LAPORAN PRAKTIKUM PBO

TUGAS RESUME MODUL BAB I – IV

Disusun Oleh :

Nama : Rizki Esa Fadillah

NIM : 121140084

Kelas : PBO – RB

PROGRAM STUDI TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI SUMATERA

LAMPUNG SELATAN

2023

MODUL BAB I

Pengenalan Dasar Pemrograman Python

1. Pengenalan Bahasa Pemrograman Python

Python dibuat oleh Guido van Rossum pada akhir tahun 1980-an di Centrum Wiskunde & Informatica di Belanda. Python mendukung banyak paradigma pemrograman seperti object-oriented, functional, structured, dan juga didukung oleh banyak library(modul). Python bisa dipakai untuk:

- Pemrograman desktop ataupun mobile
- CLI
- GUI
- Web
- Otomatisasi
- Hacking
- IoT
- Robotika
- Dan lain-lain

2. Dasar Pemrograman Python

2.1. Sintaks Dasar

- **Statement**

  Statement adalah semua perintah yang bisa dieksekusi. Statement diakhiri dengan sebuah baris baru (newline), untuk stetament yang terdiri dari beberapa baris menggunakan backslash (\).

- **Baris dan Identasi**

  Kode yang berada di blok yang sama harus memiliki jumlah spasi yang sama. Untuk grouping python menggunakan tab (4 spasi).

2.2. Variabel dan Tipe Data Primitif

Variabel adalah penyimpanan suatu data/nilai. Setiap variable yang dibuat memerlukan sebuah tipe data. Berikut jenis-jenis tipe data, antara lain :

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 Program :

```python
Keterangan = True # bool
Angka = 25 # int
Desimal =15.75 # float
Kalimat = "Saya cinta PBO" # string

print(Keterangan)
print(Angka)
print(Desimal)
print(Kalimat)
```

Output :

```
PS D:\Belajar Python> & C:/Use
True
25
15.75
Saya cinta PBO
PS D:\Belajar Python>
```

2.3. Operator

- **Operator Aritmatika**

  Operator untuk operasi matematika, berikut jenis-jenis operator yang tersedia di python.

Contoh Program :

```python
Angka_1 = int(input("Bilangan ke-1 = "))
Angka_2 = int(input("Bilangan ke-2 = "))
Angka_3 = int(input("Bilangan ke-3 = "))
print("============")

print("Hasil penjumlahan Angka ke-1 dan ke-2 = ", Angka_1 + Angka_2) # Operasi Penjumlahan (+)
print("Hasil pengurangan Angka ke-1 dan ke-3 = ", Angka_1 - Angka_3) # Operasi Pengurangan (-)
print("Hasil perkalian Angka ke-3 dan ke-2 = ", Angka_3 * Angka_2) # Operasi Perkalian (*)
print("Hasil pembagian Angka ke-1 dan ke-2 = ", Angka_1 / Angka_2) # Operasi Pembagian (/)
print("Hasil pemangkatan Angka ke-1 dan ke-3 = ", Angka_1 ** Angka_3) # Operasi Pemangkatan (**)
print("Hasil pemabagian bulat Angka ke-2 dan ke-3 = ", Angka_2 // Angka_3) # Operasi Pembagian Bulat (//)
print("Hasil Modulus Angka ke-1 dan ke-2 = ", Angka_1 % Angka_2) # Operasi Modulus (%)
```

Output :

```
Bilangan ke-1 = 5
Bilangan ke-2 = 8
Bilangan ke-3 = 14
============
Hasil penjumlahan Angka ke-1 dan ke-2 =  13
Hasil pengurangan Angka ke-1 dan ke-3 =  -9
Hasil perkalian Angka ke-3 dan ke-2 =  112
Hasil pembagian Angka ke-1 dan ke-2 =  0.625
Hasil pemangkatan Angka ke-1 dan ke-3 =  6103515625
Hasil pemabgian bulat Angka ke-2 dan ke-3 =  0
Hasil Modulus Angka ke-1 dan ke-2 =  5
PS D:\Belajar Python>
```

- **Operator Perbandingan**
  Operator yang digunakan untuk membandingkan 2 buah nilai.

  Contoh Program :

```
Angka_1 = int(input("Bilangan ke-1 = "))
Angka_2 = int(input("Bilangan ke-2 = "))
Angka_3 = int(input("Bilangan ke-3 = "))
print("=============")

print("Apakah Angka ke-1 > ke-2 = ", Angka_1 > Angka_2) # Operasi Perbandingan lebih besar dari (>)
print("Apakah Angka ke-1 < ke-3 = ", Angka_1 < Angka_3) # Operasi Perbandingan lebih kecil dari (<)
print("Apakah Angka ke-3 == ke-2 = ", Angka_3 == Angka_2) # Operasi Perbandingan sama dengan (==)
print("Apakah  Angka ke-1 != ke-2 = ", Angka_1 != Angka_2) # Operasi Perbandingan tidak sama dengan (!=)
print("Apakah Angka ke-1 >= ke-3 = ", Angka_1 >= Angka_3) # Operasi Perbandingan lebih besar sama dengan dari (>=)
print("Apakah Angka ke-2 <= ke-3 = ", Angka_2 <= Angka_3) # Operasi Perbandingan lebih kecil sama dengan dari (<=)
```

  Output :

```
PS D:\Belajar Python> & C:/Users/LENOVO/A
Bilangan ke-1 = 5
Bilangan ke-2 = 7
Bilangan ke-3 = 4
=============
Apakah Angka ke-1 > ke-2 =  False
Apakah Angka ke-1 < ke-3 =  False
Apakah Angka ke-3 == ke-2 =  False
Apakah Angka ke-1 != ke-2 =  True
Apakah Angka ke-1 >= ke-3 =  True
Apakah Angka ke-2 <= ke-3 =  False
PS D:\Belajar Python>
```

- **Operator Penugasan**
  Operator penugasan adalah operator yang digunakan untuk memberi nilai ke variabel.

  Contoh Program :

```
Angka = int(input("Bilangan input = "))
Nilai_1 = 5
Nilai_2 = 4
print("======================")

Angka += Nilai_1
print("Operator penugasan (+=), Angka += Nilai_1 = ", Angka) # Menambahkan operand dikanan dengan operand dikiri (a += b -> a = a+b)
Angka -= Nilai_2
print("Operator penugasan (-=), Angka -= Nilai_2 = ", Angka) # Mengurangkan operand dikanan dengan operand dikiri (a -= b -> a = a-b)
Angka *= Nilai_1
print("Operator penugasan (*=), Angka *= Nilai_1 = ", Angka) # Mengalikan operand dikanan dengan operand dikiri (a *= b -> a = a*b)
Angka /= Nilai_2
print("Operator penugasan (/=), Angka /= Nilai_2 = ", Angka) # Membagikan operand dikanan dengan operand dikiri (a /= b -> a = a*b)
Angka **= Nilai_1
print("Operator penugasan (**=), Angka **= Nilai_1 = ", Angka) # Memangkatkan operand dikanan dengan operand dikiri (a **= b -> a = a**b)
Angka //= Nilai_2
print("Operator penugasan (//=), Angka //= Nilai_2 = ", Angka) # Membagi secara bulat operand dikanan dengan operand dikiri (a //= b -> a = a//b)
Angka %= Nilai_1
print("Operator penugasan (%=), Angka %= Nilai_1 = ", Angka) # Melakukan operasi modulus operand dikanan dengan operand dikiri (a %= b -> a = a%b)
```

Output :

```
PS D:\Belajar Python> & C:/Users/LENOVO/AppData/Local/Programs/Python/
Bilangan input = 10
====================
Operator penugasan (+=), Angka += Nilai_1 =  15
Operator penugasan (-=), Angka -= Nilai_2 =  11
Operator penugasan (*=), Angka *= Nilai_1 =  55
Operator penugasan (/=), Angka /= Nilai_2 =  13.75
Operator penugasan (**=), Angka **= Nilai_1 =  491488.6474609375
Operator penugasan (//=), Angka //= Nilai_2 =  122872.0
Operator penugasan (%=), Angka %= Nilai_1 =  2.0
PS D:\Belajar Python>
```

Nilai variable Angka akan terus diperbarui nilainya hingga operand modulus.

- **Operator Logika**
  Operator logika adalah operator yang digunakan untuk melakukan operasi logika.

  Contoh Program :

```
A = 30
B = 15

# Operator Logika "and" akan bernilai True jika kedua operasi bernilai benar
Logika_1 = A and A > 20
print("Hasil Kondisi operand logika_1 = ", Logika_1)
# Operator Logika "or" akan bernilai True jika salah satu operasi bernilai benar
Logika_2 = B or A < 20
print("Hasil Kondisi operand logika_1 = ", Logika_2)
# Operator Logika "not" akan bernilai True jika operasi bernilai salah
Logika_3 = not A
print("Hasil Kondisi operand logika_1 = ", Logika_3)
```

  Output :

```
Hasil Kondisi operand logika_1 =  True
Hasil Kondisi operand logika_1 =  15
Hasil Kondisi operand logika_1 =  False
PS D:\Belajar Python>
```

- **Operator Bitwise**
  Operator bitwise adalah operator yang melakukan operasi bit terhadap operand.

  Contoh program :

```python
Angka_1 = int(input("Angka ke-1 = "))
Angka_2 = int(input("Angka ke-2 = "))
print("======================")

Bitwise_and = Angka_1 & Angka_2
print("Hasil Bitwise_and = ", Bitwise_and) # Operasi Bitwise AND
Bitwise_or = Angka_1 | Angka_2
print("Hasil Bitwise_or = ", Bitwise_or) # Operasi Bitwise OR
Bitwise_not = ~Angka_2
print("Hasil Bitwise_not = ", Bitwise_not) # Operasi Bitwise NOT
Bitwise_xor = Angka_1 ^ Angka_2
print("Hasil Bitwise_xor = ", Bitwise_xor) # Operasi Bitwise XOR
Bitwise_right_shift = Angka_1 >> 2
print("Hasil Bitwise_right_shift = ", Bitwise_right_shift) # Operasi Bitwise Right Shift
Bitwise_left_shift = Angka_2 << 2
print("Hasil Bitwise_left_shift = ", Bitwise_left_shift) # Operasi Bitwise Left Shift
```

Output :

```
PS D:\Belajar Python> & C:/Users/LE
Angka ke-1 = 10
Angka ke-2 = 4
======================
Hasil Bitwise_and =  0
Hasil Bitwise_or =  14
Hasil Bitwise_not =  -5
Hasil Bitwise_xor =  14
Hasil Bitwise_right_shift =  2
Hasil Bitwise_left_shift =  16
PS D:\Belajar Python>
```

- **Operator Identitas**

  Operator identitas adalah operator yang memeriksa apakah dua buah nilai (atau variabel) berada pada lokasi memori yang sama.

Contoh Program :

```python
A = "Anak Baik"
B = "Anak Nakal"
C = "Anak Baik"

# Operator identitas "is" akan bernilai true jika kedua variabel bernilai sama
print("A is C : ", A is C)
# Operator identitas "is Not" akan bernilai true jika kedua variabel berbeda nilai
print("A is Not C : ", A is not C)
```

Output :

```
PS D:\Belajar Python> & C:/U
A is C :  True
A is Not C :  False
PS D:\Belajar Python>
```

- **Operator Keanggotaan**

  Operator keanggotaan adalah operator yang digunakan untuk memeriksa apakah suatu nilai/variabel merupakan suatu data dalam (string, list, tuple, set, dan dictionary).

Contoh Program :

```
A = "Anak Baik"
B = "Anak Nakal"
C = ("Anak Baik", "Anak Aneh")

# Operator Keanggotaan "in" akan bernilai true jika nilai/variabel ditemukan dalam data
print("A in C : ", A in C)
# Operator Keanggotaan "Not in" akan bernilai true jika nilai/variabel tidak ditemukan
print("A Not In C : ", A not in C)
```

Output :

```
PS D:\Belajar Python> & C:/Users/
A in C :  True
A Not In C :  False
PS D:\Belajar Python>
```

2.4. 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

2.5. Percabangan

Pemrograman python terdapat beberapa percabangan, percabangan adalah proses penentuan keputusan. Terdapat 3 jenis Percabangan If, If-Else, dan If-Else-If.

Contoh Program :

```
Harga = int(input("Masukkan Harga = "))

if Harga >= 100000:
    print("Harga " + str(Harga) + " masih tergolong mahal")
elif Harga < 100000 and Harga >= 50000:
    print("Harga " + str(Harga) + " masih tergolong terjangkau")
else :
    print("Harga " + str(Harga) + " masih tergolong sangat terjangkau/murah")
```

Output :

```
PS D:\Belajar Python> & C:/Users/LENOVO/Ap
Masukkan Harga = 75000
Harga 75000 masih tergolong terjangkau
PS D:\Belajar Python>
```

- **Nested if**

  Merupakan percabangan IF dengan struktur yang lebih kompleks**.** Dimana didalam sebuah pernyataan IF terdapat pernyataan IF lainnya.

  Contoh Program :

```
Harga = int(input("Masukkan Harga = "))

if Harga <= 50000:
    if Harga != 0:
        print("Harga " + str(Harga) + " masih tergolong sangat terjangkau/murah")
    elif Harga == 0:
        print("Tidak berharga")
```

  Output :

```
PS D:\Belajar Python> & C:/Users/LENOVO/AppData/Local/P
Masukkan Harga = 25000
Harga 25000 masih tergolong sangat terjangkau/murah
PS D:\Belajar Python> & C:/Users/LENOVO/AppData/Local/P
Masukkan Harga = 0
Tidak berharga
PS D:\Belajar Python>
```

2.6. Perulangan

Perulangan adalah suatu proses untunk mengulang program sebanyak n-kali. Ada 2 jenis perulangan antara, lain:

- **Perulangan For**

  Biasanya digunakan untuk iterasi pada urutan berupa list, tuple, dan string.

  Contoh Program :

```
Daftar_Barang = ["TV, Motor, Lemari, HP, Jam Tangan"] # list
Harga_Barang = (2150000, 22500000, 3250000, 1900000, 150000) # tuple
Total_Harga = 0

print("Daftar barang yang dibeli : ")
for i in Daftar_Barang: # perulangan for pada list
    print ("=> ", i)

for i in Harga_Barang: # perulangan for pada tuple
    Total_Harga += i

print("Total harga yang harus dibayar = Rp.",Total_Harga)
```

  Output :

```
PS D:\Belajar Python> & C:/Users/LENOVO/AppData
Daftar barang yang dibeli :
=>  TV, Motor, Lemari, HP, Jam Tangan
Total harga yang harus dibayar = Rp. 29950000
PS D:\Belajar Python>
```

  - Sintaks Range
    - Satu Parameter, for I in range(x): => Perulangan dilakukan dari indeks 0 – x
    - Dua Parameter, for I in range(x, y): => Perulangan dilakukan dari indeks x hingga kurang dari indeks y.
    - Tiga Parameter, for I in range(x, y, z): => Perulangan dilakukan dari indeks x higga kuranf dari indeks y, dengan indeks bertambah/berkurang sebanyak z setiap perulangannya.

- **Perulangan While**

  Perulangan yang akan berjalan ketika kondisi tertentu terpenuhi.

  Contoh Program :

```
print("Masukkan angka : ")
Angka = int(input())
Jumlah = 0

while (Angka != 0):
    Jumlah += Angka
    Angka = int(input())

print("Jumlah daftar angka = ", Jumlah)
```

Output :

```
Masukkan angka :
11
25
34
3
7
-12
9
-33
0
Jumlah daftar angka =  44
PS D:\Belajar Python>
```

2.7. Fungsi

Fungsi merupakan sintaks yang digunakan untuk mengeksekusi suatu blok kode tanpa harus menulis berulang-ulang.

Contoh Program :

```python
def izin_SIM(Nama, Umur):
    if Umur >= 17:
        return f"{Nama} layak memiliki SIM "
    else :
        return f"{Nama} tidak layak memiliki SIM"

print("Data diri calon penerima SIM : ")
Nama = input("Nama : ")
Umur = int(input("Umur : "))

print(izin_SIM(Nama, Umur))
```

Output :

```
PS D:\Belajar Python> & C:/Users/LENOVO/
Data diri calon penerima SIM :
Nama : Rizki
Umur : 20
Rizki layak memiliki SIM
PS D:\Belajar Python> & C:/Users/LENOVO/
Data diri calon penerima SIM :
Nama : Esa
Umur : 16
Esa tidak layak memiliki SIM
PS D:\Belajar Python>
```

MODUL BAB II

Objek dan Kelas dalam Python

1. Kelas (Class)

Class merupakan cetakan dari objek/instance yang ingin dibuat agar dapat didesain dengan bebas. Didalam kelas mendefinisikan atribut/property dan metode untuk objek. Sebuah class memerlukan intansiasi untuk dapat mengimplementasikan objek didalamnya. Untuk membuat sebuah kelas perlu sebuah methode konstruktor (__init__) untuk membuat objek.

1.1. Atribut/Property

Ada 2 jenis atribut, yaitu atribut kelas dan atribut objek. Atribut kelas merupakan sifat yang dimiliki oleh sebuah kelas dan setiap objek. Atribut objek merupakan atribut dari masing-masing objek.

1.2. Method

Method adalah suatu fungsi yang terdapat didalam kelas. Methode dapat diibaratkan sebagai sebuah aktivitas/proses yang dapat dilakukan oleh sebuah objek.

Contoh Program class beserta atribut dan methodnya :

```
class Rizki_Esa_Fadillah:
    #methode __init__ merupakan methode konstruktor yang didalamnya terdapat atribut variabel data mahasiswa
    def __init__(self, Nama, Nim, Kelas_PBO_Siakad, Nilai_Matkul, Prodi):
        self.Nama = Nama
        self.Nim = Nim
        self.Kelas_PBO_siakad = Kelas_PBO_Siakad
        self.Nilai_Matkul = Nilai_Matkul
        self.Prodi = Prodi
    #methode print_data untuk mencetak/mengoutputkan data mahasiswa, dimana data di peroleh dari pemanggilan class pada baris
    def print_data(self):
        print("Data Mahasiswa :\nNama = "+self.Nama+"\nNIM = "+self.Nim+"\nProdi = "+self.Prodi+"\nKelas = "+self.Kelas_PBO_siakad+"\nNilai = "+self.Nilai_Matkul)
#baris 18, merupakan sebuah instance class dengan variabel mahasiswa, dimana isinilah pemanggilan class datanya akan di teruskan ke methode konstruktor(__init__)
Mahasiswa = Rizki_Esa_Fadillah("Rizki Esa Fadillah", 121140084, "RB", "A", "Teknik Informatika")
#baris 20 merupakan proses pemanggilan methode untuk mencetak data
Mahasiswa.print_data()
```

Output :

```
PS D:\Belajar Python> & C:/Users/LENO
Data Mahasiswa :
Nama = Rizki Esa Fadillah
NIM = 121140084
Prodi = Teknik Informatika
Kelas = RB
Nilai = A
PS D:\Belajar Python>
```

2. Objek

Objek berfungsi sebagai pengganti pemanggilan kelas, dan hanya dapat digunakan untuk satu kelas saja. Objek bermanfaat untuk mempermudah pemanggilan kelas terutama dengan nama yang terlalu panjang.

```
Mahasiswa = Rizki_Esa_Fadillah("Rizki Esa Fadillah", 121140084, "RB", "A", "Teknik Informatika")
```

3. Magic Method

Magic method adalah metode yang diawali dan diakhiri dengan double underscore (dunder). Tujuannya untuk mengubah sifat bawaan dari suatu objek. Magic methode ada disetiap jenis objek dan variable. Berikut magic methode bawaan yang terdapat pada python antara lain:

```
>>> dir(int)
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '__delattr__',
'__dir__', '__divmod__', '__doc__', '__eq__', '__float__', '__floor__', '__floordiv__',
'__format__', '__ge__', '__getattribute__', '__getnewargs__', '__gt__', '__hash__',
'__index__', '__init__', '__init_subclass__', '__int__', '__invert__', '__le__',
'__lshift__', '__lt__', '__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__or__',
'__pos__', '__pow__', '__radd__', '__rand__', '__rdivmod__', '__reduce__', '__reduce_ex__',
'__repr__', '__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__', '__ror__', '__round__',
'__rpow__', '__rrshift__', '__rshift__', '__rsub__', '__rtruediv__', '__rxor__',
'__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__truediv__',
'__trunc__', '__xor__', 'bit_length', 'conjugate', 'denominator', 'from_bytes', 'imag',
'numerator', 'real', 'to_bytes']
```

Contoh Program Magic Method :

```
1  class Selisih_Harga:
2      def __init__(self, Beli):
3          self.Beli = Beli
4
5      def __sub__(self, other):
6          return self.Beli - other.Beli
7
8  A = Selisih_Harga(50000)
9  B = Selisih_Harga(35000)
10 print(A - B)
```

Output:

```
PS D:\Belajar Python> & C:/Users/
15000
PS D:\Belajar Python>
```

4. Konstruktor

Konstruktor adalah method yang pasti dijalankan secara otomatis pada saat objek dibuat untuk mewakili kelas tersebut. Konstruktor juga dapat menerima argument yang diberikan ketika objek dibuat, dan selanjutnya akan diproses dalam kelas.

Contoh Program Kontruktor :

```python
class Rizki_Esa_Fadillah:
    #method __init__ merupakan method konstruktor yang didalamnya terdapat atribut variabel data mahasiswa
    def __init__(self, Nama, Nim, Kelas_PBO_Siakad, Nilai_Matkul, Prodi):
        self.Nama = Nama
        self.Nim = Nim
        self.Kelas_PBO_siakad = Kelas_PBO_Siakad
        self.Nilai_Matkul = Nilai_Matkul
        self.Prodi = Prodi
    #method print_data untuk mencetak/menggabungkan data mahasiswa, dimana data di peroleh dari pemanggilan class pada baris
    def print_data(self):
        print("Data Mahasiswa \nNama = {self.Nama} \nNIM = {self.Nim}\nProdi = {self.Prodi}\nKelas = {self.Kelas_PBO_siakad}\nNilai = {self.Nilai_Matkul} ")

Mahasiswa = Rizki_Esa_Fadillah("Rizki Esa Fadillah", 121140084, "RB", "A", "Teknik Informatika")
Mahasiswa.print_data()
```

Output :

```
PS D:\Belajar Python> & C:/Users/LENO
Data Mahasiswa :
Nama = Rizki Esa Fadillah
NIM = 121140084
Prodi = Teknik Informatika
Kelas = RB
Nilai = A
PS D:\Belajar Python>
```

5. Destruktor

Destruktor adalah fungsi yang dipanggil ketika user menghapus objek. Fungsi ini bekerja secara otomatis, sehingga tidak memerlukan pemanggilan. Tujuan dibuat destructor adalah untuk melakukan bersih-bersih terakhir sebelum objek benar-benar dihapus.

Contoh Program Destructor :

```python
class Selisih_Harga:
    def __init__(self, Beli):
        self.Beli = Beli

    def __del__(self):
        print ("Objek {self.Beli} dihapus")

A = Selisih_Harga(50000)
B = Selisih_Harga(35000)
```

Output :

```
PS D:\Belajar Python> & C:/Users/LENO
Objek {self.Beli} dihapus
Objek {self.Beli} dihapus
PS D:\Belajar Python>
```

6. 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. Getter digunakan untuk mengambil nilai.

Contoh Program Setter dan Getter :

```python
class Keahlian:
    def __init__(self, ahli = ""):
        self._ahli = ahli

    # getter methode
    def get_ahli(self):
        return self._ahli

    # setter methode
    def set_ahli(self, a):
        self._ahli = a

Teknisi = Keahlian()
# set
Teknisi.set_ahli("Informatika")
# get
print(Teknisi.get_ahli())
print(Teknisi._ahli)
```

Output :

```
PS D:\Belajar Python> & C:/Users/L
Informatika
Informatika
PS D:\Belajar Python>
```

7. Decorator

Property decorator memiliki manfaat yang sama denga setter dan getter, nampun decorator tidak perlu membuat fungsi baru, melainkan menggunkan 1 buah variable.

Contoh Program Decorator :

```python
class Harta:
    def __init__(self, jumlah = 0):
        self._jumlah = jumlah

    @property
    def jumlah(self):
        return self._jumlah

    @jumlah.setter
    def jumlah(self, a):
        self._jumlah = a

Rizki = Harta()

print(Rizki.jumlah)
Rizki.jumlah += 15000000
print(Rizki.jumlah)
```

Output :

```
PS D:\Belajar Python> & C:/Use
0
15000000
PS D:\Belajar Python>
```

MODUL BAB III

Abstraksi dan Enkapsulasi

1. Abstraksi

Abstrsksi merupakan salah satu konsep OOP dimana program hanya memperlihatkan atribut yang diperlukan dan menyembunyikan detail-detail yang tidak perlu dari user agar program menjadi ringkas. Sehingga user hanya mengetahui apa yang dilakukan objek, tapi tidak tau apa yang terjadi dibaliknya.

Contoh Program Abstraksi :

```python
# class abstrak (parent)
from abc import ABC, abstractclassmethod

class Bangun_Datar(ABC):
    @abstractclassmethod
    def Jumlah_Sisi(self):
        pass

# class anak
class Persegi(Bangun_Datar):
    def Jumlah_Sisi(self):
        print("Persegi memiliki 4 Sisi")

class Segitiga(Bangun_Datar):
    def Jumlah_Sisi(self):
        print("Segitiga memiliki 3 Sisi")

class Lingkaran(Bangun_Datar):
    def Jumlah_Sisi(self):
        print("Lingkaran memiliki 1 Sisi")

# Eksekusis
A = Segitiga()
A.Jumlah_Sisi()

B = Persegi()
B.Jumlah_Sisi()
```

Output :

```
PS D:\Belajar Python> & C:/User
Segitiga memiliki 3 Sisi
Persegi memiliki 4 Sisi
Lingkaran memiliki 1 Sisi
```

2. Enkapsulasi

Enkapsulasi adalah metode yang digunakan untuk menyembunykan atribut suatu entitas untuk melindungi informasi dari luar agar elemen penting yang terdapat dalam kelas dapat terjaga. Terdapat 4 cara untuk melindungi atribut tersebut, antara lain:

2.1. Public Access Modifier

Public access modifier merupakan cara mendeklarasikan variable atau obejk secara umum (default). Contoh self.nama = nama

2.2. Protected Access Modifier

Protected access modifier merupakan cara mendeklarasikan variable atau method dengan menambahkan 1 underscore (_) agar hanya dapat diakses oleh kelas turunannya. Contoh self._warna = warna

2.3. Private Access Modifier

Private access modifier merupakan cara mendeklarasikan variable atau method dengan menambahkan 2 underscore(__) agar hanya dapat diakses dalam kelas itu sendiri. Metode access ini sangat sesuai untuk enkapsulasi karena memberikan perlindungan yang paling aman. Contoh self.__merk = merk

2.4. Setter dan Getter

Seperti penjelasan di bab sebelumnya, cara ini dapat digunakan dalam enkapsulasi dikarenakan property dengan access modifier privat hanya dapat diakses dari dalam kelas, sehingga kita mengaksesnya dengan metode ini.

Contoh Program Enkapsulasi :

```
class Handphone:
    jumlah_Handphone = 0

    def __init__(self, merk, Harga, warna):
        #atribut public
        self.merk = merk
        # atribut protected
        self._warna = warna
        # atribut private
        self.__Harga = Harga
        Handphone.jumlah_Handphone += 1

    def info_Handphone(self):
        print("Merk:", self.merk)
        print("Harga:", self.__Harga)
        print("Warna:", self._warna)

    # method public
    def ubah_merk(self, merk_baru):
        self.merk = merk_baru

    # method protected
    def _ubah_warna(self, warna_baru):
        self._warna = warna_baru

```

```
    # method private
    def __ubah_Harga(self, Harga_baru):
        self.__Harga = Harga_baru

    def _Tampilkan_Harga(self):
        print("Harga Handphone : ", self.__Harga)

Handphone1 = Handphone("Samsung", 7500000, "hitam")
Handphone2 = Handphone("Apple", 20500000, "putih")

# program untuk menggunakan atribut public, protect, dan private
print("Pilih merk ponsel yang akan diubah warna dan harganya : \n1). Samsung \n2). Apple")
print(f"Jumlah Handphone :  {Handphone.jumlah_Handphone}\n")
pilih = int(input("Pilih = "))
if pilih == 1:
    # akses atribut public
    print(f"Merk Handphone : {Handphone1.merk}")
    # akses atribut protected
    warna= str(input(f"Warna Handphone saat ini {Handphone1._warna} dibah menjadi warna : "))
    Handphone1._ubah_warna(warna)
    # tidak terjadi error karena atribut protect masih dapat diakses oleh class turunannya
    print(f"Warna Handphone saat ini : {Handphone1._warna}")
    # akses atribut private (Coba akses atribut privat)
    ##Handphone1.__ubah_Harga(harga) menghasilkan AttributeError karena berada diluar class
    Handphone1._Tampilkan_Harga()
    # Handphone1._Tampilkan_Harga() bisa digunakan karena methode yang bersifat protect
    # (_Tampilkan_harga) agar dapat diakses oleh class turunannya dan dalam methode
    # tersebut terdapat atribut privat (__harga) yang telah diakses dalam class handphone
    # sebelumnya sehingga tidak menyebabkan eror

elif pilih == 2:
    print(f"Merk Handphone : {Handphone2.merk}")
    warna= str(input(f"Warna Handphone saat ini {Handphone2._warna} dibah menjadi warna : "))
    Handphone2._ubah_warna(warna)
    print(f"Warna Handphone saat ini : {Handphone2._warna}")
    Handphone2._Tampilkan_Harga()
```

Output :

```
PS D:\Belajar Python> & C:/Users/LENOVO/AppData/Local/Programs/
Pilih merk ponsel yang akan diubah warna dan harganya :
1). Samsung
2). Apple
Jumlah Handphone :  2

Pilih = 1
Merk Handphone : Samsung
Warna Handphone saat ini hitam dibah menjadi warna : putih
Warna Handphone saat ini : putih
Harga Handphone :  7500000
PS D:\Belajar Python>
```

Object

1. **Membuat Instance Objek**

   Instance objek menggunakan nama dari class dan argument yang diterima oleh metode init yang terbungkus dalam kurung ().

   Contoh :

```
Mahasiswa = Rizki_Esa_Fadillah("Rizki Esa Fadillah", 121140084, "Teknik Informatika")
```

2. **Mengakses Atribut Objek**

   Mengakses atribut dari objek dapat menggunakan operator titik (.).

   Contoh :

```
Mahasiswa.print_data()
```

3. **Memodifikasi Atribut Objek**

   Objek yang telah dibuat dapat dimodifikasi sesuai keperluan, seperti:

   - getattr(obj, name[, default]) – Mengakses atribut dari objek.

```
getattr(maha1,'semester')
```

   - hasattr(obj, name) – Memeriksa apakah suatu objek memiliki atribut tertentu.

```
hasattr(maha2,'nama')
```

   - setattr(obj, name, value) – Mengatur nilai atribut. Jika ternyata atribut tidak ada, maka atribut tersebut akan dibuat.

```
setattr(maha3,'nama','Taku')
```

   - delattr(obj, name) – Menghapus atribut dari suatu objek.

```
delattr(maha3,'semester')
```

MODUL BAB IV

Pewarisan dan Polimorfisme

1. Inheritance (Pewarisan)

Inheritance merupakan salah satu OOP dimana program dapat menurunkan kelas yang saling berbagi atribut dan metode.

1.1. Inheritance Identik

Inheritance identic merupakan pewarisan yang menambahkan constructor pada class anak, agar class child memiliki constructornya sendiri tanpa menghilangkan contructor pada class parentnya. Metode ini ditandai dengan Kata kunci super().

1.2. Menambah Karaketristik pada Child Class

Penambahan karakteristik pada child class dapat dilakukan dengan menambahkan beberapa atribut atau mrthod agar tidak identic dengan parent class.

Contoh Program Inheritance :

```python
class Rumah:
    def __init__(self, biaya_hidup, jumlah_penghuni):
        self.biaya_hidup = biaya_hidup
        self.jumlah_penghuni = jumlah_penghuni

    def Hitung_Pengeluaran(self):
        return self.biaya_hidup * self.jumlah_penghuni

class Toko(Rumah):
    def __init__(self, pendapatan, jumlah_hari):
        self.pendapatan = pendapatan
        self.jumlah_hari = jumlah_hari

    def Hitung_Pendapatan(self):
        return self.pendapatan * self.jumlah_hari

class Ruko(Toko):
    def __init__(self, biaya_hidup, jumlah_penghuni, pendapatan, jumlah_hari):
        Rumah.__init__(self, biaya_hidup, jumlah_penghuni)
        Toko.__init__(self, pendapatan, jumlah_hari)

    def Hitung_Laba_Bersih(self):
        return f"Ruko bulan ini memiliki laba bersih senilai Rp{self.Hitung_Pendapatan() - self.Hitung_Pengeluaran()}"

Herman = Ruko(2000000, 3, 3000000, 22)
print(Herman.Hitung_Pengeluaran())
print(Herman.Hitung_Pendapatan())
print(Herman.Hitung_Laba_Bersih())
```

Output :

```
PS D:\Belajar Python> & C:/Users/LENOVO/AppData/Local/Programs
6000000
66000000
Ruko bulan ini memiliki laba bersih senilai Rp60000000
PS D:\Belajar Python>
```

2. Polymorphism

Polymorphism merupakan salah satu bentuk OOP yang memungkinkan program menggunakan suatu interface yang sama untuk memerintah objek melakukan aksi yang sama namun secara proses berbeda.

Contoh Program Polymorphisme :

```python
class Hewan:
    def __init__(self, nama, usia):
        self.nama = nama
        self.usia = usia
    def __str__(self): #Polymorphism
        return self.nama
    def suara(self):
        pass

class Kucing(Hewan):
    def __init__(self, nama, usia):
        super().__init__(nama, usia)
    def info(self):
        return f"Saya Seekor Kucing"
    def suara(self):
        return "Meow"

a = Kucing("Sipus", 3)
print(a) # Pemanggilan konsep polymorphism
print(a.info())
print(a.suara())
```

Output :

```
PS D:\Belajar Python> & C:/Users/LE
Sipus
Saya Seekor Kucing
Meow
PS D:\Belajar Python>
```

3. Override/Overriding

Suatu metode yang ada pada parent class dengan mendefinisikan kembali method dengan nama yang sama pada child class. Sehingga metode yang dijalankan adalah metode pada class child.

Contoh Program :

```
class Hewan:
    def __init__(self, nama, usia):
        self.nama = nama
        self.usia = usia
    def info(self): #Overriding
        return f"Saya Hewan bernama {self.nama}"

class Kucing(Hewan):
    def __init__(self, nama, usia):
        super().__init__(nama, usia)
    def info(self): #Overriding
        return f"Saya Seekor Kucing bernama {self.nama}"
    def suara(self):
        return "Meow"

a = Kucing("Sipus", 3)
print(a.info())
```

Output :

```
PS D:\Belajar Python> & C:/Users/L
Saya Seekor Kucing bernama Sipus
PS D:\Belajar Python>
```

4. Overloading

Metode ini mengizinkan sebuah class untuk memiliki sekumpulan fungsi dengan nama yang sama tapi argument yang berbeda. Metode ini sangat tricky karena python tidak mengizinkan deklarasi fungsi dengan nama yang sama.

Contoh Program :

```
class Anjing:
    def __init__(self, nama, usia):
        self.nama = nama
        self.usia = usia
    def suara(self): # Fungsi suara Anjing
        print("Suara Anjing Guk Guk")

class Kucing:
    def __init__(self, nama, usia):
        self.nama = nama
        self.usia = usia
    def suara(self): # Fungsi suara Kucing
        print("Suara Kucing Meow")

# Trik untuk memanggil Fungsi suara Anjing dan Kucing agar tidak error
def Panggil(other):
    other.suara()

a = Kucing("Sipus", 3)
b = Anjing("Doggy", 5)
Panggil(a)
Panggil(b)
```

Output :

```
PS D:\Belajar Python> & C:/Users/LE
Suara Kucing Meow
Suara Anjing Guk Guk
PS D:\Belajar Python>
```

5. Multiple Inheritance

Inheritance memungkinkan untuk situasi dimana class anak memiliki warisan dari 2 atau lebih class induknya.

```
class Base1:
    pass

class Base2:
    pass

class MultiDerived(Base1, Base2):
    pass
```

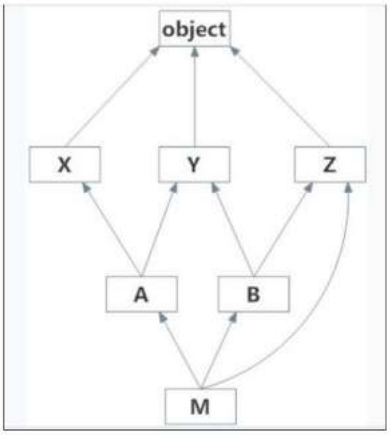
6. Methode Resolution Order di Python (MRO)

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. Proses pencarian dari class objek – super class, lalu jika ada banyak super class pencarian dilakukan dari class paling kiri – kanan.

Demonstrasi MRO :

```
# Demonstration of MRO

class X:
    pass


class Y:
    pass


class Z:
    pass


class A(X, Y):
    pass


class B(Y, Z):
    pass


class M(B, A, Z):
    pass
```

Visualisasi Diagram :



7. Dynamic Cast (Type Conversion)

Dynamic cast adalah proses mengubah nilai dari satu tipe data lainnya seperti dari string ke int atau sebaliknya.

7.1. Implisit

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

Contoh Program :

```
Angka_int = 125
Angka_float = 1.25

Jumlah = Angka_int * Angka_float

print("Tipe data angka_int = ", type(Angka_int))
print("Tipe data angka_float = ", type(Angka_float))
print("Jumlah Angka_int * Angka_float = ", Jumlah)
# Tipe data akan berubah karena eksekusi program itu sendiri
print("Tipe data jumlah angka = ", type(Jumlah))
```

Output :

```
PS D:\Belajar Python> & C:/Users/LENOVO/AppDa
Tipe data angka_int =  <class 'int'>
Tipe data angka_float =  <class 'float'>
Jumlah Angka_int * Angka_float =  156.25
Tipe data jumlah angka =  <class 'float'>
PS D:\Belajar Python>
```

7.2. 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 program :

```
Angka_1 = 125
Angka_2 = "250"

print("Tipe data angka_int = ", type(Angka_1))
print("Tipe data angka_float = ", type(Angka_2))

Angka_2 = int(Angka_2)

print("Tipe data angka_int = ", type(Angka_1))
print("Tipe data angka_float = ", type(Angka_2))

print("Jumlah Angka_1 * Angka_2 = ", Angka_1 + Angka_2)
print("Tipe data jumlah angka = ", type(Angka_1 + Angka_2))
```

Output :

```
PS D:\Belajar Python> & C:/Users/LENOVO/AppDat
Tipe data angka_int =  <class 'int'>
Tipe data angka_float =  <class 'str'>
Tipe data angka_int =  <class 'int'>
Tipe data angka_float =  <class 'int'>
Jumlah Angka_1 * Angka_2 =  375
Tipe data jumlah angka =  <class 'int'>
PS D:\Belajar Python>
```

8. Casting

8.1. Downcasting

Parent class mengakses atribut yang ada pada kelas bawah (child class).

Contoh program ;

```python
class Karakter:
    def __init__(self, nama, role):
        self.nama = nama
        self.role = role

    def info_karakter(self):
        print(f"{self.nama} merupakan seorang {self.role} dengan rank {self.rank}")
        # Pada self.rank diambil dari atribut pada class child

class Pahlawan(Karakter):
    def __init__(self, nama, role, rank):
        super().__init__(nama, role)
        self.rank = rank

player_1 = Pahlawan("Tanjiro", "Pendekar pedang dengan jurus teknik air", "A => Kuat")
player_1.info_karakter()
```

Output :

```
PS D:\Belajar Python> & C:/Users/LENOVO/AppData/Local/Programs/Python/Python311/python.exe "
Tanjiro merupakan seorang Pendekar pedang dengan jurus teknik air dengan rank A => Kuat
PS D:\Belajar Python>
```

8.2. Upcasting

Child class mengakses atribut yang ada pada kelas atas (parent class).

Contoh Program :

```python
class Karakter:
    Teknik = " matahari" # Atribut parent class
    def __init__(self, nama, role):
        self.nama = nama
        self.role = role

    def info_karakter(self):
        print(f"{self.nama} merupakan seorang {self.role} dengan rank {self.rank}")

class Pahlawan(Karakter):
    def __init__(self, nama, role, rank):
        super().__init__(nama, role)
        self.rank = rank

    def info_karakter(self):
        print(f"{self.nama} merupakan seorang {self.role} dan teknik {super().Teknik} dengan rank {self.rank}")
        # Pada self.teknik diambil dari atribut class parent

player_1 = Pahlawan("Tanjiro", "Pendekar pedang dengan jurus teknik air", "A => Kuat")
player_1.info_karakter()
```

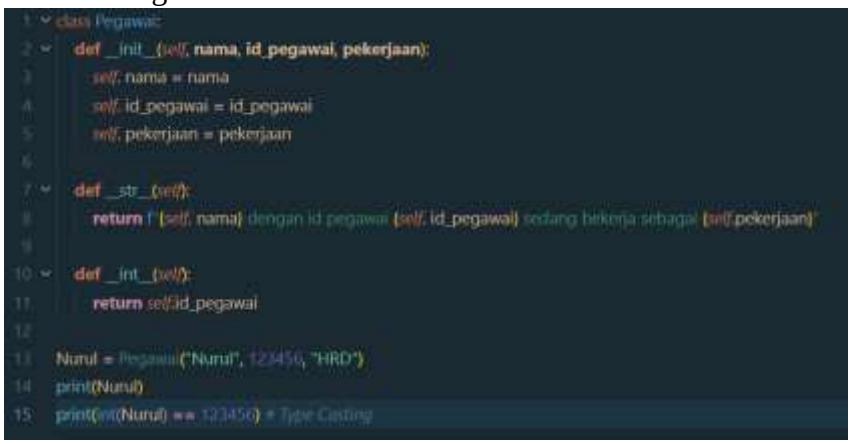
Output :

```
PS D:\Belajar Python> & C:/Users/LENOVO/AppData/Local/Programs/Python/Python311/python.exe "d:/Belajar Python/Ku
Tanjiro merupakan seorang Pendekar pedang dengan jurus teknik air dan teknik  matahari dengan rank A => Kuat
PS D:\Belajar Python>
```

8.3. Type Casting

Konversi tipe kelas agar memiliki sifat/perilaku tertentu yang secara default tidak dimiliki kelas tersebut. Umumnya dapat dilakukan menggunakan magic method.

Contoh Program :



```python
class Pegawai:
    def __init__(self, nama, id_pegawai, pekerjaan):
        self.nama = nama
        self.id_pegawai = id_pegawai
        self.pekerjaan = pekerjaan

    def __str__(self):
        return f"{self.nama} dengan id pegawai {self.id_pegawai} sedang bekerja sebagai {self.pekerjaan}"

    def __int__(self):
        return self.id_pegawai

Nurul = Pegawai("Nurul", 123456, "HRD")
print(Nurul)
print(int(Nurul) == 123456) # Type Casting
```

Output :

```
PS D:\Belajar Python> & C:/Users/LENOVO/AppData/Local/Program
Nurul dengan id pegawai 123456 sedang bekerja sebagai HRD
True
PS D:\Belajar Python>
```