

**RESUME PRAKTIKUM  
PEMROGRAMAN BERORIENTASI OBJEK  
PERTEMUAN 1-4**



Disusun oleh:

Nama : KHAIRANI BILQIS

NIM : 121140091

Kelas : Pemrograman Berorientasi Objek RB

**PROGRAM STUDI TEKNIK INFORMATIKA  
INSTITUT TEKNOLOGI SUMATERA  
LAMPUNG SELATAN  
2023**

## Daftar Isi

<b>Daftar Isi .....</b>	<b>2</b>
<b>Resume Pertemuan 1 .....</b>	<b>4</b>
<b>A. Sejarah Bahasa Pemrograman Python .....</b>	<b>4</b>
<b>B. Dasar Pemrograman Python.....</b>	<b>4</b>
1. Sintaks Dasar.....	4
2. Variabel dan Tipe Data Primitif.....	5
3. Operator .....	5
4. Tipe Data Bentuk.....	11
5. Percabangan .....	11
6. Perulangan.....	12
7. Fungsi.....	14
<b>Resume Pertemuan 2 .....</b>	<b>15</b>
<b>A. Kelas .....</b>	<b>15</b>
1. Atribut/Property .....	15
2. Method .....	16
<b>B. Objek.....</b>	<b>16</b>
<b>C. Magic Method.....</b>	<b>17</b>
<b>D. Konstruktor .....</b>	<b>17</b>
<b>E. Destruktor.....</b>	<b>18</b>
<b>F. Setter dan Getter .....</b>	<b>18</b>
<b>G. Decorator .....</b>	<b>19</b>
<b>Resume Pertemuan 3 .....</b>	<b>20</b>
<b>A. Abstraksi.....</b>	<b>20</b>
<b>B. Enkapsulasi (Encapsulation).....</b>	<b>20</b>
1. Public Access Modifier .....	20
2. Protected Access Modifier.....	20
3. Private Access Modifier.....	20
<b>C. Object.....</b>	<b>22</b>
1. Membuat Instance Object .....	22
2. Mengakses Atribut Object .....	22
<b>Resume Pertemuan 4 .....</b>	<b>23</b>
<b>A. Inheritance (Pewarisan) .....</b>	<b>23</b>
1. Inheritance Identik .....	23
2. Multiple Inheritance .....	24
<b>B. Polymorphism .....</b>	<b>25</b>

<b>C. Override/Overriding.....</b>	<b>26</b>
<b>D. Overloading .....</b>	<b>26</b>

## Resume Pertemuan 1

### A. Sejarah Bahasa Pemrograman Python

Bahasa pemrograman python dibuat oleh Guido Van Rossum pada tahun 1980-an akhir di *Centrum Wiskunde & Informatica*, Belanda. Python mendukung banyak paradigma pemrograman seperti object-oriented, functional dan structured. Bahasa Python menjadi populer sekarang dikarenakan sintaks nya yang mudah, didukung oleh library(modul) yang berlimpah dan dapat dipakai untuk pemrograman desktop maupun mobile, CLI, GUI, web, otomatisasi, hacking, IoT, robotika, dan lain sebagainya.

Python sangat mementingkan *readability* pada kode, untuk pengimplementasiannya Python tidak menggunakan kurung kurawal ({} ) atau keyword (ex. **start**, **begin**, **end**) melainkan menggunakan spasi (*white space*) untuk memisahkan blok-blok kodenya.

### B. Dasar Pemrograman Python

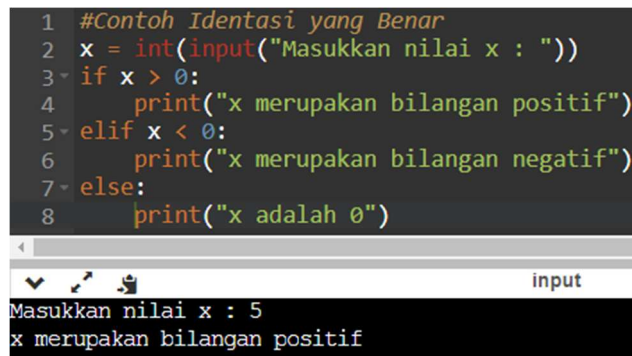
#### 1. Sintaks Dasar

##### 1.1 Statement

Statement adalah semua perintah yang dapat dieksekusi Python. Dalam Python akhir dari sebuah statement adalah baris baru (*newline*) tapi dapat dimungkinkan membuat statement yang terdiri dari beberapa baris dengan menggunakan *backslash* (\).

##### 1.2 Baris dan Indentasi

Python tidak menggunakan kurung kurawal sebagai grouping blok kode melainkan menggunakan spasi. Kode yang berada di blok yang sama harus memiliki jumlah spasi yang sama di awal.



```
1 #Contoh Identasi yang Benar
2 x = int(input("Masukkan nilai x : "))
3 if x > 0:
4     print("x merupakan bilangan positif")
5 elif x < 0:
6     print("x merupakan bilangan negatif")
7 else:
8     print("x adalah 0")
```

input

Masukkan nilai x : 5

x merupakan bilangan positif

Jika identasi yang dibuat salah maka compiler akan mengeluarkan **IndentationError**. Contoh yang salah:

```
1 #Contoh Identasi yang Salah
2 x = int(input("Masukkan nilai x : "))
3 if x > 0:
4     print("x merupakan bilangan positif")
5 elif x < 0:
6     print("x merupakan bilangan negatif")
7 else:
8     print("x adalah 0")
9
input
File "/home/main.py", line 4
    print("x merupakan bilangan positif")
    ^
IndentationError: expected an indented block after 'if' statement on line 3
```

## 2. Variabel dan Tipe Data Primitif

Variabel merupakan lokasi penyimpanan yang berguna untuk menyimpan suatu data atau suatu nilai. Sedangkan tipe data adalah pasangan dari variabel yang berhubungan dalam pendeklarasian variabel untuk menentukan jenis data yang ada, di bawah ini merupakan tipe data, jenis dan nilai yang ada dari tipe data yaitu :

1. bool (Boolean)  
Nilai : True/False
2. int (Bilangan Bulat)  
Nilai : Seluruh bilangan bulat
3. float (Bilangan real)  
Nilai : Seluruh bilangan real
4. string (Teks)  
Nilai : Kumpulan karakter

Contoh pendeklarasian dalam Python :

```
1 #Membuat variabel tanpa menuliskan tipe data
2 ceknilai1 = True #Boolean
3 angka1 = 10 #Integer
4 desimal1 = 3.14 #Float
5 huruf1 = "Tugas Praktikum" #String
6
7 #Membuat variabel dengan menuliskan tipe data
8 ceknilai2 = bool(True) #Boolean
9 angka2 = int(10) #Integer
10 desimal2 = float(3.14) #Float
11 huruf2 = str("Tugas Praktikum") #String
```

## 3. Operator

### 3.1 Operator Aritmatika

Operator aritmatika merupakan operator yang digunakan untuk melakukan operasi matematika. Contoh operator matematika dan fungsinya :

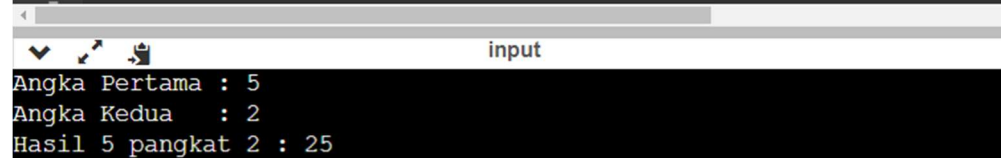
1. “+” merupakan operator penjumlahan yang berfungsi untuk menjumlahkan

Contoh penulisan :  $a + b$

2. “-” merupakan operator pengurangan yang berfungsi untuk mengurangi  
Contoh penulisan :  $a - b$
3. “\*” merupakan operator perkalian yang berfungsi untuk mengalikan  
Contoh penulisan :  $a * b$
4. “/” merupakan operator pembagian yang berfungsi untuk membagi  
Contoh penulisan :  $a / b$
5. “\*\*” merupakan operator pemangkatan yang berfungsi untuk memangkatkan bilangan  
Contoh penulisan :  $a ** b$
6. “//” merupakan operator pembagian bulat yang berfungsi untuk menghasilkan hasil bagi tanpa koma  
Contoh penulisan :  $a // b$
7. “%” merupakan operator modulus yang berfungsi untuk menghasilkan sisa pembagian 2 bilangan  
Contoh penulisan :  $a \% b$

Contoh :

```
1 #Contoh pemakaian operator aritmatika
2 angka1 = int(input("Angka Pertama : "))
3 angka2 = int(input("Angka Kedua : "))
4 print(f"Hasil {angka1} pangkat {angka2} : {angka1 ** angka2}")
```



### 3.2 Operator Perbandingan

Operator perbandingan merupakan operator yang digunakan untuk membandingkan 2 buah nilai yang dari hasil perbandingannya akan menghasilkan True atau False tergantung kondisi. Contoh operator perbandingan dan fungsinya :

1. “>” merupakan “operator lebih besar dari” yang akan menghasilkan True jika nilai sebelah kiri lebih besar dari nilai sebelah kanan  
Contoh penulisan :  $a > b$
2. “<” merupakan “operator lebih kecil dari” yang akan menghasilkan True jika nilai sebelah kiri lebih kecil dari nilai sebelah kanan  
Contoh penulisan :  $a < b$
3. “==” merupakan “operator sama dengan” yang akan menghasilkan True jika nilai sebelah kiri sama dengan nilai sebelah kanan  
Contoh penulisan :  $a == b$
4. “!=” merupakan “operator tidak sama dengan” yang akan menghasilkan True jika nilai sebelah kiri tidak sama dengan nilai sebelah kanan

Contoh penulisan :  $a \neq b$

5. “ $\geq$ ” merupakan “operator lebih besar atau sama dengan” yang akan menghasilkan True jika nilai sebelah kiri lebih besar atau sama dengan nilai sebelah kanan

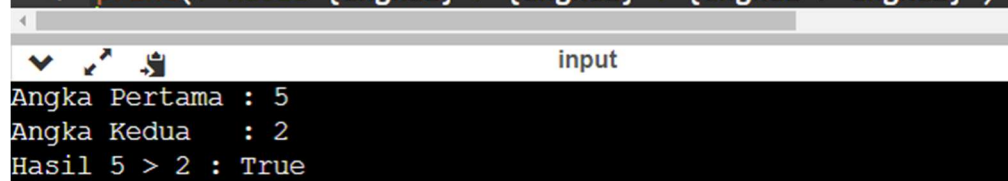
Contoh penulisan :  $a \geq b$

6. “ $\leq$ ” merupakan “operator lebih kecil atau sama dengan” yang akan menghasilkan True jika nilai sebelah kiri lebih kecil atau sama dengan nilai sebelah kanan

Contoh penulisan :  $a \leq b$

Contoh :

```
1 #Contoh pemakaian operator perbandingan
2 angka1 = int(input("Angka Pertama : "))
3 angka2 = int(input("Angka Kedua : "))
4 print(f"Hasil {angka1} > {angka2} : {angka1 > angka2}")
```



input

Angka Pertama : 5  
Angka Kedua : 2  
Hasil 5 > 2 : True

### 3.3 Operator Penugasan

Operator penugasan merupakan operator yang digunakan untuk memberi nilai ke variabel. Berikut operator penugasan dan penjelasannya :

1. “ $=$ ” menugaskan nilai yang ada di sebelah kanan ke operand di sebelah kiri  
Contoh penulisan :  $z = x + y$  menugaskan  $x + y$  ke  $z$

2. “ $+=$ ” menambahkan operand yang ada di kanan dengan operand yang ada di kiri dan hasilnya ditugaskan ke operand yang di kiri  
Contoh penulisan :  $a += b$  sama dengan  $a = a + b$

3. “ $-=$ ” mengurangi operand yang di kanan dengan operand yang ada di kiri dan hasilnya ditugaskan ke operand yang di kiri  
Contoh penulisan :  $a -= b$  sama dengan  $a = a - b$

4. “ $*=$ ” mengalikan operand yang di kanan dengan operand yang ada di kiri dan hasilnya ditugaskan ke operand yang di kiri  
Contoh penulisan :  $a *= b$  sama dengan  $a = a * b$

5. “ $/=$ ” membagi operand yang di kanan dengan operand yang ada di kiri dan hasilnya ditugaskan ke operand yang di kiri  
Contoh penulisan :  $a /= b$  sama dengan  $a = a / b$

6. “ $**=$ ” mengangkat operand yang di kanan dengan operand yang ada di kiri dan hasilnya ditugaskan ke operand yang di kiri  
Contoh penulisan :  $a **= b$  sama dengan  $a = a ** b$

7. “ $//=$ ” melakukan pembagian bulat operand di kanan terhadap operand di kiri dan hasilnya disimpan di operand yang di kiri

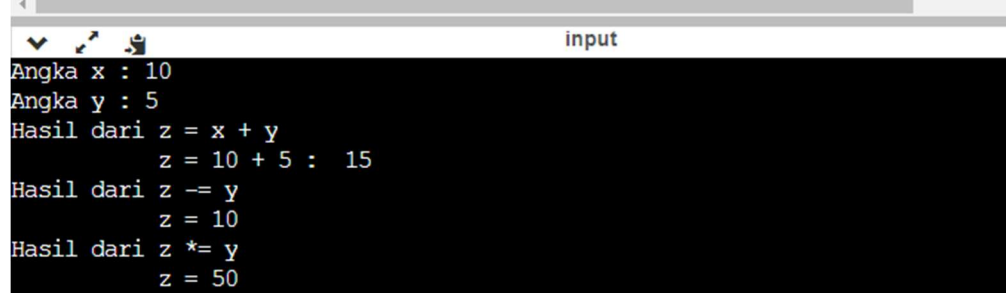
Contoh penulisan : **a //= b** sama dengan **a = a // b**

8. “**%=**” melakukan operasi sisa bagi operand di kanan dengan operand di kiri dan hasilnya disimpan di operand yang di kiri

Contoh penulisan : **a %= b** sama dengan **a = a % b**

Contoh :

```
1  #Contoh pemakaian operator penugasan
2  x = int(input("Angka x : "))
3  y = int(input("Angka y : "))
4
5  #Operator (=)
6  z = x + y
7  print(f"Hasil dari z = x + y \n          z = {x} + {y} : ", z)
8
9  #Operator (-=)
10 z -= y
11 print(f"Hasil dari z -= y \n          z =", z)
12
13 #Operator (*=)
14 z *= y
15 print(f"Hasil dari z *= y \n          z =", z)
```



```
input
Angka x : 10
Angka y : 5
Hasil dari z = x + y
          z = 10 + 5 : 15
Hasil dari z -= y
          z = 10
Hasil dari z *= y
          z = 50
```

### 3.4 Operator Logika

Operator logika merupakan operator yang digunakan untuk melakukan operasi logika.

Berikut operator logika dan penjelasannya :

1. “**and**” akan bernilai True jika kedua operandnya bernilai benar  
Contoh penulisan : **a and b**
2. “**or**” akan bernilai True jika salah satu atau kedua operandnya bernilai benar  
Contoh penulisan : **a or b**
3. “**not**” akan bernilai True jika operandnya bernilai salah (kebalikan nilai)  
Contoh penulisan : **not a**

Contoh :



```
1 #Contoh pemakaian operator logika
2 x = 10
3 y = 20
4
5 #Operator and
6 Operator1 = x > 15 and x < y
7 print("Hasil Kondisi Operator1 : ", Operator1)
8
9 #Operator or
10 Operator2 = x > 15 or x < y
11 print("Hasil Kondisi Operator2 : ", Operator2)
```

input

```
Hasil Kondisi Operator1 : False
Hasil Kondisi Operator2 : True
```

### 3.5 Operator Bitwise

Operator bitwise merupakan operator yang melakukan operasi bit terhadap operand. Operator ini beroperasi bit per bit sesuai dengan namanya. Berikut operator bitwise :

1. “&” merupakan operator bitwise AND  
Contoh penulisan : a & b
2. “|” merupakan operator bitwise OR  
Contoh penulisan : a | b
3. “~” merupakan operator bitwise NOT  
Contoh penulisan : ~ b
4. “^” merupakan operator bitwise XOR  
Contoh penulisan : a ^ b
5. “>>” merupakan operator bitwise right shift  
Contoh penulisan : a >> b
6. “<<” merupakan operator bitwise left shift  
Contoh penulisan : a << b

Contoh :

```
1 #Contoh pemakaian operator bitwise
2 angka1 = int(input("Angka Pertama : "))
3 angka2 = int(input("Angka Kedua : "))
4
5 #Operator bitwise and
6 bitwise = angka1 & angka2
7 print("Hasil bitwise AND :", bitwise)
```

input

```
Angka Pertama : 7
Angka Kedua : 11
Hasil bitwise AND : 3
```

### 3.6 Operator Identitas

Operator identitas merupakan operator yang digunakan memeriksa apakah dua buah nilai (atau variabel) berada pada lokasi memori yang sama. Berikut operator identitas dan penjelasannya :

1. **“is”** akan bernilai True jika kedua operand identik (menunjuk ke objek yang sama)


Contoh penulisan : a is b

2. **“is not”** akan bernilai True jika kedua operand tidak identik (tidak merujuk ke objek yang sama)

Contoh penulisan : a is not b

Contoh :

```
1 #Contoh pemakaian operator identitas
2 x = "Tugas Praktikum PBO"
3 y = "Resume Pertemuan 1-4"
4 z = "Tugas Praktikum PBO"
5
6 #Operator is
7 print("x is z :", x is z)
8
9 #Operator is not
10 print("x is not y :", x is not y)
```



### 3.7 Operator Keanggotaan

Operator keanggotaan merupakan operator yang digunakan untuk memeriksa apakah suatu nilai atau variabel merupakan anggota atau ditemukan di dalam suatu data (string, list, tuple, set, dan dictionary). Berikut operator keanggotaan dan penjelasannya :

1. **“in”** akan bernilai True jika nilai atau variabel ditemukan di dalam data


Contoh penulisan : 5 in a

2. **“not in”** akan bernilai True jika nilai atau variabel tidak ada di dalam data

Contoh penulisan : 5 not in a

Contoh :

```
1 #Contoh pemakaian operator keanggotaan
2 #Contoh pada list
3 list_angka = [1, 2, 3, 4, 5, 6, 7, 8, 9]
4
5 print("Hasil :", 5 in list_angka)
6 print("Hasil :", 10 not in list_angka)
7
8 #Contoh pada string
9 x = "Tugas Praktikum"
10 y = "Tugas Praktikum PBO"
11 z = "PBO"
12
13 print("x in y :", x in y)
14 print("x not in z:", x not in z)
```



## 4. Tipe Data Bentukan

### 4.1 List

Kumpulan data yang menyimpan berbagai tipe data, isinya dapat diubah, dan memungkinkan ada anggota yang sama

### 4.2 Tuple

Kumpulan data yang menyimpan berbagai tipe data, isinya tidak dapat diubah, dan memungkinkan ada anggota yang sama

### 4.3 Set

Kumpulan data yang tidak berurutan, tidak terindeks, elemen datanya harus unik, dan tidak memungkinkan ada anggota yang sama

### 4.4 Dictionary

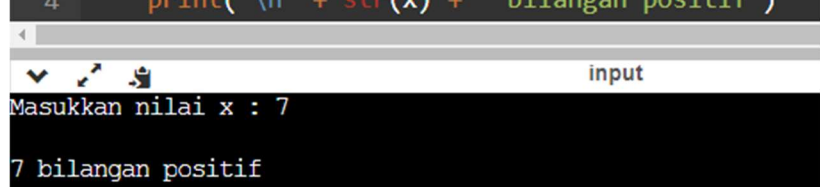
Kumpulan data yang menyimpan berbagai tipe data berupa pasangan penunjuk dan nilai, tidak berurutan, dapat diubah, tidak memungkinkan ada anggota yang sama

```
1 #Contoh penulisan tipe data bentukan
2 list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9]
3 tuple1 = (1, 3, 5, 7, 9, 1)
4 set1 = {"aku", "ayah", "ibu"}
5 dictionary1 = {"Warna" : "Biru", "Ibukota" : "Jakarta"}
```

## 5. Percabangan

### 5.1 Percabangan IF

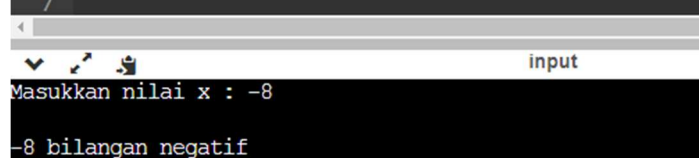
```
1 #Percabangan IF
2 x = int(input("Masukkan nilai x : "))
3 if (x > 0):
4     print("\n" + str(x) + " bilangan positif")
```



Percabangan ini hanya akan mencetak kondisi pertama yaitu kondisi if jika syaratnya memenuhi, jika tidak maka tidak akan mengeksekusi apapun. Jadi jika pengguna memasukkan angka untuk x yaitu kurang dari nol atau sama dengan nol maka program tidak akan mengeluarkan apapun.

### 5.2 Percabangan IF-ELSE

```
1 #Percabangan IF-ELSE
2 x = int(input("Masukkan nilai x : "))
3 if (x > 0):
4     print("\n" + str(x) + " bilangan positif")
5 else:
6     print("\n" + str(x) + " bilangan negatif")
7
```



Percabangan ini akan mencetak kondisi pertama yaitu kondisi if jika syaratnya memenuhi, jika tidak terpenuhi maka program akan mencetak kondisi yang kedua seperti program di atas.

### 5.3 Percabangan IF-ELSE-IF

```
1 #Percabangan IF-ELSE-IF
2 x = int(input("Masukkan nilai x : "))
3 if (x > 0):
4     print("\n" + str(x) + " bilangan positif")
5 elif (x < 0):
6     print("\n" + str(x) + " bilangan negatif")
7 else:
8     print("\n" + str(x) + " bilangan nol")
```

input

Masukkan nilai x : 0

0 bilangan nol

Percabangan ini akan mencetak kondisi pertama yaitu kondisi if jika syaratnya memenuhi, jika tidak terpenuhi maka program akan mencetak kondisi yang kedua yaitu elif, dan jika kedua kondisi tersebut masih belum terpenuhi maka program akan mencetak hasil dari kondisi else seperti program di atas.

### 5.4 Nested IF

Menempatkan percabangan di dalam percabangan yang lain sehingga membuat percabangan tersebut menjadi percabangan bersarang.

```
1 #Percabangan Nested IF
2 x = int(input("Masukkan nilai x : "))
3 if (x % 2 == 0):
4     if (x > 0):
5         print("\n" + str(x) + " bilangan genap positif")
6     elif (x < 0):
7         print("\n" + str(x) + " bilangan genap negatif")
8     else:
9         print("\n" + str(x) + " bilangan nol")
10
11 else:
12     if (x > 0):
13         print("\n" + str(x) + " bilangan ganjil positif")
14     elif (x < 0):
15         print("\n" + str(x) + " bilangan ganjil negatif")
16     else:
17         print("\n" + str(x) + " bilangan nol")
```

input

Masukkan nilai x : 10

10 bilangan genap positif

## 6. Perulangan

### 6.1 Perulangan For

Perulangan for biasa digunakan untuk iterasi pada urutan berupa list, tuple, atau string. Perulangan ini biasanya digunakan untuk iterasi yang batas perulangannya sudah diketahui. Sintaks dasar perulangan for :

```
1 - for i in (kondisi):  
2     #perintah
```

Sintaks umum penggunaan range pada for :

1. Menggunakan 1 parameter

Perulangan akan dilakukan dari indeks 0 sampai kurang dari x

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

2. Menggunakan 2 parameter

Perulangan akan dilakukan dari indeks ke-x sampai kurang dari y

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

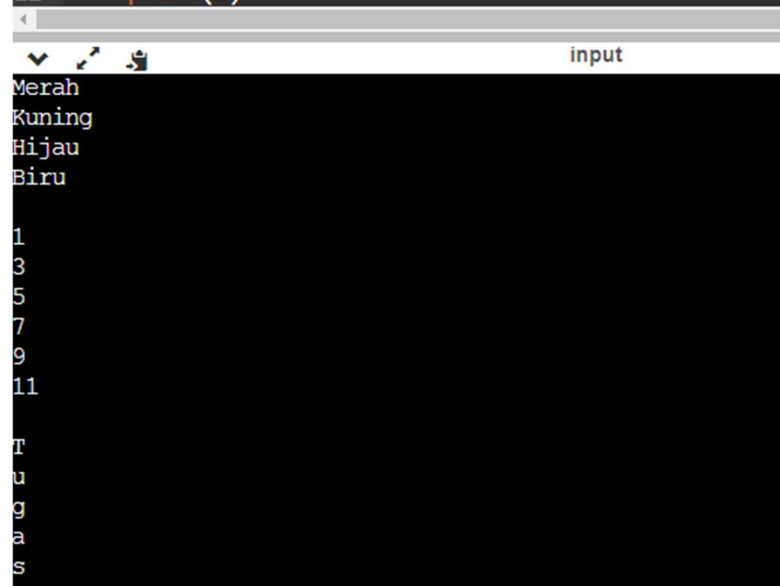
3. Menggunakan 3 parameter

Perulangan akan dilakukan dari indeks ke-x sampai kurang dari y dengan indeks bertambah/increment sejumlah z

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

Contoh perulangan for :

```
1 #contoh perulangan for  
2 list_warna = ["Merah", "Kuning", "Hijau", "Biru"]  
3 for i in (list_warna):  
4     print(i, end="\n")  
5     print()  
6  
7 for i in range(1, 12, 2):  
8     print(i, end="\n")  
9     print()  
10  
11 for i in ("Tugas"):  
12     print(i)
```



```
input  
Merah  
Kuning  
Hijau  
Biru  
  
1  
3  
5  
7  
9  
11  
  
T  
u  
g  
a  
s
```

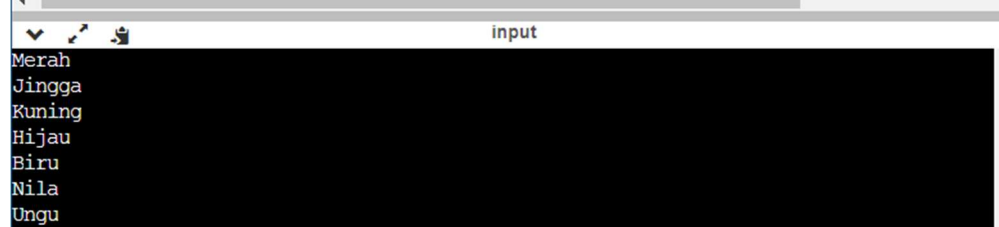
## 6.2 Perulangan While

Perulangan while dapat digunakan untuk iterasi yang memiliki batas perulangan selama suatu kondisi tertentu terpenuhi. Sintaks umum perulangan while :

```
1 while (kondisi):  
2     #lakukan sesuatu
```

Contoh perulangan while :

```
1 #contoh perulangan while  
2 list_warna = ["Merah", "Jingga", "Kuning", "Hijau", "Biru", "Nilu", "Ungu"]  
3  
4 i = 0  
5 while (i < len(list_warna)):  
6     print(list_warna[i])  
7     i += 1
```




input

Merah  
Jingga  
Kuning  
Hijau  
Biru  
Nilu  
Ungu

## 7. Fungsi

Fungsi berguna untuk mengeksekusi suatu blok *code* tanpa harus menuliskan berulang-ulang.

```
1 #contoh penggunaan fungsi  
2 def faktorial(N):  
3     if (N <= 1):  
4         return 1  
5     else:  
6         return N * faktorial(N-1)  
7     print()  
8  
9 N = int(input("Masukkan Angka : "))  
10 print(N, '! = ', faktorial(N), end="")
```



input

Masukkan angka : 3  
3 ! = 6



## Resume Pertemuan 2

### A. Kelas

Kelas atau *class* pada python dapat digambarkan sebagai sebuah blueprint (cetakan) dari objek (atau instance) yang ingin kita buat desainnya secara bebas. Kelas mendefinisikan dan berisi atribut/properti serta metode untuk objeknya nanti. Satu buah kelas dapat membuat banyak objek dan kelas sendiri tidak bisa langsung digunakan, harus diimplementasikan menjadi sebuah objek dulu, proses ini disebut Instansiasi. Untuk membuat kelas, gunakan kata kunci **class** diikuti oleh **nama kelas** tersebut dan tanda titik dua (":"). Contoh:

```
1 class Mahasiswa: #class
2     def __init__(self, nama, nim, kelas_pbo, sks): #konstruktor
3         self.nama = nama #atribut objek
4         self.nim = nim #atribut objek
5         self.kelas_pbo = kelas_pbo #atribut objek
6         self.sks = sks #atribut objek
7
8     #method
9     def data_mahasiswa(self):
10        print("Nama Mahasiswa      : ", self.nama)
11        print("NIM Mahasiswa       : ", self.nim)
12        print("Kelas PBO Siakad      : ", self.kelas_pbo)
13        print("Jumlah SKS Mahasiswa    : ", self.sks)
14
15 data = Mahasiswa("Khairani Bilqis", "121140091", "RB", "22") # data adalah objek dari class mahasiswa
16 data.data_mahasiswa()
```

input

```
Nama Mahasiswa      : Khairani Bilqis
NIM Mahasiswa       : 121140091
Kelas PBO Siakad   : RB
Jumlah SKS Mahasiswa : 22
```

Dapat dilihat diatas terdapat `__init__()`, method tersebut adalah konstruktor untuk membuat kelas mahasiswa. Untuk method `__init__()` lebih lanjut akan dibahas pada pembahasan selanjutnya.

#### 1. Atribut/Property

Dalam suatu kelas, umumnya dideklarasikan sebuah variabel yang akan disebut sebagai sebuah atribut. Karena dideklarasikan di dalam sebuah kelas, maka setiap objek yang dibentuk dari kelas tersebut juga akan memiliki atribut yang dimiliki oleh kelas tersebut. Terdapat 2 jenis atribut, yaitu atribut kelas dan atribut objek. Atribut kelas merupakan sifat yang dimiliki oleh sebuah kelas dan juga akan dimiliki oleh setiap objek. Sedangkan atribut objek adalah sebuah atribut dari masing - masing objek. Contoh :

```
1 class Mahasiswa: #class
2     kampus = "ITERA" #atribut kelas
3
4     def __init__(self, nama, nim, kelas_pbo, sks): #konstruktor
5         self.nama = nama #atribut objek
6         self.nim = nim #atribut objek
7         self.kelas_pbo = kelas_pbo #atribut objek
8         self.sks = sks #atribut objek
9
10    #method
11    def data_mahasiswa(self):
12        print("Nama Mahasiswa      :", self.nama)
13        print("NIM Mahasiswa       :", self.nim)
14        print("Kelas PBO Siakad    :", self.kelas_pbo)
15        print("Jumlah SKS Mahasiswa :", self.sks)
16
17 data = Mahasiswa("Khairani Bilqis", "121140091", "RB", "22") # data adalah objek dari class mahasiswa
18 data.data_mahasiswa()
19 print(f"Kampus          : {data.kampus}")
```

input

```
Nama Mahasiswa      : Khairani Bilqis
NIM Mahasiswa       : 121140091
Kelas PBO Siakad   : RB
Jumlah SKS Mahasiswa : 22
Kampus              : ITERA
```

Dari contoh diatas dapat dilihat bahwa kampus merupakan atribut kelas yang nilainya akan sama dalam setiap objek yang dibuat menggunakan kelas tersebut. Sedangkan nama, nim, kelas\_pbo, dan sks merupakan atribut objek yang akan memiliki nilai dari masing\_masing objek.

Untuk memanggil atribut dapat menggunakan sintaks :

`nama_objek.nama_atribut`

## 2. Method

Method adalah suatu fungsi yang terdapat di dalam kelas. Sama seperti atribut, semua objek yang dibuat menggunakan kelas yang sama akan memiliki method yang sama pula. Method dapat diibaratkan sebagai sebuah aktivitas/proses yang dapat dilakukan oleh sebuah objek.

Contoh :

```
1 class Mahasiswa: #class
2     kampus = "ITERA" #atribut kelas
3
4     def __init__(self, nama, nim, kelas_pbo, sks): #konstruktor
5         self.nama = nama #atribut objek
6         self.nim = nim #atribut objek
7         self.kelas_pbo = kelas_pbo #atribut objek
8         self.sks = sks #atribut objek
9
10    #method
11    def data_mahasiswa(self):
12        print("Nama Mahasiswa      :", self.nama)
13        print("NIM Mahasiswa       :", self.nim)
14        print("Kelas PBO Siakad      :", self.kelas_pbo)
15        print("Jumlah SKS Mahasiswa    :", self.sks)
16
17    data = Mahasiswa("Khairani Bilqis", "121140091", "RB", "22") # data adalah objek dari class mahasiswa
18    data.data_mahasiswa() # Memanggil method
```

input

```
Nama Mahasiswa      : Khairani Bilqis
NIM Mahasiswa       : 121140091
Kelas PBO Siakad   : RB
Jumlah SKS Mahasiswa : 22
```

Dari contoh diatas dapat dilihat bahwa kelas mahasiswa memiliki method `data_mahasiswa()` yang akan mencetak nama, nim, kelas pbo siakad, dan jumlah sks dari mahasiswa.

Untuk memanggil method dapat menggunakan sintaks :

`nama_objek.nama_method()`

## B. Objek

Objek adalah sesuatu yang “mewakili” kelas. Objek berfungsi sebagai pengganti pemanggilan sebuah kelas, sebuah objek hanya dapat mewakili sebuah kelas saja. Untuk membuat sebuah objek yang mewakili sebuah kelas, kita dapat membuatnya dengan cara memanggil nama dari kelas yang diinginkan, ditambah dengan tanda kurung (). Berikut sintaksnya :

`ini_objek = kelas_yang_dipanggil()`

Contoh :

```
data = Mahasiswa("Khairani Bilqis", "121140091", "RB", "22")
```

Pembuatan objek mempermudah pemanggilan kelas, terutama ketika nama dari suatu kelas terlalu panjang dan kelas tersebut kadang perlu berinteraksi dengan kelas lain.



### C. 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 seperti menggunakan operator kurang (`__sub__`), membuat objek (`__init__`), dan lain-lain. Tujuan utama dari penggunaan magic method adalah untuk mengubah sifat (behavior) bawaan dari suatu objek. Untuk melihat apa saja magic method bawaan python pada class int, gunakan sintaks `dir(int)`.

```
1 class Angka:
2     def __init__(self, angka):
3         self.angka = angka
4
5     def __sub__(self, objek):
6         return self.angka - objek.angka
7
8 x = Angka(5)
9 y = Angka(3)
10 print(x - y)
```

Magic method tidak terbatas pada kelas int saja, tetapi ada di setiap jenis objek dan variabel. Salah satu contoh magic method yaitu `__sub__()`. Method ini ditambahkan agar user dapat melakukan operasi pengurangan (-) secara langsung pada objek, tanpa mengakses atribut isi objek (dalam hal ini yaitu `self.angka`). Tanpa penambahan magic method `__sub__()`, maka statement `print(x - y)` akan menghasilkan error.

### D. Konstruktor

Konstruktor adalah *method* yang “pasti” dijalankan secara otomatis pada saat sebuah objek dibuat untuk mewakili kelas tersebut. Seperti dengan *method* lain, konstruktor dapat melakukan operasi seperti melakukan print. Selain operasi *method* dasar, konstruktor dapat menerima argumen yang diberikan ketika objek dibuat. Argumen ini akan diproses di dalam kelas nantinya. Contoh :

```
1 class Mahasiswa: #class
2     kampus = "ITERA" #atribut kelas
3
4     def __init__(self, nama, nim, kelas_pbo, sks): #konstruktor
5         self.nama = nama #atribut objek
6         self.nim = nim #atribut objek
7         self.kelas_pbo = kelas_pbo #atribut objek
8         self.sks = sks #atribut objek
9
10    #method
11    def data_mahasiswa(self):
12        print("Nama Mahasiswa      :", self.nama)
13        print("NIM Mahasiswa       :", self.nim)
14        print("Kelas PBO Siakad      :", self.kelas_pbo)
15        print("Jumlah SKS Mahasiswa    :", self.sks)
16
17    data = Mahasiswa("Khairani Bilqis", "121140091", "RB", "22") # data adalah objek dari class mahasiswa
18    data.data_mahasiswa() # Memanggil method
```

input

```
Nama Mahasiswa      : Khairani Bilqis
NIM Mahasiswa       : 121140091
Kelas PBO Siakad   : RB
Jumlah SKS Mahasiswa : 22
```

## E. Destruktor

Destruktor adalah fungsi yang dipanggil ketika user menghapus objek. Fungsi ini bekerja secara otomatis, jadi tidak perlu dilakukan pemanggilan. Tujuan adanya destruktur adalah melakukan final cleaning up atau “bersih-bersih” terakhir sebelum sebuah objek benar-benar dihapus dari memori.

```
1 class Angka:
2     def __init__(self, angka): #Konstruktor
3         self.angka = angka
4
5     def __del__(self): #Destruktor
6         print(f"Objek {self.angka} dihapus")
7
8 x = Angka(5)
9 y = Angka(3)
```

input

Objek 5 dihapus  
Objek 3 dihapus

## F. 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 (lebih jelas pada resume pertemuan ke-3), sedangkan getter digunakan untuk mengambil nilai.

```
1 class Mahasiswa:
2     def __init__(self, name = ""):
3         self._name = name
4
5     # getter method
6     def get_name(self):
7         return self._name
8
9     # setter method
10    def set_name(self, new_name):
11        self._name = new_name
12
13 mahasiswa_if = Mahasiswa()
14
15 #setting the name using setter
16 mahasiswa_if.set_name("Khairani Bilqis")
17
18 #retrieving name using getter
19 print(mahasiswa_if.get_name())
20 print(mahasiswa_if._name)
```

input

Khairani Bilqis  
Khairani Bilqis

Dari contoh tersebut terdapat 2 cara untuk mengakses atribut protected, yaitu menggunakan getter dan memanggil nama atribut secara langsung. Berbeda dengan atribut protected, atribut private tidak dapat diakses secara langsung tanpa menggunakan getter seperti contoh di bawah.

```
1 class Mahasiswa:
2     def __init__(self, name = ""):
3         self.__name = name
4
5     # getter method
6     def get_name(self):
7         return self.__name
8
9     # setter method
10    def set_name(self, new_name):
11        self.__name = new_name
12
13    mahasiswa_if = Mahasiswa()
14
15    #setting the name using setter
16    mahasiswa_if.set_name("Khairani Bilqis")
17
18    #retrieving name using getter
19    print(mahasiswa_if.get_name())
20    print(mahasiswa_if.__name)
```

input

Khairani Bilqis

Traceback (most recent call last):  
File "/home/main.py", line 20, in <module>  
 print(mahasiswa\_if.\_\_name)  
AttributeError: 'Mahasiswa' object has no attribute '\_\_name'

## G. Decorator

Selain menggunakan fungsi setter dan getter tambahan seperti pada contoh sebelumnya, dalam Python kita juga dapat memanfaatkan property decorator untuk mendapatkan hasil yang sama, melalui property decorator ini kita tidak perlu membuat fungsi lagi dengan nama yang berbeda-beda (cukup menggunakan 1 buah nama/variabel). Contoh penggunaan decorator:

```
1 class Mahasiswa:
2     def __init__(self, name = ""):
3         self.__name = name
4
5     @property
6     def name(self): # fungsi mengambil nilai name
7         return self.__name
8
9     @name.setter
10    def name(self, new_name): # fungsi menetapkan nilai
11        self.__name = new_name
12
13    mahasiswa_if = Mahasiswa()
14
15    mahasiswa_if.name = "Khairani Bilqis"
16    print(mahasiswa_if.name)
```

input

Khairani Bilqis

## Resume Pertemuan 3

### A. Abstraksi

Abstraksi adalah konsep OOP dimana model yang dibuat hanya memperlihatkan atribut yang esensial dan menyembunyikan detail-detail yang tidak penting dari user yang berguna untuk mengurangi kompleksitas. User mengetahui apa yang objek lakukan, tapi tidak tahu mekanisme yang terjadi di belakang layar bagaimana.

Abstraksi adalah cara untuk menyembunyikan informasi yang tidak dibutuhkan. Ketika mendefinisikan kelas, sebenarnya sedang membuat Abstrak dari suatu Objek. Kelas merupakan bentuk abstrak atau cetak biru (blue print) dari suatu objek nyata. Wujud nyata suatu kelas dinamakan instance (Objek).

### B. Enkapsulasi (Encapsulation)

Enkapsulasi adalah sebuah metode yang digunakan untuk mengatur struktur kelas (property dan method) dengan cara menyembunyikan alur kerja dari kelas tersebut. Dengan konsep ini, kita dapat “menyembunyikan” property dan method dari suatu kelas agar hanya property dan method tertentu saja yang dapat diakses dari luar kelas. Dengan menghalangi akses dari luar kelas tersebut, elemen penting yang terdapat dalam kelas dapat lebih terjaga, dan menghindari kesalahan jika elemen tersebut diubah secara tidak sengaja.

Enkapsulasi adalah cara menyembunyikan atribut suatu entitas serta metode untuk melindungi informasi dari luar. Untuk membatasi hak akses terhadap property dan method dalam suatu kelas, terdapat 3 jenis access modifier yang terdapat dalam python sebagai berikut:

#### 1. Public Access Modifier

Pada umumnya ketika kita mendeklarasikan suatu variabel atau method, maka itulah public access modifier. Setiap class, variable dan method yang dibuat secara default merupakan public.

#### 2. Protected Access Modifier

Jika suatu variabel dan method dideklarasikan secara protected, maka variable dan method tersebut hanya dapat diakses oleh kelas turunan darinya. Cara mendeklarasikannya dengan menambahkan 1 underscore ( `_` ) sebelum variable atau method.

#### 3. Private Access Modifier

Jika suatu variabel dan method dideklarasikan secara private, maka variable dan method tersebut hanya dapat diakses di dalam kelas itu sendiri, private access merupakan yang paling aman. Dalam mendeklarasikannya, hanya perlu menambahkan double underscore ( `__` ) sebelum nama variable dan methodnya.

Contoh pemakaian ketiga access modifier tersebut dapat dilihat seperti dibawah ini:

```

1 class Laptop:
2     #constructor
3     def __init__(self, merk, pemilik, ram, jenis_processor):
4         self.merk = merk #atribut public
5         self._pemilik = pemilik #atribut protected
6         self.__ram = ram #atribut private
7         self.__jenis_processor = jenis_processor #atribut private
8
9     #fungsi untuk menampilkan merek laptop
10    def merk_laptop(self):
11        print("Merek Laptop : ", self.merk)
12
13    #fungsi untuk menampilkan nama user
14    def akses_pemilik(self):
15        print("Nama User : ", self._pemilik)
16
17    #fungsi untuk menampilkan ram laptop
18    def tampilkan_ram(self):
19        print("RAM Laptop : ", self.__ram, " GB")
20
21    #fungsi untuk menampilkan jenis processor
22    def tampilkan_processor(self):
23        print("Jenis Processor : ", self.__jenis_processor)
24
25    laptop1 = Laptop("Acer", "Khairani", 4, "Intel Core i7-4790 3.6Ghz")
26
27    laptop1.merk_laptop()
28    laptop1.akses_pemilik()
29    #Cara akses private atribut
30    laptop1.tampilkan_ram()
31    laptop1.tampilkan_processor()
32    print(laptop1._pemilik) #Akses atribut protected dari luar kelas
33    print(laptop1.__ram) #Akses atribut private dari luar kelas (akan terjadi error)

```

Jika mengakses atribut private dari luar kelas akan mengeluarkan error sebagai berikut:

```

Traceback (most recent call last):
  File "/home/main.py", line 33, in <module>
    print(laptop1.__ram) #Akses atribut private dari luar kelas (akan terjadi error)
AttributeError: 'Laptop' object has no attribute '__ram'

```

Jika mengakses fungsi tampilkan\_ram dan tampilkan\_processor yang berisikan atribut private maka nilainya dapat dikeluarkan sebagai berikut:

```

Merek Laptop      : Acer
Nama User         : Khairani
RAM Laptop        : 4 GB
Jenis Processor   : Intel Core i7-4790 3.6Ghz
Khairani

```

Setter dan Getter sangat berperan dalam pengaksesan private dan protected atribut baik dengan decorator atau tanpa decorator, contoh tampilkan\_ram dan tampilkan\_processor diatas merupakan implementasi dari getter tanpa decorator (dapat dilihat kembali pada resume pertemuan 2).

## C. Object

### 1. Membuat Instance Object

```
1 class Laptop:
2     #constructor
3     def __init__(self, merk, pemilik, ram, jenis_processor):
4         self.merk = merk #atribut public
5         self._pemilik = pemilik #atribut protected
6         self.__ram = ram #atribut private
7         self.__jenis_processor = jenis_processor #atribut private
8
9     #fungsi untuk menampilkan merek Laptop
10    def merk_laptop(self):
11        print("Merek Laptop : ", self.merk)
12
13    #fungsi untuk menampilkan nama user
14    def akses_pemilik(self):
15        print("Nama User : ", self._pemilik)
16
17    #fungsi untuk menampilkan ram laptop
18    def tampilkan_ram(self):
19        print("RAM Laptop : ", self.__ram, " GB")
20
21    #fungsi untuk menampilkan jenis processor
22    def tampilkan_processor(self):
23        print("Jenis Processor : ", self.__jenis_processor)
```

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

Bentuk umum dari instansiasi objek yaitu:

`nama_objek = kelas_yang_dipanggil(atribut pada kelas)`

Berikut objek dari kelas diatas :

```
laptop1 = Laptop("Acer", "Khairani", 4, "Intel Core i7-4790 3.6Ghz")
```

laptop1 sebagai nama\_objek, Laptop sebagai kelas\_yang\_dipanggil dan atribut yang ada dalam kurung merupakan atribut permintaan dari def \_\_init\_\_ yang ada di kelas.

### 2. Mengakses Atribut Object

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

Sintaks umum dari mengakses atribut objek yaitu:

`nama_objek.atribut`

Berikut cara mengakses atribut object dari kelas Laptop diatas :

```
print(laptop1.merk) #Akan mengeluarkan merk dari laptop1
```

Output yang dihasilkan:

Acer



## Resume Pertemuan 4

### A. Inheritance (Pewarisan)

Inheritance adalah salah satu konsep dasar dari Pemrograman Berbasis Objek (OOP). Pada inheritance, kita dapat menurunkan kelas dari kelas lain untuk hirarki kelas yang saling berbagi atribut dan metode. Berikut merupakan sintaks dasar inheritance:

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

Contoh penerapan inheritance seperti di bawah ini:

```
1 class Manusia:
2     def __init__(self, nama_depan, nama_belakang, umur):
3         self.nama_depan = nama_depan
4         self.nama_belakang = nama_belakang
5         self.umur = umur
6
7     def datadiri(self):
8         print(f"{self.nama_depan} {self.nama_belakang} umur {self.umur} tahun")
9
10 class Mahasiswa(Manusia):
11     pass
12
13 mahasiswa1 = Mahasiswa("Khairani", "Bilqis", 20)
14 mahasiswa1.datadiri()
```

input

Khairani Bilqis umur 20 tahun

#### 1. Inheritance Identik

Inheritance identik merupakan pewarisan yang menambahkan constructor pada class child sehingga class child memiliki constructornya sendiri tanpa menghilangkan constructor pada class parentnya. Metode ini ditandai dengan adanya class child yang menggunakan constructor dan menggunakan kata kunci super().

```
1 class Manusia:
2     def __init__(self, nama_depan, nama_belakang, umur):
3         self.nama_depan = nama_depan
4         self.nama_belakang = nama_belakang
5         self.umur = umur
6
7     def datadiri(self):
8         print(f"{self.nama_depan} {self.nama_belakang} umur {self.umur} tahun")
9
10 class Mahasiswa(Manusia):
11     def __init__(self, nama_depan, nama_belakang, umur, semester):
12         super().__init__(nama_depan, nama_belakang, umur)
13         self.semester = semester
14
15     def datadiri(self):
16         print(f"{self.nama_depan} {self.nama_belakang} umur {self.umur} tahun")
17         print(f"Mahasiswa Semester {self.semester}")
18
19 mahasiswa1 = Mahasiswa("Khairani", "Bilqis", 20, 4)
20 mahasiswa1.datadiri()
```

input

Khairani Bilqis umur 20 tahun  
Mahasiswa Semester 4

## 2. Multiple Inheritance

Python memungkinkan kelas parent menurunkan banyak kelas dan menurunkan kelas turunan ke kelas turunan lain (Bertingkat). Kelas child dapat mewarisi dari banyak kelas parent.

```
1 class Manusia:
2     def __init__(self, nama_depan, nama_belakang, umur):
3         self.nama_depan = nama_depan
4         self.nama_belakang = nama_belakang
5         self.umur = umur
6
7     def datadiri(self):
8         print(f"{self.nama_depan} {self.nama_belakang} Umur {self.umur} tahun")
9
10 class Pelajar:
11     def __init__(self, tahun_masuk, semester):
12         self.tahun_masuk = tahun_masuk
13         self.semester = semester
14
15     def data_diri(self):
16         print(f"{self.nama_depan} {self.nama_belakang} Semester {self.semester}")
17         print(f"{self.nama_depan} {self.nama_belakang} Masuk Tahun {self.tahun_masuk}")
18
19 class Pekerjaan(Manusia, Pelajar):
20     def __init__(self, nama_depan, nama_belakang, umur, tahun_masuk, semester, status_pekerja):
21         Manusia.__init__(self, nama_depan, nama_belakang, umur)
22         Pelajar.__init__(self, tahun_masuk, semester)
23         self.status_pekerja = status_pekerja
24
25     def biodata(self):
26         #return Manusia.datadiri(self) + Pelajar.data_diri(self)
27         print(f"{self.nama_depan} {self.nama_belakang} Pekerjaan {self.status_pekerja}")
28
29 orang1 = Pekerjaan("Khairani", "Bilqis", 20, 2021, 4, "Mahasiswa")
30 orang1.datadiri()
31 orang1.data_diri()
32 orang1.biodata()
```

Output yang dihasilkan :

```
Khairani Bilqis Umur 20 tahun
Khairani Bilqis Semester 4
Khairani Bilqis Masuk Tahun 2021
Khairani Bilqis Pekerjaan Mahasiswa
```

Kita dapat mewarisi dari kelas turunan atau disebut warisan bertingkat. Pewarisan ini dapat dilakukan hingga ke dalaman berapa pun. Dalam kelas turunan dari kelas dasar dan turunannya dapat diwarisi ke dalam kelas turunan yang baru



```

1 class Manusia:
2     def __init__(self, nama_depan, nama_belakang, umur):
3         self.nama_depan = nama_depan
4         self.nama_belakang = nama_belakang
5         self.umur = umur
6
7     def datadiri(self):
8         print(f"{self.nama_depan} {self.nama_belakang} umur {self.umur} tahun")
9
10 class Mahasiswa(Manusia):
11     def __init__(self, nama_depan, nama_belakang, umur, tahun_masuk, semester):
12         super().__init__(nama_depan, nama_belakang, umur)
13         self.tahun_masuk = tahun_masuk
14         self.semester = semester
15
16     def datadiri(self):
17         print(f"{self.nama_depan} {self.nama_belakang} umur {self.umur} tahun")
18         print(f"Mahasiswa Semester {self.semester}")
19         print(f"Mahasiswa Masuk Tahun {self.tahun_masuk}")
20
21 class CalonMahasiswa(Mahasiswa):
22     def __init__(self, nama_depan, nama_belakang, umur, tahun_masuk, semester, asal_sekolah):
23         super().__init__(nama_depan, nama_belakang, umur, tahun_masuk, semester)
24         self.asal_sekolah = asal_sekolah
25
26     def datadiri(self):
27         print(f"{self.nama_depan} {self.nama_belakang} umur {self.umur} tahun")
28         print(f"Calon Mahasiswa akan masuk Semester {self.semester}")
29         print(f"Calon Mahasiswa Masuk Tahun {self.tahun_masuk}")
30         print(f"Calon Mahasiswa Berasal dari {self.asal_sekolah}")
31
32 mahasiswa1 = Mahasiswa("Khairani", "Bilqis", 20, 2021, 4)
33 clnmahasiswa1 = CalonMahasiswa("Zaki", "Abdillah", 17, 2024, 1, "SMA NEGERI 7 BEKASI")
34 clnmahasiswa1.datadiri()

```

Output yang dihasilkan :

```

Zaki Abdillah umur 17 tahun
Calon Mahasiswa akan masuk Semester 1
Calon Mahasiswa Masuk Tahun 2024
Calon Mahasiswa Berasal dari SMA NEGERI 7 BEKASI

```

## B. 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. Polymorphism merupakan kemampuan suatu method untuk bekerja dengan lebih dari satu tipe argumen. Contoh :

```

1 class Jeruk:
2     def tipe(self):
3         print("Buah")
4
5     def warna(self):
6         print("Jingga")
7
8 class Wortel:
9     def tipe(self):
10        print("Sayur")
11
12    def warna(self):
13        print("Jingga")
14
15 def func(obj):
16     obj.tipe()
17     obj.warna()
18
19 obj_jeruk = Jeruk()
20 obj_wortel = Wortel()
21
22 func(obj_jeruk)
23 print()
24 func(obj_wortel)

```

Output yang dihasilkan :

```
Buah
Jingga

Sayur
Jingga
```

### C. 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. Dengan begitu maka method yang ada parent class tidak berlaku dan yang akan dijalankan adalah method yang terdapat di child class. Contoh :

```
1 class BangunRuang:
2     def cetak(self):
3         print("Ini bangun ruang")
4
5 class Tabung(BangunRuang):
6     def cetak(self):
7         print("Ini tabung")
8
9 ruang1 = Tabung()
10 ruang1.cetak()
```

Ini tabung

Pada contoh di atas terdapat *parent class* yaitu class BangunRuang dengan method cetak dan terdapat *child class* yaitu class Tabung yang memiliki method yang sama dengan method pada *parent class*. Dengan begitu ketika melakukan instansiasi objek *child class* dan memanggil method cetak() maka yang akan dipanggil ialah method pada *child class*, method pada *parent class* tidak berlaku.

### D. Overloading

Metode overloading mengizinkan sebuah class untuk memiliki sekumpulan fungsi dengan nama yang sama dan argumen yang berbeda. Akan tetapi, Python tidak mengizinkan pendeklarasian fungsi (baik pada class ataupun tidak) dengan nama yang sama. Hal itu menyebabkan implementasi overloading pada python menjadi “tricky”.

Secara umum overloading memiliki beberapa signature, yaitu jumlah argumen, tipe argumen, tipe keluaran dan urutan argumen. Karena python adalah Bahasa pemrograman yang bersifat duck typing (dynamic typing), overloading secara tidak langsung dapat diterapkan. Contoh :

```
1 class Angka:
2     def __init__(self, angka):
3         self.angka = angka
4
5     def __add__(self, other):
6         return Angka(self.angka + other.angka)
7
8     def __str__(self):
9         return f"Objek Angka dengan nilai {self.angka}"
10
11 bil1 = Angka(15)
12 bil2 = Angka(30)
13
14 print(bil1)
15 print(bil2)
16 print(bil1 + bil2)
```

Objek Angka dengan nilai 15  
Objek Angka dengan nilai 30  
Objek Angka dengan nilai 45