

**PRAKTIKUM  
PEMROGRAMAN BERBASIS OBJEK**

**TUGAS 5  
RESUME MODUL 1 SAMPAI MODUL 4**



Disusun Oleh:

Nama : Bendry Lakburlawal

NIM : 1211440111

Kelas : RB

**PROGRAM STUDI TEKNIK INFORMATIKA**

**INSTITUT TEKNOLOGI SUMATERA**

**LAMPUNG SELATAN**

**2023**

<b>MODUL 1.....</b>	<b>2</b>
<b>1. Pengenalan Bahasa Pemrograman.....</b>	<b>2</b>
<b>2. Dasar Pemrograman.....</b>	<b>2</b>
2.1 Sintaks Dasar.....	2
2.2 Variabel dan Tipe Data Primitif.....	3
2.3 Operator.....	3
2.4 Tipe Data Bentuk.....	7
2.5 Percabangan.....	7
2.6 Perulangan.....	9
2.7 Fungsi.....	10
<b>MODUL 2.....</b>	<b>11</b>
<b>1. Kelas.....</b>	<b>11</b>
a. Atribut/Property.....	11
b. Method.....	11
<b>2. Objek.....</b>	<b>12</b>
<b>3. Magic Method.....</b>	<b>12</b>
<b>4. Konstruktor.....</b>	<b>13</b>
<b>5. Destruktor.....</b>	<b>13</b>
<b>6. Setter dan Getter.....</b>	<b>14</b>
<b>7. Decorator.....</b>	<b>14</b>
<b>MODUL 3.....</b>	<b>15</b>
<b>1. Abstraksi.....</b>	<b>15</b>
<b>2. Enkapsulasi.....</b>	<b>15</b>
a. Public Access Modifier.....	15
b. Protected Access Modifier.....	16
c. Private Access Modifier.....	16
<b>3. Object.....</b>	<b>16</b>
<b>MODUL 4.....</b>	<b>18</b>
<b>1. Inheritance (Pewarisan).....</b>	<b>18</b>
• Inheritance Identik.....	18
• Menambah Karakteristik pada Child Class.....	19
<b>2. Polymorphism.....</b>	<b>19</b>
<b>3. Override/Overriding.....</b>	<b>20</b>
<b>4. Overloading.....</b>	<b>20</b>
<b>5. Multiple Inheritance.....</b>	<b>21</b>
<b>6. Method Resolution Order di Python.....</b>	<b>21</b>
<b>7. Dynamic Cast.....</b>	<b>22</b>
<b>8. Casting.....</b>	<b>23</b>

# MODUL 1

## 1. Pengenalan Bahasa Pemrograman

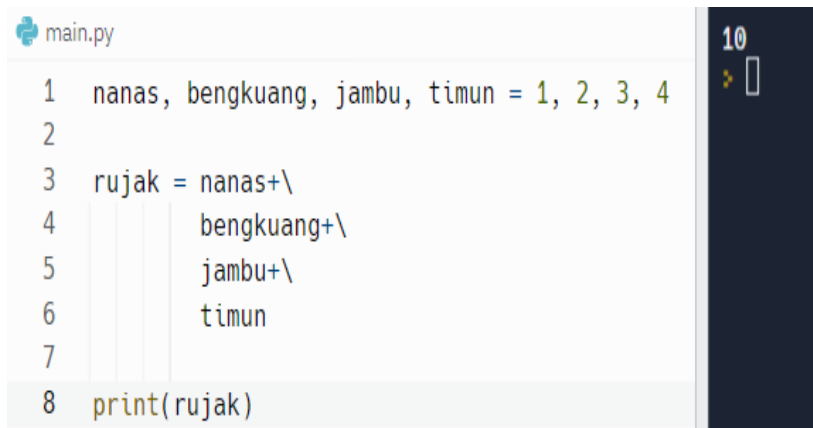
Bahasa Pemrograman python dibuat oleh Guido Van Rossum pada tahun 1980-an di Belanda. Bahasa Pemrograman ini lebih mendukung paradigma pemrograman *objek-oriented*, *functional* dan *structural*. Python saat ini merupakan bahasa pemrograman yang memiliki sintaks yang mudah, memiliki banyak *library*, 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* (\).



```
main.py
1  nanas, bengkuang, jambu, timun = 1, 2, 3, 4
2
3  rujak = nanas+\
4         bengkuang+\
5         jambu+\
6         timun
7
8  print(rujak)
```

10

- Baris dan Indentasi

Python tidak menggunakan kurung kurawal untuk mengelompokkan blok kode melainkan menggunakan spasi/tab.



```
main.py
1  for i in range(10):
2      print(i, end=" ")
```

0123456789

## 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:

```
main.py
1  a = True #Boolean
2  Angka = 10 #Integer
3  Desimal= 3.14 #Float
4  Huruf = 'Python' #String
5  Huruf2 = "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:

```
main.py
1  a1 = 2
2  a2 = 3
3
4  print("Hasil jumlah:", a2+a1)
5  print("Hasil kurang:", a2-a1)
6  print("Hasil bagi:", a2/a1)
7  print("Hasil kali:", a2*a1)
8  print("Hasil bagi pembulatan:", a2//a1)
9  print("Hasil pangkat:", a2**a1)
10 print("Hasil modulo: ", a2%a1)
```

```
Hasil jumlah: 5
Hasil kurang: 1
Hasil bagi: 1.5
Hasil kali: 6
Hasil bagi pembulatan: 1
Hasil pangkat: 9
Hasil modulo: 1
> []
```

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:

```
main.py
1  a1 = 2
2  a2 = 3
3
4  print("Hasil < :", a2<a1)
5  print("Hasil > :", a2>a1)
6  print("Hasil ==:", a2==a1)
7  print("Hasil !=:", a2!=a1)
8  print("Hasil >=:", a2>=a1)
9  print("Hasil <=:", a2<=a1)
```

```
Hasil < : False
Hasil > : True
Hasil ==: False
Hasil !=: True
Hasil >=: True
Hasil <=: False
> []
```

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 :

```
main.py
1  a, b = 2, 3
2
3  c=2+3 #menugaskan a+b ke c
4  print("Hasil c :", c) #c=5
5  c+=a #c=c+a
6  print("Hasil c :", c) #c=7
7  c-=a #c=c-a
8  print("Hasil c :", c) #c=5
9  c*=a #c=c*a
10 print("Hasil c :", c) #c=10
11 c/=a #c=c/a
12 print("Hasil c :", c) #c=5
13 c**=a #c=c**a
14 print("Hasil c :", c) #25
15 c//=a #c=c//a
16 print("Hasil c :", c) #12
17 c%=a #c=c%a
18 print("Hasil c :", c) #0
```

```
Hasil c : 5
Hasil c : 7
Hasil c : 5
Hasil c : 10
Hasil c : 5.0
Hasil c : 25.0
Hasil c : 12.0
Hasil c : 0.0
> []
```

#### d. Operator Logika

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

```
main.py
1  a, b = 2, 3
2
3  operator1= a and b<5
4  print("Kondisi operator1 : ", operator1)
5
6  operator2= a or not b
7  print("Kondisi operator2 : ", operator2)
8
9  operator3= a and b<1 or a and b>5
10 print("Kondisi operator3 : ", operator3)
```

```
Kondisi operator1 : True
Kondisi operator2 : 2
Kondisi operator3 : False
> []
```

#### 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:

```
main.py
1  a, b= 5, 3
2  print("Nilai a:",a,"\nNilai b:",b,"\n")
3
4  c=a&b #bitwise AND
5  print("Hasil a&b :", c)
6
7  c=a|b #bitwise OR
8  print("Hasil a|b :", c)
9
10 c=~b #bitwise NOT
11 print("Hasil ~b :", c)
12
13 c=a^b #bitwise XOR
14 print("Hasil a^b :", c)
15
16 c=a>>5 #bitwise right shift
17 print("Hasil a>>5 :", c)
18
19 c=a<<5 #bitwise left shift
20 print("Hasil a<<5 :", c)
```

```
Nilai a: 5
Nilai b: 3

Hasil a&b : 1
Hasil a|b : 7
Hasil ~b : -4
Hasil a^b : 6
Hasil a>>5 : 0
Hasil a<<5 : 160
> []
```

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:

```
main.py
1 A="PrakPrak"
2 B="Prak"
3 C="PrakPrak"
4
5 print("A is B :", A is B)
6 print("B is not C :", B is not C)
7 print("A is C :", A is C)
```

```
A is B : False
B is not C : True
A is 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 :

```
main.py
1 A="PrakPrak"
2 B="Prak"
3 C="rak"
4
5 print("B in A :", B in A)
6 print("B not in C :", B not in C)
7 print("A in C :", A in C)
```

```
B in A : True
B not in C : True
A in C : False
> []
```

## 2.4 Tipe Data Bentukan

- *List* : Kumpulan data terurut, dapat diubah dan elemennya bisa sama.

```
main.py
1 list = ["a", "b", "c", "c"]
2 print(list)
```

```
['a', 'b', 'c', 'c']
> []
```

- *Tuple* : Kumpulan data terurut, tidak dapat diubah dan elemennya bisa sama

```
main.py
1 tuple = ("a", "b", "c", "c")
2 print(tuple)
```

```
('a', 'b', 'c', 'c')
> []
```

- *Set* : Kumpulan data tidak terurut, tidak terindeks dan elemennya tidak ada yang sama

```
main.py
1 set = {"c", "b", "a", "a"}
2 print(set)
```

```
{'a', 'c', 'b'}
> []
```

- *Dictionary* : Kumpulan data tidak terurut, tidak terindeks dan elemennya bisa sama, serta memiliki key dan nilai.

<pre>main.py 1 dict = {1:"c", 2:"b", 3:"a"} 2 print(dict) 3 4 print(dict[1])</pre>	<pre>{1: 'c', 2: 'b', 3: 'a'} c &gt; []</pre>
--	---

## 2.5 Percabangan

### a. Percabangan IF

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

<pre>main.py 1 #mengecek bilangan positif saja 2 n=int(input("Masukkan n :")) 3 4 if n&gt;0 : 5     print(n, "= Positif")</pre>	<pre>Masukkan n :4 4 = Positif &gt; []</pre>
---	--

### b. Percabangan IF-ELSE

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

<pre>main.py 1 #mengecek bilangan positif dan negatif 2 n=int(input("Masukkan n :")) 3 4 if n&gt;0 : 5     print(n, "= Positif") 6 else : 7     print(n, "= Negatif")</pre>	<pre>Masukkan n :-1 -1 = Negatif &gt; []</pre>
---	--

### c. Percabangan IF-ELSE-IF

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

<pre>main.py 1 #mengecek bilangan positif dan negatif 2 n=int(input("Masukkan n :")) 3 4 if n&gt;0 : 5     print(n, "= Positif") 6 elif n&lt;0 : 7     print(n, "= Negatif") 8 else: 9     print(n, "= Netral")</pre>	<pre>Masukkan n :0 0 = Netral &gt; []</pre>
---	---



#### d. Nested IF

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

```
main.py
1  #mengecek bilangan positif dan
   negatif
2  n=int(input("Masukkan n :"))
3
4  if n>0 :
5      if n%2==0:
6          print(n, "Genap Positif")
7      else:
8          print(n, "Ganjil Positif")
9  else:
10     if n%2==0:
11         print(n, "Genap Negatif")
12     else:
13         print(n, "Ganjil Negatif")
```

Masukkan n :5  
5 Ganjil 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:

```
main.py
1  #contoh perulangan for pada list
2  data=["Bendry", "Lakburlawal",
        "121140111", "RB"]
3  for i in data:
4      print(i)
```

Bendry  
Lakburlawal  
121140111  
RB  
> █

Sintaks umum penggunaan range pada for:

##### 1. Menggunakan 1 parameter

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

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

##### 2. Menggunakan 2 parameter

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

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

### 3. Menggunakan 3 parameter

```
1 ~ for i in range(x,y,z):  
2     #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:

```
main.py  
1  n=0  
2 ~ while n!=-999:  
3      n=int(input("Masukkan n: "))
```

Masukkan n: 2  
Masukkan n: 3  
Masukkan n: 4  
Masukkan n: 5  
Masukkan n: 6  
Masukkan n: -999  
>

## 2.7 Fungsi

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

```
main.py  
1 ~ def tambah(a, b):  
2     print(a+b)  
3  
4  a=int(input("Masukkan a :"))  
5  b=int(input("Masukkan b :"))  
6  tambah(a,b)
```

Masukkan a :5  
Masukkan b :6  
11  
>

## 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:



```
main.py
1 class Kucing:
2     def __init__(self, nama, warna, suara):
3         self.nama=nama
4         self.warna=warna
5         self.suara=suara
6
7     def __str__(self):
8         return "{} {} {}".format(self.nama, self.warna, self.suara)
9
10 meong1=Kucing("Joni", "Coklat", "Meonggg")
11 meong2=Kucing("Surti", "Hitam", "Auuuuu")
12
13 print(meong1)
14 print(meong2)
```

The screenshot shows a Python IDE with a file named 'main.py'. The code defines a class 'Kucing' with two methods: '\_\_init\_\_' and '\_\_str\_\_'. The '\_\_init\_\_' method takes three arguments: 'nama', 'warna', and 'suara', and assigns them to instance variables. The '\_\_str\_\_' method returns a string formatted as '{nama} {warna} {suara}'. Below the class definition, two instances of the 'Kucing' class are created: 'meong1' with attributes 'Joni', 'Coklat', and 'Meonggg'; and 'meong2' with attributes 'Surti', 'Hitam', and 'Auuuuu'. The code then prints both instances. The output on the right shows the string representation of the objects: 'Joni Coklat Meonggg' and 'Surti Hitam Auuuuu'.

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
1 class Saya:
2     jumlah_kaki=2 #atribut kelas
3     jumlah_tangan=2
4     def __init__(self, nama, gender):
5         self.nama=nama #atribut objek
6         self.gender=gender
7
8     def tampil(self): #method
9         print("Nama :",self.nama)
10        print("Jenis Kelamin :",self.gender)
11        print("Jumlah kaki :",Saya.jumlah_kaki)
12        print("Jumlah tangan :", Saya.jumlah_tangan)
13
14    Bendry=Saya("Ben", "Pria")
15    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:

```
main.py
1 class Saya:
2     def __init__(self, nama, gender):
3         self.nama=nama
4         self.gender=gender
5
6    Bendry=Saya("Ben", "Pria") #pemanggilan kelas
    dengan objek Bendry
```

## 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:

```
main.py
1 print(dir(int))
2
```

```
['_abs_', '_add_', '_and_', '_bool_', '_ceil_', '_class_', '_delattr_', '_dir_', '_divmod_', '_doc_', '_eq_', '_float_', '_floor_', '_floor_div_', '_format_', '_ge_', '_getattr_', '_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_', 'as_integer_ratio', 'bit_count', 'bit_length', 'conjugate', 'denominator', 'from_bytes', 'imag', 'numerator', 'real', 'to_bytes']
>
```

Contoh penggunaan *magic method* `__add__`:

```
main.py
1 class Angka:
2     def __init__(self, angka):
3         self.angka=angka
4
5     def __add__(self, objek):
6         return self.angka+objek.angka
7 X=Angka(4)
8 Y=Angka(6)
9 print(X+Y)
```

10  
> []

#### 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:

```
main.py
1 class Saya:
2     def __init__(self, nama, gender): #konstruktor
3         self.nama=nama
4         self.gender=gender
5
6     def tampil(self): #menampilkan inputan
7         print("Nama :",self.nama)
8         print("Jenis Kelamin :",self.gender)
9
10 Bendry=Saya("Ben", "Pria") #nilai akan diisi ke
    konstruktor
11 Bendry.tampil() #memanggil method
```

Nama : Ben  
Jenis Kelamin : Pria  
> []

#### 5. Destruktor

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

```
main.py
1 class Saya:
2     def __init__(self, nama, gender): #konstruktor
3         self.nama=nama
4         self.gender=gender
5
6     def __del__(self):
7         print(f>Nama {self.nama} dihapus")
8         print(f"Gender {self.gender} dihapus")
9
10 Bendry=Saya("Ben", "Pria") #nilai akan diisi ke
    konstruktor
11 del Bendry
```

Nama Ben dihapus  
Gender Pria dihapus  
> []

## 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:

```
main.py
1 class Saya:
2     def __init__(self, nama, NIK):
3         self.nama=nama
4         self.__NIK=NIK
5
6     def set_NIK(self, x):
7         self.__NIK=x
8
9     def get_NIK(self):
10        return self.__NIK
11
12 Bendry=Saya("Bendry", None)
13 Bendry.set_NIK(123456)
14
15 print(Bendry.nama)
16 print(Bendry.get_NIK())
```

Bendry  
123456

## 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*:

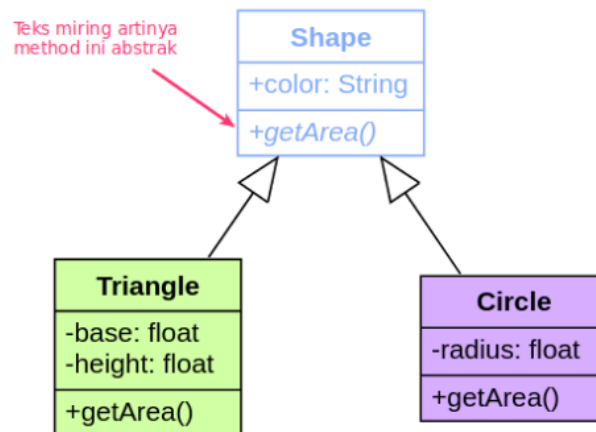
```
main.py
1 class Saya:
2     def __init__(self, nik):
3         self.__NIK=nik
4
5     @property
6     def NIK(self):
7         return self.__NIK
8
9     @NIK.setter
10    def NIK(self, x):
11        self.__NIK=x
12
13 Bendry=Saya("123456")
14 print(Bendry.NIK)
```

123456

## 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:

```
main.py
1 class Mahasiswa:
2     def __init__(self, nama, NIM):
3         self.nama=nama #atribut public
4         self.NIM=NIM #atribut public
5     def tampilkan(self): #fungsi public
6         print(f>Nama : {self.nama}\nNIM : {self.NIM}")
7
8 Bendry=Mahasiswa("Bendry", "121140111")
9 Bendry.tampilkan()
```

```
Nama : Bendry
NIM : 121140111
```

## 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:

```
1 ~ class Mahasiswa:
2 ~     def __init__(self, nama, NIM):
3 ~         self._nama=nama #atribut protected
4 ~         self._NIM=NIM #atribut protected
5
6 ~     def _tampilkan(self): #fungsi protected
7 ~         print(f>Nama : {self._nama}\nNIM :
           {self._NIM}")
```

## 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:



```
main.py
1 ~ class Mahasiswa:
2 ~     def __init__(self, nama, NIM):
3 ~         self.nama=nama #atribut public
4 ~         Mahasiswa.__NIM=NIM #atribut private
5
6 ~     def __ubah_NIM(self, x): #fungsi private
7 ~         Mahasiswa.__NIM=x
8
9 ~     def tampil(self): #fungsi public
10 ~         print(f>Nama : {self.nama}\nNIM : {Mahasiswa.__NIM}")
11
12 Bendry=Mahasiswa("Bendry", 121140111)
13 Bendry.tampil()
14 Bendry._Mahasiswa__ubah_NIM(123456) #panggil fungsi private
15 Bendry.tampil()
```

Terminal Output:

```
Nama : Bendry
NIM : 123456
```

## 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`.

```
1 ~ class Mahasiswa:
2 ~     def __init__(self, nama, NIM):
3 ~         self.nama=nama
4 ~         Mahasiswa.NIM=NIM
5
6 #nama_objek=nama_kelas(isi_atribut_pada_fungsi_init)
7 Bendry=Mahasiswa("Bendry", 121140111)
```



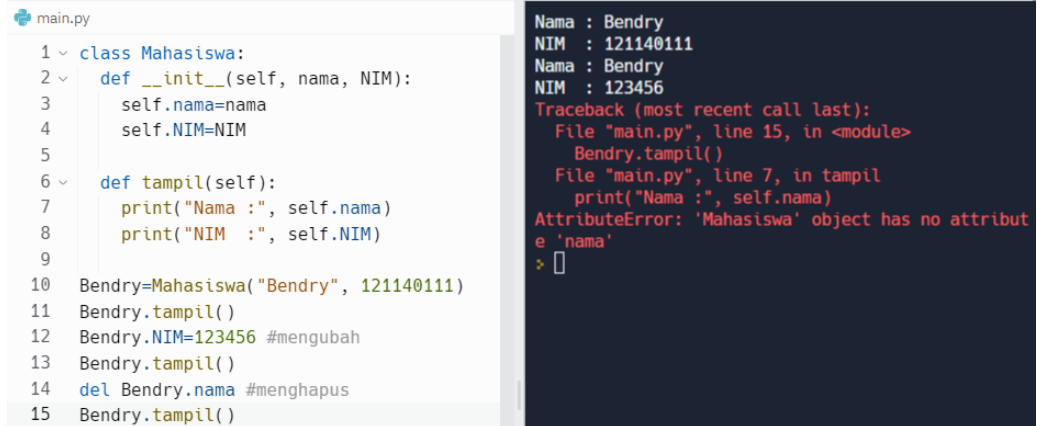
b. Mengakses Atribut Object.

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

```
#Nama_Objek.Nama_atribut/fungsi  
Bendry.tampil()
```

c. Menambah, Menghapus dan Mengubah Atribut Object.

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



```
main.py  
1 class Mahasiswa:  
2     def __init__(self, nama, NIM):  
3         self.nama=nama  
4         self.NIM=NIM  
5  
6     def tampil(self):  
7         print("Nama :", self.nama)  
8         print("NIM :", self.NIM)  
9  
10 Bendry=Mahasiswa("Bendry", 121140111)  
11 Bendry.tampil()  
12 Bendry.NIM=123456 #mengubah  
13 Bendry.tampil()  
14 del Bendry.nama #menghapus  
15 Bendry.tampil()  
  
Nama : Bendry  
NIM : 121140111  
Nama : Bendry  
NIM : 123456  
Traceback (most recent call last):  
  File "main.py", line 15, in <module>  
    Bendry.tampil()  
  File "main.py", line 7, in tampil  
    print("Nama :", self.nama)  
AttributeError: 'Mahasiswa' object has no attribute 'nama'  
> []
```

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:

```
main.py
1 class Parent:
2     pass
3
4 class Child(Parent):
5     pass
```

- 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:

```
main.py
1 class Mahasiswa:
2     def __init__(self, nama, NIM):
3         self.nama=nama
4         self.NIM=NIM
5
6     def biodata(self):
7         print("Nama :", self.nama)
8         print("NIM :", self.NIM)
9
10 class KetuaHMIF(Mahasiswa):
11     def __init__(self, nama, NIM, prodi):
12         super().__init__(nama, NIM)
13         self.prodi=prodi
14
15     def biodata(self):
16         print("Nama :", self.nama)
17         print("NIM :", self.NIM)
18         print("Prodi:", self.prodi)
19
20 Beliau=KetuaHMIF("Dia", 6666, "IF")
21 Beliau.biodata()
```

```
Nama : Dia
NIM : 6666
Prodi: IF
> []
```

- **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:

```

main.py
1 class Mahasiswa:
2     minSKS=144
3     def __init__(self, nama, NIM):
4         self.nama=nama
5         self.NIM=NIM
6
7     def status(self):
8         print("Nama :", self.nama)
9         print("NIM  :", self.NIM)
10        print("SKS   :", Mahasiswa.minSKS)
11
12 class KetuaHMIF(Mahasiswa):
13     def __init__(self, nama, NIM, prodi, IPK, SKS_sekarang):
14         super().__init__(nama, NIM)
15         self.prodi=prodi
16         self.IPK=IPK
17         self.SKS=SKS_sekarang
18
19     def siakad(self):
20         self.sisaSKS=Mahasiswa.minSKS-self.SKS
21         print("Nama :", self.nama)
22         print("NIM  :", self.NIM)
23         print("Prodi:", self.prodi)
24         print("IPK  :", self.IPK)
25         print("SKS   :", self.SKS)
26         print("Sisa SKS:", self.sisaSKS)
27
28 Beliau=KetuaHMIF("Dia", 6666, "IF", 3.66, 66)
29 Alm=Mahasiswa("Bagus", 9999)
30 Beliau.siakad()
31 print()
32 Alm.status()

```

```

Nama : Dia
NIM  : 6666
Prodi: IF
IPK  : 3.66
SKS   : 66
Sisa SKS: 78

Nama : Bagus
NIM  : 9999
SKS   : 144
> 

```

## 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:

```

main.py
1 class Buaya():
2     def type(self):
3         print("Setia")
4     def color(self):
5         print("Loreng")
6
7 class Cowok():
8     def type(self):
9         print("Selingkuh")
10    def color(self):
11        print("Sawo mateng")
12
13 def func(objek):
14     objek.type()
15     objek.color()
16
17 Buaya1=Buaya()
18 Cowok1=Cowok()
19 func(Buaya1)
20 print()
21 func(Cowok1)

```

```

Setia
Loreng

Selingkuh
Sawo mateng
> 

```

### 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:

```
main.py
1 ~ class Buaya():
2 ~     def type(self):
3 ~         print("Setia")
4
5 ~ class Cowok(Buaya):
6 ~     def type(self):
7 ~         print("Selingkuh")
8
9 ~ Cowok1=Cowok( )
10 ~ Cowok1.type( )
```

Selingkuh

### 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:

```
main.py
1 ~ class Buaya:
2 ~     def type(self):
3 ~         print("Setia")
4
5 ~ class Cowok:
6 ~     def type(self):
7 ~         print("Selingkuh")
8
9 ~ def func(objek):
10 ~     objek.type( )
11
12 ~ Buaya1=Buaya( )
13 ~ Cowok1=Cowok( )
14 ~ func(Buaya1)
15 ~ func(Cowok1)
```

Setia  
Selingkuh

## 5. Multiple Inheritance

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

```
1 ~ class Ayah:
2     pass
3
4 ~ class Ibu:
5     pass
6
7 ~ class Anak(Ayah, Ibu):
8     pass
```

```
1 ~ class Kakek:
2     pass
3
4 ~ class Ayah:
5     pass
6
7 ~ class Anak(Ayah):
8     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:

```
1 ~ class A:
2     pass
3
4 ~ class B:
5     pass
6
7 ~ class C:
8     pass
9
10 ~ class X(A, B):
11     pass
12
13 ~ class Y(B, C):
14     pass
15
16 ~ class Z(Y, X, C):
17     pass
```

## 7. Dynamic Cast

### a. Implisit

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

```
main.py
1  a = 3
2  b = 3.5
3
4  c = a + b
5
6  print(type(a))
7  print(type(b))
8  print(c)
9  print(type(c))
```

```
<class 'int'>
<class 'float'>
6.5
<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:

```
main.py
1  a = 3
2  b = 3.5
3
4  b = int(b)
5  c = a + b
6
7  print(type(a))
8  print(type(b))
9  print(c)
10 print(type(c))
```

```
<class 'int'>
<class 'int'>
6
<class 'int'>
```

## 8. Casting

### a. Downcasting

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

contoh:

```
main.py
1 class Rakyat:
2     def __init__(self, nama, NIK):
3         self.nama=nama
4         self.NIK=NIK
5
6     def biodata(self):
7         print("Nama :", self.nama)
8         print("NIM :", self.NIK)
9         print("Status ", self.status)
10
11 class Saya(Rakyat):
12     def __init__(self, nama, NIK, status):
13         super().__init__(nama, NIK)
14         self.status=status
15
16 Bendry = Saya("Bendry", 813001, "Mahasiswa")
17 Bendry.biodata()
```

```
Nama : Bendry
NIM : 813001
Status Mahasiswa
```

### b. Upcasting

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

Contohnya:

```
main.py
1 class Rakyat:
2     Negara="Indonesia"
3     def __init__(self, nama, NIK):
4         self.nama=nama
5         self.NIK=NIK
6
7 class Saya(Rakyat):
8     def __init__(self, nama, NIK, status):
9         super().__init__(nama, NIK)
10        self.status=status
11
12    def biodata(self):
13        print("Negara :", super().Negara)
14        print("Nama :", self.nama)
15        print("NIM :", self.NIK)
16        print("Status :", self.status)
17
18 Bendry = Saya("Bendry", 813001, "Mahasiswa")
19 Bendry.biodata()
```

```
Negara : Indonesia
Nama : Bendry
NIM : 813001
Status : 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:

```
main.py
1 ~ class Rakyat:
2     Negara="Indonesia"
3 ~ def __init__(self, nama, NIK):
4     self.nama=nama
5     self.NIK=NIK
6
7 ~ class Saya(Rakyat):
8 ~ def __init__(self, nama, NIK, status):
9     super().__init__(nama, NIK)
10    self.status=status
11
12 ~ def __str__(self):
13    return f"Negara : {super().Negara}\nNama : {self.nama}\nNIM : {self.NIK}\nStatus : {self.status}"
14
15 Bendry = Saya("Bendry", "813001", "Mahasiswa")
16 print(Bendry)
```

```
Negara : Indonesia
Nama :Bendry
NIM : 813001
Status : Mahasiswa
> []
```



## **References**

MAHASISWA INFORMATIKA ITERA. (n.d.). *Modul Praktikum*.