# RESUME PRAKTIKUM

# PEMOGRAMAN BERORIENTASI OBJEK


Disusun oleh:

Nama: Heni Artha Uli br Turnip

NIM: 121140080

Kelas: RB

**PROGRAM STUDI TEKNIK INFORMATIKA**

**INSTITUT TEKNOLOGI SUMATERA**

**LAMPUNG SELATAN**

**2023**

**Dasar Pemograman Phyton**

Python adalah bahasa pemrograman tingkat tinggi yang bersifat interpretatif dan mendukung pemrograman berorientasi objek, pemrograman fungsional, dan struktur data yang beragam. Python adalah bahasa yang populer karena mudah dipelajari, memiliki sintaks yang bersih dan mudah dibaca, serta memiliki banyak library dan modul yang siap pakai. Berikut adalah contoh codingan dasar dalam bahasa pemrograman Python:

- Menampilkan teks di layar

```
print("Hello, Heni!")
```

Output: Hello, Heni!

- Menghitung jumlah dua bilangan:

```
a = 10
b = 20
c = a + b
print(c)
```

Output: 30

- Menerima input dari pengguna:

```
nama = input("Masukkan nama Anda: ") print("Hai, " + nama + "!")
```

Output:

```
Masukkan nama Anda:  Heni
Hai,  Heni!
```

- Menampilkan bilangan ganjil dari 1 hingga 10

```
for i in range(1, 11):
    if i % 2 != 0:
        print(i)
```

Output: 1 3 5 7 9

Tipe data di Python seperti list, tuple, dictionary, dan set adalah kumpulan data yang berbeda. Setiap jenis tipe data ini digunakan untuk menyimpan data dalam bentuk yang berbeda, dan masing-masing memiliki karakteristik dan fungsi yang berbeda. Berikut adalah penjelasan dan contoh codingan dari masing-masing tipe data:

- **List**

List adalah tipe data yang paling umum digunakan di Python. List adalah kumpulan data yang terurut, dapat diubah, dan memungkinkan duplikat. Dalam Python, list ditulis dalam tanda kurung siku [].

Contoh penggunaan list:

```
#membuat list kosong
my_list = []

#membuat list dengan data
my_list = [1,2,3,'four','five']

#mengakses elemen dalam list
print(my_list[0]) #output: 1

#mengubah elemen dalam list
my_list[3] = 'fourth'

#menambahkan elemen ke dalam list
my_list.append(6)

#menghapus elemen dari list
del my_list[1]

#menghitung jumlah elemen dalam list
print(len(my_list)) #output: 5
```

- **Tuple**

Tuple adalah kumpulan data yang terurut dan tidak dapat diubah. Tuple ditulis dalam tanda kurung biasa ().

Contoh penggunaan tuple:

```
#membuat tuple kosong
my_tuple = ()

#membuat tuple dengan data
my_tuple = (1,2,3,'four','five')

#mengakses elemen dalam tuple
print(my_tuple[0]) #output: 1

#mencoba mengubah elemen dalam tuple (akan menghasilkan error)
my_tuple[3] = 'fourth'

#menghitung jumlah elemen dalam tuple
print(len(my_tuple)) #output: 5
```

- **Dictionary**

Dictionary adalah kumpulan data yang tidak terurut, dapat diubah, dan memungkinkan duplikat. Setiap elemen dalam dictionary terdiri dari pasangan kunci-nilai. Dictionary ditulis dalam tanda kurung kurawal {}.

Contoh penggunaan dictionary:

```
#membuat dictionary kosong
my_dict = {}

#membuat dictionary dengan data
my_dict = {'name': 'John', 'age': 25, 'city': 'New York'}

#mengakses nilai dengan kunci
print(my_dict['name']) #output: John

#mengubah nilai dengan kunci
my_dict['age'] = 30

#menambahkan elemen baru ke dalam dictionary
my_dict['occupation'] = 'Developer'

#menghapus elemen dari dictionary
del my_dict['city']

#menghitung jumlah elemen dalam dictionary
print(len(my_dict)) #output: 3
```

- **Set**

Set adalah kumpulan data yang tidak terurut dan tidak diindeks. Set tidak memungkinkan adanya duplikat. Set ditulis dalam tanda kurung kurawal {}.

Contoh penggunaan set:

```
#membuat set kosong
my_set = set()

#membuat set dengan data
my_set = {1,2,3,4,5}

#menambahkan elemen baru ke dalam set
my_set.add(6)

#menghapus elemen dari set
my_set.remove(5)
```

- **Class**

Class adalah sebuah blueprint atau template untuk membuat objek yang memiliki karakteristik yang sama. Class ini terdiri dari atribut (variabel) dan metode (fungsi) yang dapat diakses dan dimanipulasi oleh objek yang dibuat dari class tersebut.

Contoh codingan untuk membuat class pada Python:

```python
class Mahasiswa:
    def __init__(self, nama, nim):
        self.nama = nama
        self.nim = nim

    def tampilkan_nama(self):
        print("Nama: ", self.nama)

    def tampilkan_nim(self):
        print("NIM: ", self.nim)
```

Pada contoh codingan di atas, kita membuat class dengan nama **Mahasiswa**. Class ini memiliki atribut **nama** dan **nim**, serta metode **tampilkan_nama()** dan **tampilkan_nim()**. Metode **__init__()** adalah metode khusus pada Python yang digunakan untuk menginisialisasi atribut pada saat objek dibuat.

- **Atribut**

Atribut adalah variabel yang terdapat pada class dan merepresentasikan karakteristik atau data yang dimiliki oleh objek yang dibuat dari class tersebut. Atribut dapat berupa variabel dengan tipe data apapun.

Contoh codingan untuk membuat atribut pada class Mahasiswa:

```python
class Mahasiswa:
    def __init__(self, nama, nim):
        self.nama = nama
        self.nim = nim
```

Pada contoh codingan di atas, kita membuat dua atribut pada class **Mahasiswa**, yaitu **nama** dan **nim**.

- **Metode**

Metode adalah fungsi yang terdapat pada class dan dapat digunakan untuk memanipulasi atau mengakses atribut pada objek yang dibuat dari class tersebut. Metode ini dapat memiliki parameter atau tidak.

Contoh codingan untuk membuat metode pada class `Mahasiswa`:

```python
class Mahasiswa:
    def __init__(self, nama, nim):
        self.nama = nama
        self.nim = nim

    def tampilkan_nama(self):
        print("Nama: ", self.nama)

    def tampilkan_nim(self):
        print("NIM: ", self.nim)
```

Pada contoh codingan di atas, kita membuat dua metode pada class `Mahasiswa`, yaitu `tampilkan_nama()` dan `tampilkan_nim()`. Metode `tampilkan_nama()` digunakan untuk menampilkan nama mahasiswa, sedangkan metode `tampilkan_nim()` digunakan untuk menampilkan NIM mahasiswa.

- **Objek**

Objek adalah entitas yang memiliki sifat dan perilaku. Di Python, setiap nilai yang dibuat dianggap sebagai objek, yang dapat memiliki atribut (variabel) dan metode (fungsi). Contoh:

```python
#membuat objek kucing
class Kucing:
    def __init__(self, nama, umur):
        self.nama = nama
        self.umur = umur

    def bersuara(self):
        print("Meow")

kucing1 = Kucing("Tom", 3)
kucing1.bersuara() #output: Meow
```

- **Konstruktor**

Konstruktor adalah metode khusus yang dipanggil saat objek dibuat. Ini digunakan untuk menginisialisasi objek dengan nilai-nilai awal. Contoh:

```python
#membuat objek kucing
class Kucing:
    def __init__(self, nama, umur):
        self.nama = nama
        self.umur = umur

kucing1 = Kucing("Tom", 3)
print(kucing1.nama) #output: Tom
```

- **Destruktor**

Destruktor adalah metode khusus yang dipanggil saat objek dihapus dari memori. Ini digunakan untuk membersihkan sumber daya atau melakukan tindakan terakhir sebelum objek dihapus. Contoh:

```python
#membuat objek kucing
class Kucing:
    def __init__(self, nama, umur):
        self.nama = nama
        self.umur = umur

    def __del__(self):
        print("Objek dihapus")

kucing1 = Kucing("Tom", 3)
del kucing1 #output: Objek dihapus
```

- **Setter dan Getter**

Setter dan Getter adalah metode khusus yang digunakan untuk mengatur dan mendapatkan nilai atribut objek. Setter digunakan untuk mengatur nilai atribut dan Getter digunakan untuk mendapatkan nilai atribut. Contoh:

```python
#membuat objek kucing
class Kucing:
    def __init__(self, nama, umur):
        self.__nama = nama
        self.__umur = umur

    def setNama(self, nama):
        self.__nama = nama

    def getNama(self):
        return self.__nama

    def setUmur(self, umur):
        self.__umur = umur

    def getUmur(self):
        return self.__umur

kucing1 = Kucing("Tom", 3)
print(kucing1.getNama()) #output: Tom
kucing1.setNama("Jerry")
print(kucing1.getNama()) #output: Jerry
```

- **Decorator**

Decorator adalah fungsi yang digunakan untuk memodifikasi atau memperluas fungsi lain. Di Python, dekorator dapat digunakan untuk menambahkan fungsi tambahan ke fungsi atau metode kelas. Contoh:

```
def decor(func):
    def wrapper():
        print("Sebelum fungsi dieksekusi")
        func()
        print("Setelah fungsi dieksekusi")
    return wrapper

# Fungsi yang akan didekorasi
def kucing():
    print("Meong-meong!")

# Menggunakan decorator untuk menghias fungsi kucing
kucing = decor(kucing)

# Menjalankan fungsi kucing yang telah didekorasi
kucing()
```

- **ABSTRAKSI**

Abstraksi dalam OOP (Object-Oriented Programming) adalah proses menyembunyikan detail tertentu dari suatu objek dan hanya menampilkan informasi penting yang diperlukan untuk memenuhi tujuan tertentu. Dalam Python, abstraksi dapat dilakukan dengan menggunakan konsep kelas dan metode.

Contoh sederhana abstraksi dalam OOP dengan Python adalah sebagai berikut:

```
class Shape:
    def __init__(self):
        pass

    def area(self):
        pass

class Rectangle(Shape):
    def __init__(self, length, width):
        super().__init__()
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

class Circle(Shape):
    def __init__(self, radius):
        super().__init__()
        self.radius = radius

    def area(self):
        return 3.14 * self.radius * self.radius
```

Pada contoh di atas, kita membuat kelas **Shape** sebagai kelas abstrak yang memiliki metode **area()** yang belum diimplementasikan. Kemudian, kita membuat dua kelas baru yaitu **Rectangle** dan **Circle** yang merupakan turunan dari kelas **Shape**. Kedua kelas tersebut mengimplementasikan metode **area()** sesuai dengan bentuknya masing-masing.

Dengan demikian, penggunaan abstraksi dalam contoh di atas adalah kita menyembunyikan detail implementasi dari metode **area()** pada kelas **Shape**, sehingga pengguna hanya perlu memanggil metode **area()** pada objek **Rectangle** atau **Circle** untuk mendapatkan hasil yang diinginkan tanpa harus tahu cara implementasinya.

- **ENKAPSULASI**

Enkapsulasi adalah salah satu konsep dalam pemrograman berorientasi objek (OOP) yang memungkinkan data dan perilaku objek untuk disembunyikan dari objek lain, sehingga hanya dapat diakses melalui metode objek itu sendiri.

Dalam Python, enkapsulasi dapat dilakukan dengan menggunakan akses modifier (public, protected, dan private) pada atribut dan metode sebuah class.

Atribut dan metode yang didefinisikan sebagai public dapat diakses dari mana saja, baik dari dalam class itu sendiri maupun dari luar class.

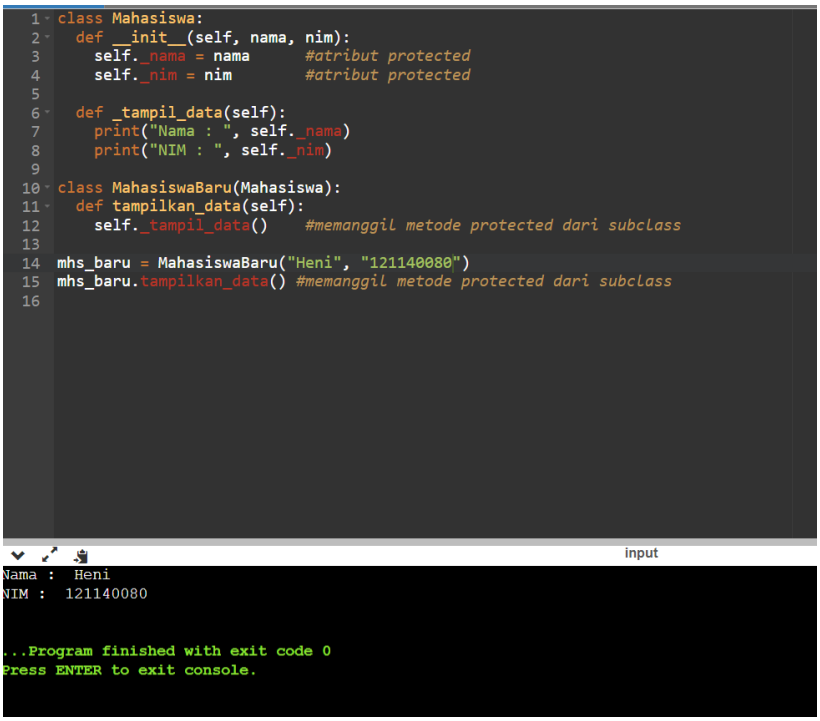
Contoh:

```
1  class Mahasiswa:
2    def __init__(self, nama, nim):
3      self.nama = nama      #atribut public
4      self.nim = nim        #atribut public
5
6    def tampil_data(self):
7      print("Nama : ", self.nama)
8      print("NIM : ", self.nim)
9
10 mhs = Mahasiswa("Heni", "12114000")
11 mhs.tampil_data()         #memanggil metode public
12
```

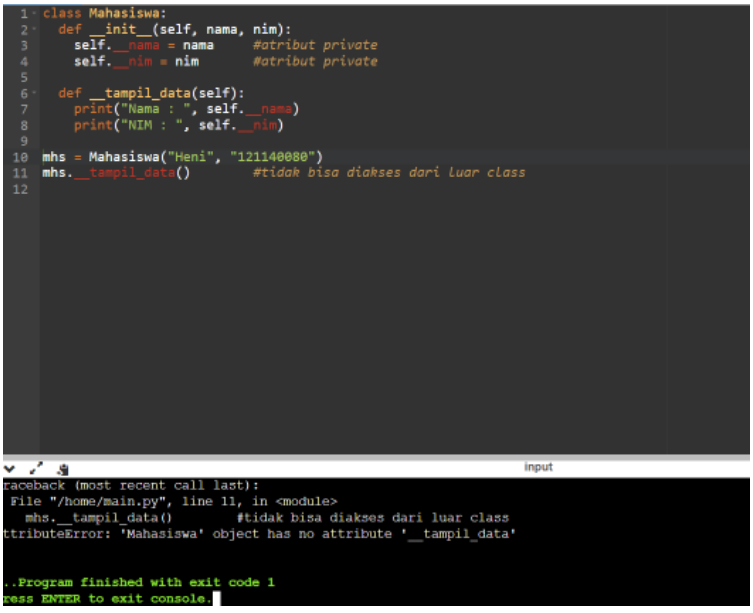
```
input
Nama :  Heni
NIM :  12114000


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

Atribut dan metode yang didefinisikan sebagai protected dapat diakses dari dalam class itu sendiri dan subclass, tetapi tidak dapat diakses dari luar class.

Contoh:




Atribut dan metode yang didefinisikan sebagai private hanya dapat diakses dari dalam class itu sendiri. Contoh:



- **Inheritance**

Inheritance dalam OOP (Object Oriented Programming) adalah konsep dimana sebuah class dapat mewarisi properti dan metode dari class yang lain. Class yang mewarisi properti dan metode disebut sebagai subclass atau turunan, sedangkan class yang properti dan metodenya diwarisi disebut sebagai superclass atau induk. Contoh penggunaan inheritance dalam Python adalah sebagai berikut:

```python
class Kendaraan:
    def __init__(self, nama, jumlah_roda):
        self.nama = nama
        self.jumlah_roda = jumlah_roda

    def info(self):
        print("Ini adalah kendaraan", self.nama, "dengan", self.jumlah_roda, "roda")

class Mobil(Kendaraan):
    def __init__(self, nama, jumlah_roda, merk):
        super().__init__(nama, jumlah_roda)
        self.merk = merk

    def info(self):
        print("Ini adalah mobil", self.merk, "dengan nama", self.nama, "dan memiliki", self.jumlah_roda, "roda")

mobil = Mobil("Avanza", 4, "Toyota")
mobil.info()
```

```
input
Ini adalah mobil Toyota dengan nama Avanza dan memiliki 4 roda


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

- **Polymorphism**

Polymorphism dalam pemrograman berorientasi objek (OOP) mengacu pada kemampuan objek untuk menerima bentuk atau perilaku yang berbeda, tergantung pada konteks di mana mereka digunakan. Dengan kata lain, objek yang sama dapat berperilaku dengan cara yang berbeda, tergantung pada bagaimana mereka digunakan.

Contoh sederhana dari konsep polymorphism adalah fungsi **len()**. Fungsi ini dapat digunakan untuk mengembalikan panjang dari berbagai tipe objek seperti string, list, tuple, dictionary, dan lain-lain. Walaupun tipe data objek yang berbeda, namun fungsi **len()** dapat diaplikasikan pada semua objek tersebut. Contoh kode:

```python
my_list = [1, 2, 3, 4, 5]
my_dict = {'a': 1, 'b': 2, 'c': 3}

print(len(my_list))  # Output: 5
print(len(my_dict))  # Output: 3
```

Dalam contoh di atas, fungsi **len()** menerima objek **my_list** dan **my_dict** dengan tipe data yang berbeda (list dan dictionary), namun tetap mengembalikan hasil yang benar-benar diinginkan (jumlah elemen dalam list/dictionary).

Contoh lain dari polymorphism dalam Python dapat dilihat dalam konsep "operator overloading". Ini mengacu pada kemampuan operator matematika seperti +, -, *, dan / untuk diterapkan pada objek yang berbeda, termasuk tipe data bawaan seperti int dan float, serta tipe data kustom yang ditentukan oleh pengguna. Berikut ini contoh operator overloading dengan operator + pada kelas **Book** dan **Magazine**:

```
class Book:
    def __init__(self, title, author, pages):
        self.title = title
        self.author = author
        self.pages = pages

    def __add__(self, other):
        return self.pages + other.pages


class Magazine:
    def __init__(self, title, publisher, pages):
        self.title = title
        self.publisher = publisher
        self.pages = pages

    def __add__(self, other):
        return self.pages + other.pages

book1 = Book("Belajar Python", "John Doe", 300)
magazine1 = Magazine("Python Today", "Python Inc.", 50)

print(book1 + magazine1)  # Output: 350
```

Dalam contoh di atas, kelas **Book** dan **Magazine** memiliki metode **__add__()** yang menentukan cara objek-objek tersebut dapat dijumlahkan dengan operator +. Dalam kasus ini, objek **book1** dan **magazine1** dapat dijumlahkan dengan cara menjumlahkan halaman dari masing-masing objek untuk menghasilkan total jumlah halaman yang dihasilkan dari kedua buku.

- **Override/Overriding**

Override atau overriding dalam pemrograman berorientasi objek (OOP) adalah sebuah konsep di mana sebuah method pada kelas turunan (subclass) mengganti atau menimpa (override) method dengan nama yang sama pada kelas induk (superclass). Dalam bahasa Python, konsep overriding dapat diterapkan pada method dalam kelas.

Contoh implementasi overriding pada Python adalah sebagai berikut:

```python
class Employee:
    def get_info(self):
        print("This is an employee.")

class Manager(Employee):
    def get_info(self):
        print("This is a manager.")

emp = Employee()
emp.get_info() # Output: This is an employee.

man = Manager()
man.get_info() # Output: This is a manager.
```

Pada contoh di atas, terdapat dua kelas yaitu **Employee** dan **Manager**. **Manager** merupakan turunan dari **Employee**. Kedua kelas tersebut memiliki method **get_info()**, namun pada kelas **Manager**, method tersebut di-override dan menampilkan output yang berbeda dengan method pada kelas **Employee**.

Ketika objek **emp** dari kelas **Employee** dipanggil, method **get_info()** pada kelas tersebut akan dipanggil dan menampilkan output "This is an employee.". Namun ketika objek **man** dari kelas **Manager** dipanggil, method **get_info()** pada kelas tersebut yang telah di-override akan dipanggil dan menampilkan output "This is a manager.".

- **Overloading**

Overloading pada OOP (Object-Oriented Programming) adalah kemampuan untuk mendefinisikan beberapa fungsi dengan nama yang sama tetapi dengan parameter yang berbeda. Dalam Python, overloading dapat dilakukan dengan menggunakan konsep fungsi dengan parameter default atau menggunakan operator khusus seperti "+" atau "-" pada tipe data yang berbeda. Contoh Overloading pada OOP dengan Python:

- Overloading pada method constructor:

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __init__(self, name):
        self.name = name
        self.age = None
```

Pada contoh di atas, terdapat dua method constructor dengan nama yang sama (init), namun dengan jumlah parameter yang berbeda. Jika objek Person dibuat dengan dua

parameter, maka method constructor pertama yang akan dieksekusi. Jika objek Person dibuat dengan satu parameter, maka method constructor kedua yang akan dieksekusi.

- Overloading pada operator aritmatika:

```
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

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

    def __sub__(self, other):
        return Vector(self.x - other.x, self.y - other.y)
```

Pada contoh di atas, operator "+" dan "-" dapat digunakan pada objek Vector. Jika operator "+" digunakan, maka method **add** yang akan dieksekusi. Jika operator "-" digunakan, maka method **sub** yang akan dieksekusi.

Dalam kedua contoh di atas, Python menggunakan konsep polymorphism untuk menentukan method mana yang harus dieksekusi, tergantung pada tipe dan jumlah parameter yang digunakan.