PRAKTIKUM
PEMROGRAMAN BERBASIS OBJEK

TUGAS 5

Nama : Umy Afifah

NIM : 1211440087

Kelas : RB

INSTITUT TEKNOLOGI SUMATERA

TAHUN AJARAN 2022/2023

LAMPUNG SELATAN

2023

MODUL 1

1. Pengenalan Bahasa Pemrograman

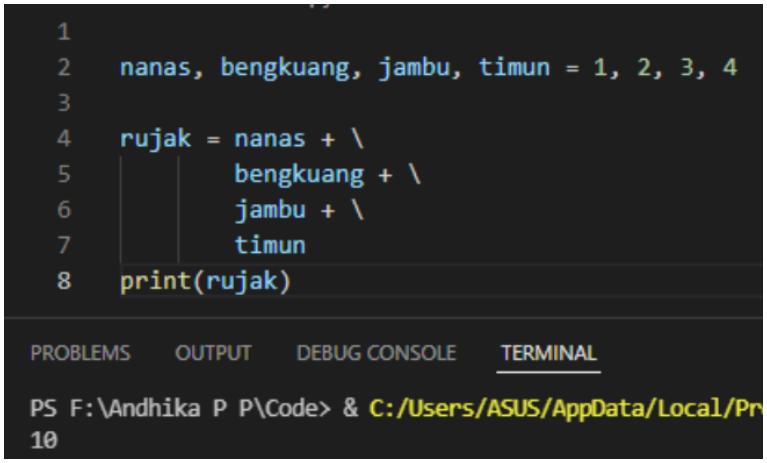
Bahasa Pemrograman python dibuat oleh Guido Van Rossum pada tahun 1980-an di Belanda. Bahasa Pemrograman ini lebih mendukung paradigma pemrograman *objek-oriented, functional* dan *structural*. Python saat ini merupakan bahasa pemrograman yang memiliki sintaks yang mudah, memiliki banyak *library*, serta dapat digunakan untuk pemrograman desktop maupun mobile. Bahasa pemrograman ini mementingkan *readability* pada kode dan untuk menjawab hal itu, maka python menggunakan indentasi.

2. Dasar Pemrograman

2.1 Sintaks Dasar

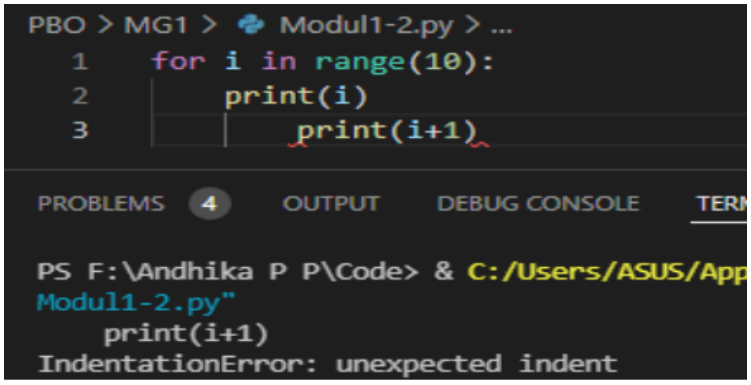
- Statement

  Semua perintah yang dapat dieksekusi pada Python disebut *statement*. *statement* dapat direpresentasikan pada baris baru atau dapat menggunakan *backslash* (\ ).



- **Baris dan Indentasi**

  Python tidak menggunakan kurung kurawal untuk mengelompokan blok kode melainkan menggunakan spasi/tab.Contoh yang Error.



2.2 Variabel dan Tipe Data Primitif

Variabel berfungsi untuk menyimpan suatu nilai. Untuk mendeklarasikan variabel, maka dibutuhkan beberapa tipe data berikut:

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 mendeklarasi variabel pada Python:

```
var1 = True # Boolean
angka1 = 10 # Integer
desimal = 3.14 # Float
huruf1 = 'Praktikum Pemrograman Python' # String
```

2.3 Operator

a. Operator Aritmatika

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

```
1  Num1 = int(input("Angka Pertama: "))
2  Num2 = int(input("Angka Kedua: "))
3  print("Hasil: ", Num1 + Num2)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Angka Pertama: 10
Angka Kedua: 20
Hasil:  30
```

```
1  Num1 = int(input("Angka Pertama: "))
2  Num2 = int(input("Angka Kedua: "))
3  print("Hasil: ", Num1 // Num2)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Angka Pertama: 32
Angka Kedua: 5
Hasil:  6
```

b. Operator Perbandingan

Operator perbandingan adalah operator yang digunakan untuk membandingkan 2 buah yang hasil perbandingannya adalah *True* atau *False*, seperti lebih besar dari, lebih kecil dari, sama dengan dan sebagainya. Contohnya:

```
1
2   Num1 = int(input("Angka Pertama: "))
3   Num2 = int(input("Angka Kedua: "))
4   lebih_kecil = Num1 < Num2
5   print("Hasil: ", lebih_kecil)
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

Angka Pertama: 2
Angka Kedua: 3
Hasil:  True
```

c. Operator Penugasan

Operator penugasan adalah operator yang digunakan untuk memberi nilai ke variabel. Contohnya a=1, yang artinya operator penugasan yang memberi nilai 1 ke variabel a yang di kiri. Contoh programnya :

```
1   Num1 = int(input("Angka Pertama: "))
2   Num2 = 10
3   Num2 -= Num1
4   print("Hasil: ", Num2)
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

Angka Pertama: 30
Hasil:  -20
```

d. Operator Logika

Operator logika berfungsi untuk melakukan operasi logika, seperti *or*, *and* dan *not*. Contohnya:

```
1   #Operator Logika
2   X = 10
3   Y = 20
4   Operator1 = Y and Y<15
5   print("Hasil Kondisi Operator1: ", Operator1)
6
7   Operator2 = X or Y>15
8   print("Hasil Kondisi Operator2: ", Operator2)
9
10  Operator3 = X and Y>15 or X and Y<11
11  print("Hasil Kondisi Operator3: ", Operator3)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
Hasil Kondisi Operator1:  False
Hasil Kondisi Operator2:  10
Hasil Kondisi Operator3:  True
```

e. Operator Bitwise

Operator bitwise adalah operator yang melakukan operasi bit terhadap operand dan beroperasi bit per bit sesuai dengan namanya. Contohnya angka 2 dalam bit ditulis 10 dalam notasi biner. Contoh program:

```
1
2   Num1 = int(input("Angka Pertama: "))
3   Num2 = int(input("Angka Kedua: "))
4   bitwise = Num1 & Num2
5   print("Hasil: ", bitwise)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
Angka Pertama: 10
Angka Kedua: 12
Hasil:  8
```

f. Operator Identitas

Operator identitas berfungsi untuk memeriksa apakah 2 buah nilai atau variabel berada pada lokasi memori yang sama atau tidak dengan menggunakan *is* dan *is not*. Contohnya:

```
1   A='Praktikum'
2   B='Praktikum'
3   C='PBO'
4
5   print(" A is B: ", A is B)
6   print(" A is not C: ", A is not C)
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL
A is B:  True
A is not C:  True
```

g. Operator Keanggotaan

Operator keanggotaan berfungsi untuk memeriksa apakah suatu nilai atau variabel merupakan anggota yang ditemukan di dalam suatu data (*string, list, tuple, set,* dan *dictionary*) atau tidak menggunakan *in* dan *not in*. Contohnya :

```
1   A='Praktikum'
2   B='Praktikum PBO'
3   C='PBO'
4
5   print(" A in B: ", A in B)
6   print(" A not in C: ", A not in C)
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL
A in B:  True
A not in C:  True
```

2.4 Tipe Data Bentukan

- *List* : Kumpulan data terurut, dapat diubah dan elemennya bisa sama.
- *Tuple* : Kumpulan data terurut, tidak dapat diubah dan elemennya bisa sama
- *Set* : Kumpulan data tidak terurut, tidak terindeks dan elemennya tidak ada yang sama
- *Dictionary* : Kumpulan data tidak terurut, tidak terindeks dan elemennya bisa sama, serta memiliki key dan nilai.

Contohnya:

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

a. Percabangan IF

Percabangan ini hanya menggunakan satu kondisi saja, yaitu kondisi IF. Contohnya:

```
1   #menentukan bilangan positif dan negatif
2
3   print("Masukkan nilai N: ", end="")
4   N = int(input())
5
6   if(N<0):
7       print(str(N) + " bilangan negatif")
```

b. Percabangan IF-ELSE

Percabangan ini berfungsi untuk memberi kondisi untuk 2 pernyataan saja. Contohnya:

```
#menentukan bilangan positif dan negatif

print("Masukkan nilai N: ", end="")
N = int(input())

if(N<0):
    print(str(N) + " bilangan negatif")
else:
    print(str(N) + " bilangan positif")
```

c. Percabangan IF-ELSE-IF

Percabangan ini berfungsi untuk memberi kondisi untuk lebih dari 2 pernyataan. Contohnya:

```
#menentukan bilangan positif dan negatif

print("Masukkan nilai N: ", end="")
N = int(input())

if(N<0):
    print(str(N) + " bilangan negatif")
elif(N==0):
    print(str(N) + " bilangan nol")
else:
    print(str(N) + " bilangan positif")
```

d. Nested IF

Percabangan ini disebut percabangan bersarang karena di dalam suatu percabangan terdapat percabangan yang lain di dalamnya. Contohnya:

```
#menentukan bilangan positif dan negatif

print("Masukkan nilai N: ", end="")
N = int(input())

if(N<=0):
    if(N==0):
        print(str(N) + " bilangan nol")
    else:
        print(str(N) + " bilangan negatif")

else:
    print(str(N) + " bilangan positif")
```

2.6 Perulangan

a. Perulangan For

Perulangan *for* merupakan perulangan yang batasannya telah didefinisikan terlebih dahulu dan biasanya digunakan untuk iterasi pada list, tuple, atau string. Salah satu contohnya:

```
1  #contoh perulangan for pada list
2  namaMahasiswa = ["Joko", "Budi", "Bambang", "Eka"]
3  for i in namaMahasiswa:
4      print(i)
```

output :

```
Joko
Budi
Bambang
Eka
```

Sintaks umum penggunaan range pada for:

1. Menggunakan 1 parameter

```
for i in range(x):
    #lakukan sesuatu
```

Note **:** Perulangan dari 0 hingga ke x

2. Menggunakan 2 parameter

```
for i in range(x,y):
    #lakukan sesuatu
```

Note **:** Perulangan dari x hingga ke < y

3. Menggunakan 3 parameter

```
for i in range(x,y,z):
    #lakukan sesuatu
```

Note **:** Perulangan dari x hingga ke < y dengan bertambah/berkurang sesuai z.

b. Perulangan While

Berbeda dengan perulangan *for*, perulangan *while* merupakan perulangan yang akan dieksekusi ketika kondisi tertentu terpenuhi. Contohnya:

```
1  #menghitung karakter c sebelum tanda !
2
3  count=int(0)
4
5  kalimat=(input())
6
7  while(kalimat!='!'):
8
9      if kalimat=='c':
10         count+=1
11     kalimat=(input())
12
13 print("jumlah karakter c adalah ", count)
14
```

```
b
c
c
a
d

!
jumlah karakter c adalah  2
```

2.7 Fungsi

Fungsi atau *method* biasanya menggunakan sintaks *def* dan berfungsi untuk mengantisipasi penulisan blok kode yang berulang. Contohnya:

```
def kelulusan(nama, nilai):
    if nilai ⩾ 40:
        return f"{nama} lulus dengan nilai {nilai}"
    else:
        return f"{nama} mengulang dengan nilai {nilai}"

list_mhs = [
    ["Bambang", 70], ["Anton", 55], ["Budi", 85],
    ["Rani", 75], ["Siti", 35], ["Aulia", 90]
]

for i in list_mhs:
    print(kelulusan(i[0], i[1]))
```

Hasil:

```
/run/media/dhika/Multi/Project/Praktikum/PBO
Bambang lulus dengan nilai 70
Anton lulus dengan nilai 55
Budi lulus dengan nilai 85
Rani lulus dengan nilai 75
Siti mengulang dengan nilai 35
Aulia lulus dengan nilai 90
```

\

MODUL 2

1. Kelas

Kelas atau *class* merupakan sebuah *blueprint* dari suatu objek yang dibuat. Dengan menggunakan *class*, maka dapat mendesain suatu objek secara bebas. Namun *class* tidak dapat langsung digunakan. Solusinya adalah dengan mengimplementasi menjadi sebuah objek terlebih dahulu, seperti kelas Mobil, kelas Manusia dan sebagainya. Contohnya:

```
1   class mobil:
2       def __init__(self, merk, warna, tahun):
3           self.merk = merk
4           self.warna = warna
5           self.tahun = tahun
6       def __str__(self):
7           return "{} {} {}".format(self.merk, self.warna, self.tahun)
8
9   mobil1 = mobil("Toyota", "Hitam", "2018")
10  mobil2 = mobil("Honda", "Merah", "2017")
11
12  print(mobil1)
13  print(mobil2)
```

Pada *class* terdapat __init__ *method* yang berperan sebagai konstruktor untuk membuat sebuah objek. Kemudian pada sebuah kelas terdapat atribut dan *method* (fungsi).

a. Atribut/Property

Dalam suatu kelas terdapat 2 jenis atribut, yaitu atribut objek dan atribut kelas. Atribut objek merupakan atribut yang dimiliki oleh masing-masing objek atau biasanya berada di dalam sebuah fungsi. Sedangkan atribut kelas adalah atribut yang dideklarasikan di dalam kelas namun tidak di dalam fungsi yang ada di kelas tersebut.

b. Method

Method atau disebut juga sebagai fungsi di dalam sebuah kelas. *Method* dapat diibaratkan sebagai aktivitas/proses yang dapat dilakukan oleh sebuah objek. Misalkan manusia dapat berjalan, berjalan dan sebagainya.

Contoh dari Atribut dan *Method* di dalam kelas:

main.py

```python
class Saya:
  jumlah_kaki=2 #atribut kelas
  jumlah_tangan=2
  def __init__(self, nama, gender):
    self.nama=nama #atribut objek
    self.gender=gender

  def tampil(self): #method
    print("Nama :",self.nama)
    print("Jenis Kelamin :",self.gender)
    print("Jumlah kaki :",Saya.jumlah_kaki)
    print("Jumlah tangan :", Saya.jumlah_tangan)

Bendry=Saya("Ben", "Pria")
Bendry.tampil() #memanggil method
```

```
Nama : Ben
Jenis Kelamin : Pria
Jumlah kaki : 2
Jumlah tangan : 2
```

2. Objek

Objek berfungsi sebagai perwakilan suatu kelas saat dipanggil ke *main*. Cara merepresentasikan objek ini adalah seperti berikut:

```
ini_objek = kelas_yang_dipanggil()
```

3. 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. Contohnya saat melakukan penjumlahan, maka operator __add__ yang dioperasikan. Gunakan sintaks dir(int) untuk melihat *Magic method* seperti berikut:

```
>>> 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 penggunaan *magic method* __add__:

```python
class Angka:
    def __init__(self, angka):
        self.angka = angka

    def __add__(self, objek):
        return self.angka + objek.angka

N=Angka(1)
P=Angka(2)
print (N + P)
```

output:

```
[Running] python -u "d:\itera\sm 6\asprak pbo\mg 3.py"
3
```

4. Konstruktor

Konstruktor adalah method yang “pasti” dijalankan secara otomatis pada saat sebuah objek dibuat untuk mewakili kelas tersebut. Selain operasi method dasar, konstruktor dapat menerima argumen yang diberikan ketika objek dibuat. Contohnya:

```python
class kelas_yang_dipanggil:

    #ini constructor
    def __init__(self,ini_argumen,ini_argumen_juga):
        self.nama = ini_argumen
        self.pekerjaan = ini_argumen_juga

    #ini buat panggil nama
    def panggilnama (self):
        print("Orang ini bernama", self.nama)
    #ini buat panggil kerjaan
    def panggilpekerjaan (self):
        print("Orang ini bekerja sebagai",self.pekerjaan)

ini_objek = kelas_yang_dipanggil("Zhongli","Tukang Gali Kubur")

ini_objek.panggilnama()
ini_objek.panggilpekerjaan()
```

```
PS D:\ngasprak\pbo\mg02>  & 'C:\Users\ammar\Ap
2022.0.1814523869\pythonFiles\lib\python\debug
Orang ini bernama Zhongli
Orang ini bekerja sebagai Tukang Gali Kubur
PS D:\ngasprak\pbo\mg02>
```

5. Destruktor

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

```python
class Angka:
    def __init__(self, angka):
        self.angka = angka

    def __del__(self):
        print ("objek {self.angka} dihapus")

N=Angka(1)
P=Angka(2)
```

```
[Running] python -u "d:\itera\sm 6\asprak pbo\mg 3.py"
objek {self.angka} dihapus
objek {self.angka} dihapus

[Done] exited with code=0 in 0.161 seconds
```

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, sedangkan getter digunakan untuk mengambil nilai. Contohnya:

```python
class siswa:
    def __init__(self, umur = 0):
        self._umur = umur

    # getter method
    def get_umur(self):
        return self._umur

    # setter method
    def set_umur(self, x):
        self._umur=x

raj = siswa()

# setting the umur using setter
raj.set_umur(19)
# retrieving umur using getter
print(raj.get_umur())
print(raj._umur)
```

```
PS D:\itera\Asprak PBO 2022>
Asprak PBO 2022/mg 2.py"
19
19
PS D:\itera\Asprak PBO 2022>
```

7. Decorator

Decorator yang biasanya ditandai dengan simbol (@) adalah alat yang memungkinkan programmer untuk mengubah perilaku fungsi atau kelas dengan cara membungkus fungsi lain untuk memperluas perilaku dari fungsi yang dibungkus,

seperti *@property*, *@classmethod* dan lain sebagainya. Salah satu contohnya adalah *Decorator Property*:

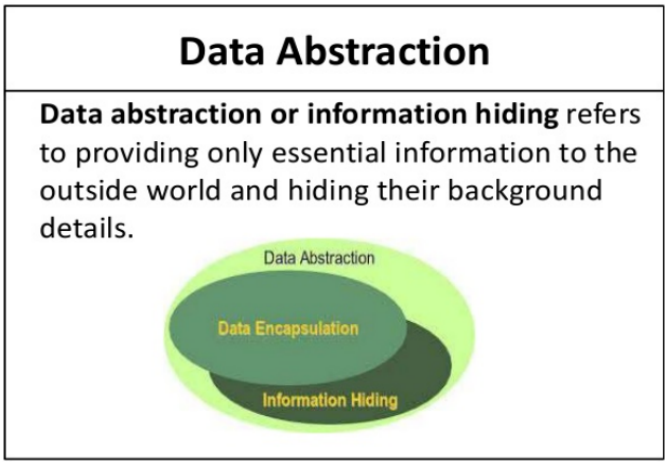
```python
class Siswa:
    def __init__(self, nama, umur = 15):
        self.__nama = nama
        self.__umur = umur

    @property
    def umur(self):
        print("Fungsi getter umur dipanggil")
        return self.__umur

    @umur.setter
    def umur(self, x):
        print("Fungsi setter umur dipanggil")
        self.__umur = x

Bambang = Siswa("Bambang")

# Akan error karena bersifat private
#print(Bambang.__umur)

# Gunakan fungsi property decorator
print(Bambang.umur)
Bambang.umur += 5
print(Bambang.umur)
```

```
Fungsi getter umur dipanggil
15
Fungsi getter umur dipanggil
Fungsi setter umur dipanggil
Fungsi getter umur dipanggil
20
```

MODUL 3

1. Abstraksi

Abstraksi adalah konsep OOP dimana model yang dibuat hanya memperlihatkan atribut yang esensial dan menyembunyikan detail-detail yang tidak penting dari user. gunanya untuk mengurangi kompleksitas, atau dapat dikatakan bahwa Abstraksi adalah cara untuk menyembunyikan informasi yang tidak dibutuhkan.



2. Enkapsulasi

Enkapsulasi adalah sebuah metode yang digunakan untuk mengatur struktur kelas dengan cara menyembunyikan alur kerja dari kelas tersebut. Yang dimaksud dengan struktur kelas tersebut adalah property dan method. 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*.

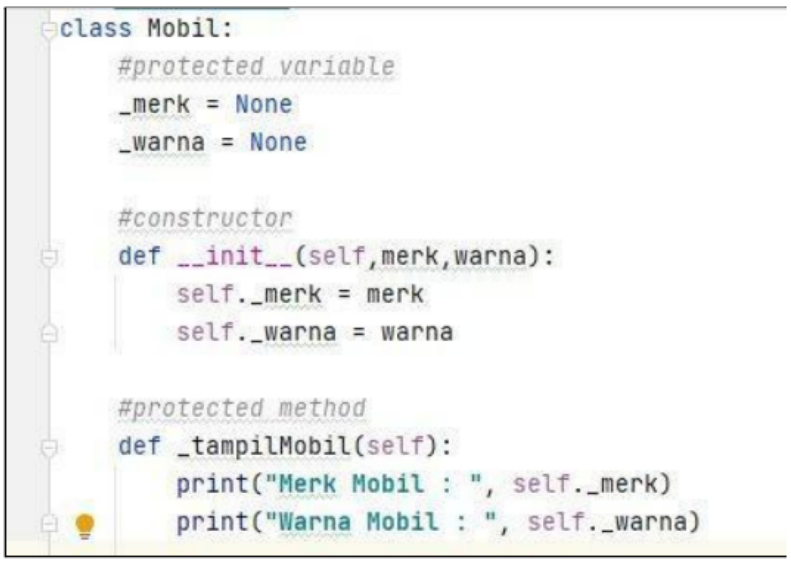
a. Public Access Modifier

Objek berjenis *public* baik atribut maupun metode dapat diakses dari dalam dan luar kelas tersebut. Penulisan atribut *public* seperti pada biasanya, yaitu tanpa menggunakan *underscore*. Contohnya:

```
class Mahasiswa:
    global_variable_jumlah = 0

    def __init__(self,nama,semester):
        self.nama = nama
        self.semester = semester
        Mahasiswa.global_variable_jumlah = Mahasiswa.global_variable_jumlah + 1

    def printjumlah(self):
        print ("total mahasiswa adalah ", Mahasiswa.global_variable_jumlah, " Orang")

    def printprofil(self):
        print ("Nama : ", self.nama)
        print ("Semester : ", self.semester)
        print()
```

b. Protected Access Modifier

Objek berjenis *protected* baik atribut maupun metode hanya dapat diakses dari dalam kelas dan turunan kelasnya. Untuk membuat atribut atau metode objek berjenis protected dapat dilakukan dengan cara menambahkan awalan garis bawah tunggal (_) ke dalam nama tersebut. Contohnya:



```
class Mobil:
    #protected variable
    _merk = None
    _warna = None

    #constructor
    def __init__(self,merk,warna):
        self._merk = merk
        self._warna = warna

    #protected method
    def _tampilMobil(self):
        print("Merk Mobil : ", self._merk)
        print("Warna Mobil : ", self._warna)
```

c. Private Access Modifier

Atribut maupun metode yang berjenis private hanya dapat diakses dari dalam kelasnya saja. Untuk membuat atribut atau metode objek berjenis private dapat dilakukan dengan cara menambahkan awalan garis bawah ganda (__) ke dalam nama. Terdapat cara untuk mengakses atribut *private* dari luar kelas yaitu: **nama_object_instance._NamaKelas__nama_atribut**. Contohnya:

```
class Mobil:
    #private variable
    __merk = None
    __warna = None

    #constructor
    def __init__(self,merk,warna):
        self.__merk = merk
        self.__warna = warna

    #private method
    def _tampilMobil(self):
        print("Merk Mobil : ", self.__merk)
        print("Warna Mobil : ", self.__warna)
```

3. Object

a. Membuat Instance Object.

Untuk membuat instance dari kelas yang telah dibuat dapat dilakukan dengan menggunakan nama dari class kemudian argumen diterima oleh metode init.

```
class Mahasiswa:
  def __init__(self, nama, NIM):
    self.nama=nama
    Mahasiswa.NIM=NIM
#nama_objek=nama_kelas(isi_atribut_pada_fungsi_init)
Bendry=Mahasiswa("Bendry", 121140111)
```

b. Mengakses Atribut Object.

Dari objek yang telah dibuat dapat dilakukan pengaksesan atribut dari objek dengan menggunakkan operator dot(titik). Contohnya:

```
namaObjek.atribut
```

c. Menambah, Menghapus dan Mengubah Atribut Object.

Objek yang sudah dibuat dapat dimodifikasi seperti ditambah, dihapus, ataupun diubah atributnya. Contohnya:

```
maha3 = Mahasiswa("Inu",3)

print("Sebelum diubah")
maha3.printprofil()

maha3.semester = 6

print("Setelah diubah")
maha3.printprofil()
```

```
Sebelum diubah
Nama :  Inu
Semester :  3

Setelah diubah
Nama :  Inu
Semester :  6
```

Error disebabkan oleh atribut nama yang telah dihapus.

d. Cara memodifikasi atribut dari suatu objek
   - **getattr(obj, name, [default])** – Mengakses atribut dari objek.
   - **hasattr(obj, name)** – Memeriksa apakah suatu objek memiliki atribut tertentu.
   - **setattr(obj, name, value)** – Mengatur nilai atribut. Jika ternyata atribut tidak ada, maka atribut tersebut akan dibuat.
   - **delattr(obj, name)** – Menghapus atribut dari suatu objek.

MODUL 4

1. Inheritance (Pewarisan)

Inheritance adalah salah satu konsep dasar dari *Object Oriented Programming* (OOP). Pada *inheritance*, kita dapat menurunkan kelas dari kelas lain untuk hirarki kelas yang saling berbagi atribut dan metode. Contoh:

```
1  class Manusia:
2    def __init__(self, nama, umur):
3      self.nama = nama
4      self.umur = umur
5
6    def cetakProfile(self):
7      print(f"{self.nama} berumur {self.umur}")
8
9  class Mahasiswa(Manusia):
10   pass
11
12 mhs1 = Mahasiswa("Joko", 20)
13 mhs1.cetakProfile()
```

```
Console  Shell

Joko berumur 20
>
```

- **Inheritance Identik**

Inheritance identik merupakan pewarisan yang menambahkan *constructor* pada *class child* sehingga *class child* memiliki *constructor*-nya sendiri tanpa menghilangkan *constructor* pada *class parent*-nya. Contohnya:

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

    def biodata(self):
        print("Nama saya " + self.firstname + " " + self.lastname)

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

    def biodata(self):
        print("Nama saya " + self.namadepan + " " + self.namabelakang)
        print("Pekerjaan : " + self.pekerjaan)

pelajar = Pekerja('Lukas','Sandy','Mahasiswa')
pelajar.biodata()
```

- **Menambah Karakteristik pada Child Class**

Pada *child class* dapat ditambahkan beberapa fitur tambahan baik atribut maupun method sehingga *child class* tidak identik dengan *parent class*. Contoh:

```
class Lingkaran():

    phi = float(3.14)

    def __init__(self, r):
        self.r = r

    def luasLingkaran(self):
        self.luas = Lingkaran.phi * self.r * self.r
        return self.luas

class Tabung(Lingkaran):
    def __init__(self, r, t):
        super().__init__(r)
        self.tinggi = t

    def luasPermukaan(self):
        self.luas = 2 * self.phi * self.r * (self.r + self.tinggi)
        return self.luas

tb1 = Tabung(3, 2)
l1 = Lingkaran(3)
print("luas permukaan tabung" , tb1.luasPermukaan())
print("Luas lingkaran", l1.luasLingkaran())
```

```
→  ~ /usr/bin/python3.7 /home/klmn/class.py
luas permukaan tabung 94.2
Luas lingkaran 28.259999999999998
```

2. Polymorphism

Polymorphism berarti banyak (poly) dan bentuk (morphism), dalam Pemrograman Berbasis Objek konsep ini memungkinkan digunakannya suatu interface yang sama untuk memerintah objek agar melakukan aksi atau tindakan yang mungkin secara prinsip sama namun secara proses berbeda. Contohnya:

```
class Tomato():
      def type(self):
        print("Vegetable")
      def color(self):
        print("Red")
class Apple():
      def type(self):
        print("Fruit")
      def color(self):
        print("Red")

def func(obj):
        obj.type()
        obj.color()

obj_tomato = Tomato()
obj_apple = Apple()
func(obj_tomato)
func(obj_apple)
```

3. Override/Overriding

Pada konsep OOP di Python kita dapat menimpa suatu metode yang ada pada *parent class* dengan mendefinisikan kembali *method* dengan nama yang sama pada *child class*. Contoh:

```
class bangunDatar():
    def tampil(self):
        print("Ini bangun datar")

class Persegi(bangunDatar):
    def tampil(self):
        print("Ini persegi")

P1 = Persegi()
P1.tampil()
```

```
→  ~ /usr/bin/python3.7 /home/klmn/class.py
Ini persegi
```

4. Overloading

Metode overloading mengizinkan sebuah class untuk memiliki sekumpulan fungsi dengan nama yang sama dan argumen yang berbeda. Secara umum overloading memiliki beberapa signature, yaitu jumlah argumen, tipe argumen, tipe keluaran dan urutan argumen. Contoh:

```
class Duck:
    def __init__(self, name):
        self.name = name
    def quack(self):
        print('Quack!')
class Car:
    def __init__(self, model):
        self.model = model
    def quack(self):
        print('I can quack, too!')

def quacks(obj):
    obj.quack()

donald = Duck('Donald Duck')
car = Car('Tesla')
quacks(donald)
quacks(car)
```

```
Quack!
I can quack, too!
```

5. Multiple Inheritance

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

```
class Base1:
    pass

class Base2:
    pass

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

```
class Base:
    pass

class Derived1(Base):
    pass

class Derived2(Derived1):
    pass
```

6. Method Resolution Order di Python

MRO adalah urutan pencarian metode dalam hirarki *class*. Hal ini terutama berguna dalam multiple inheritance. Urutan MRO dalam python yaitu bawah-atas dan kiri-kanan. Artinya, method dicari pertama kali di kelas objek. jika tidak

ditemukan, pencarian berlanjut ke *super class*. Jika terdapat banyak superclass (*multiple inheritance*), pencarian dilakukan di kelas yang paling kiri dan dilanjutkan ke kelas sebelah kanan. Contohnya:

```python
class A:
    def method(self):
        print("A.method() dipanggil")

class B:
    def method(self):
        print("B.method() dipanggil")

class C( A, B):
    pass

class D(B,A):
    pass

c=C()
c.method()

d=D()
d.method()
```

```
.exe  d:/itera/sm 6/asp
A.method() dipanggil
B.method() dipanggil
PS D:\itera\sm 6>
```

7. Dynamic Cast

a. Implisit

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

```python
num_int = 123
num_flo = 1.23

num_new = num_int + num_flo

print("datatype of num_int:",type(num_int))
print("datatype of num_flo:",type(num_flo))

print("Value of num_new:",num_new)
print("datatype of num_new:",type(num_new))
```

Output:

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

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. Contoh:

```
num_int = 123
num_str = "456"

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

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

num_sum = num_int + num_str

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

Output:

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

8. Casting

a. Downcasting

Parent class mengakses atribut yang ada pada kelas bawah (*child class*) contoh:

```
class Manusia:

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

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

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

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

Hasil:

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

b. Upcasting

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

```python
class Manusia:
    namabelakang = "Putra"

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

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

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

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

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

Hasil:

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

c. Type casting

Konversi tipe kelas agar memiliki sifat/perilaku tertentu yang secara default tidak dimiliki kelas tersebut. Karena Python merupakan bahasa pemrograman berorientasi objek, maka semua variabel atau instansi di Python pada dasarnya merupakan objek (kelas) yang sifat/perilakunya dapat dimanipulasi jika dibutuhkan (umumnya melalui *magic method*). Contoh:

```python
class Mahasiswa:

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

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

    def __int__(self):
        return self.__nim

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