Data descriptors inherited from A:
|
| __dict__
|     dictionary for instance variables (if defined)
|
| __weakref__
|     list of weak references to the object (if defined)
```

PS C:\Users\asus\coding>