

**LAPORAN HASIL PEMBELAJARAN
PEMROGRAMAN BERORIENTASI OBJEK (PBO)**

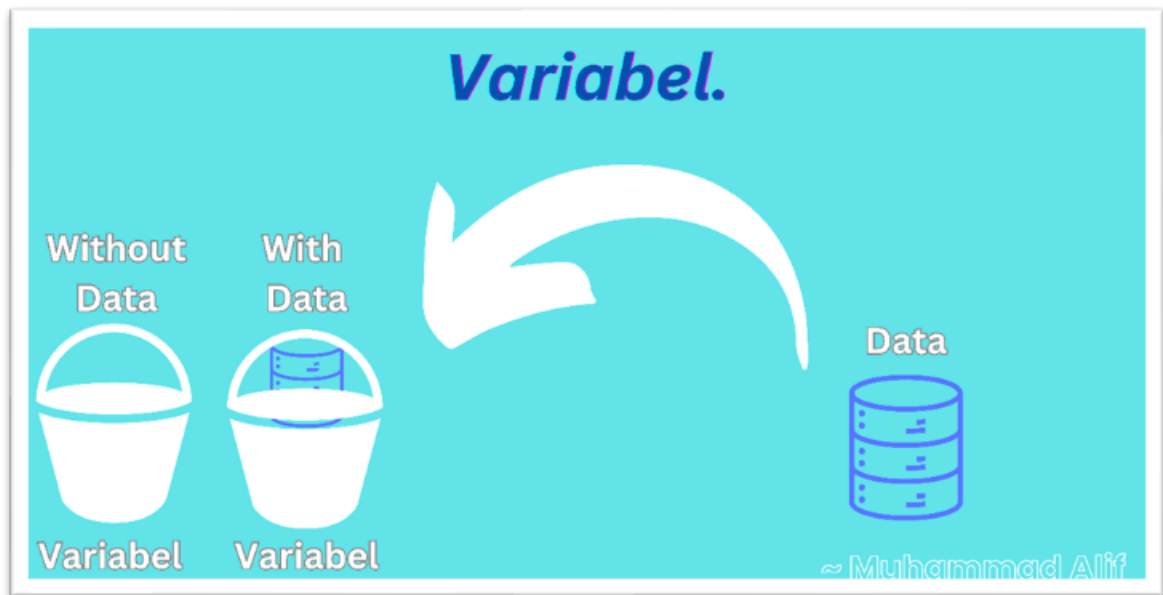
“DASAR-DASAR PEMROGRAMAN PYTHON”



2209106127 Muhammad Alif

**PROGRAM STUDI INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS MULAWARMAN
2024**

1. Variabel



Variabel, salah satu fundamental dalam python atau Bahasa pemrograman lain untuk menyimpan data sementara untuk informasi dan manipulasi data. Biasanya diawali dengan memberikan nama variable setelah itu value/datanya.

A. Cara Membuat Variabel

```
ini_variabel = aku_adalah_value  
ini_variabel2 = #tanpa value
```

- Catatan : *Ingat! Dalam membuat sebuah variabel tanpa tanda **sama dengan** (=) , variabel tidak akan tercipta.*

B. Aturan Membuat Variabel

Dalam membuat sebuah variabel **tidak bisa sembarangan** bagaikan melukis di atas kanvas kosong tanpa sketsa. Ada beberapa alasan yang bisa kamu ketahui kenapa(?).

I. Penamaan Yang Jelas

Misal, kamu ingin membuat sebuah variabel “panjang” tetapi kamu menuliskannya dengan nama “p” jika dibuat seperti itu kamu akan kebingungan karena ada variabel lain juga yang relevan menggunakan “p”, Contoh:

```
p1 = value
p2 = value
```

Dari contoh di atas sudah jelas penamaan variabel bisa membuat bingung orang lain, karena tidak tahu digunakan apa. Jadi, seharusnya dibuat seperti ini untuk memperjelas sebuah nama variabel:

```
panjang1 = 10
panjang2 = 1
```

II. Memiliki Arti Jelas dan Mudah Diingat

Saat membuat nama variabel, jika kamu membuat program menghitung aritmatika penjumlahan. Dan menggunakan variabel “x” dan “y” yang mempresentasikan *bilangan pertama dan kedua*.

Bisa saja dari variabel tersebut kamu bisa lupa, maka dari itu harus *deskriptif* dalam membuat nama variabel seperti:

```
bilangan_pertama = 10
bilangan_kedua = 15

jumlah = bilangan_pertama + bilangan_kedua
```

Dengan begini akan lebih jelas, dan tidak membuat orang lain bingung saat membaca program yang kamu miliki.

III. Penulisan Kiri ke Kanan

Dalam penulisan variabel sudah jelas cara definisikannya adalah *[nama_variabel = value]* jika tidak menggunakan format tersebut maka variabel tidak tercipta.

```
# Benar
nama_lengkap = "Muhammad Alif"
usia = 25

# Salah
= "Muhammad Alif" nama_lengkap
25 = usia
```

IV. Avoid Python Keywords

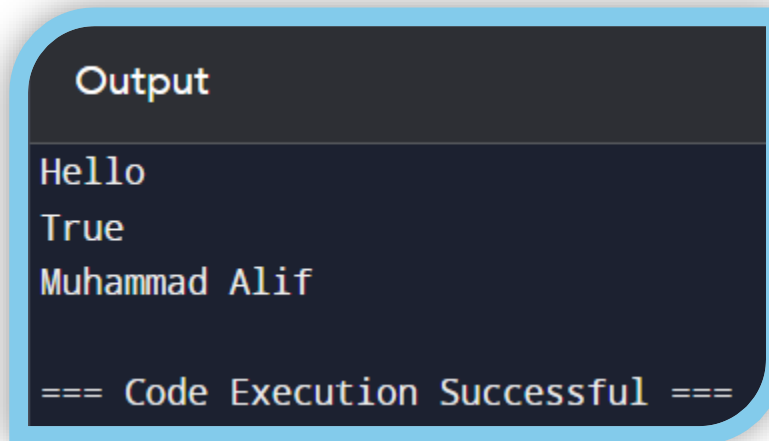
Maksudnya adalah, hindari penggunaan keyword atau kata kunci yang dimiliki oleh **python** seperti menggunakan “if”, “else”, “for”, “while” dan sebagainya. Berikut contohnya:

```
def = "Hello"  
while = 0  
if = 10
```

- Apakah ada solusi jika ingin tetap menggunakan kata kunci tersebut? Ya, ada. Kamu cukup tambahkan **kata lanjutan** atau simbol **underline** (_).

```
def_ = "Hello" # Menggunakan Underline  
ifPlayed = True # Menggunakan Kata Lanjutan  
while_nama = "Muhammad Alif" # Menggunakan Kombinasi Under dan Kata Lanjutan  
  
print(def_)  
print(ifPlayed)  
print(while_nama)
```

- **Output:**



```
Output  
  
Hello  
True  
Muhammad Alif  
  
=== Code Execution Successful ===
```

V. Dimulai dengan Huruf atau Garis Bawah

Saat membuat sebuah variabel hal terpenting adalah penulisan awalnya adalah harus menggunakan **huruf** atau **underline** (_). Seperti contoh berikut:

```
nama_depan = value #(benar)  
_umur = value #(benar)  
nilaiUjian = value #(benar)  
1skor = value #(salah)  
$username = value #(salah)
```

Maka dari itu, hal-hal dibawah ini dilarang digunakan saat membuat variabel:

- Angka
- Simbol
- Spasi

Aturan ini akan membantu kita membangun fondasi kode yang kuat, mudah dipahami, dan bebas dari kesalahan. Memulai variabel dengan huruf atau garis bawah adalah langkah kecil yang memiliki dampak besar.

VI. Case Sensitive

Dalam penulisan fundamental python, terutama dibagian variabel memiliki aturan case-sensitive. Artinya perbedaan besar kecil huruf pada nama variabel memiliki arti yang berbeda.

Jadi jika kita membuat sebuah variabel “**nama**” dan “**Nama**” kedua variabel tersebut disebut berbeda. Saya memiliki analogi yang bisa kalian pahami, jika ada seseorang kembar dengan nama Arif dan Arifin meskipun mirip tetapi mereka individu yang berbeda.

Contoh penggunaan case-sensitive:

```
panjang = 10
LEBAR = 5
lebar = 15

luas = panjang * LEBAR

print("Luas persegi panjang:", luas)
```

Pada contoh diatas, akan menghasilkan output **50** (hasil akumulasi dari variabel *LEBAR*), kenapa? Karena variabel tersebut berbeda seperti yang sudah dijelaskan sebelumnya perbedaan besar dan kecil huruf akan dianggap variabel *berbeda*.

VII. Multiple Variable One Line

Kalian bisa membuat sebuah beberapa variabel dalam satu baris, tetapi memiliki persyaratannya adalah bagian kiri (*variabel*) dan kanan (*value*) jumlahnya sama. Misalnya seperti berikut :

```
a,b,c,d = 1+1,True,3,"Hello Variabel"
print(a,c,b,d)
```

- Pada saat dirunning programnya, akan menghasilkan **output**:

```
Output
2 3 True Hello Variabel
== Code Execution Successful ==
```

Penjelasan:

```
a,b,c,d = 1+1,True,3,"Hello Variabel"
V1 V2 V3 V4 VAL1 VAL2 VAL3 VAL4
print(a,c,b,d)
```

Berikut penjelasan kode diatas,

Kotak Hijau : variabel, dalam satu baris tersebut ada 4 variabel yaitu [a,b,c,d]

Kotak Merah : value, karena variabelnya 4 maka valuenya juga seharusnya 4.

Variabel	Value
a	1+1 (Integer)
b	True (Boolean)
c	1 (Integer)
d	"Hello Variabel" (String)

Contoh Kesalahan :

Ada beberapa contoh aturan yang salah jika kalian menerapkan konsep variabel ini, yaitu:

```
e,f,g = 1,2
# Pada sisi kiri 3 variabel
# Pada sisi kanan 2 value/nilai
# Artinya : variabel "g" tidak punya nilai.
# Salah.

g,h = 1,2,3
# Pada sisi kiri 2 variabel
# Pada sisi kanan 3 value/nilai
# Artinya : value "3" tidak punya variabel.
# Salah.
```

C. Menghapus Variabel

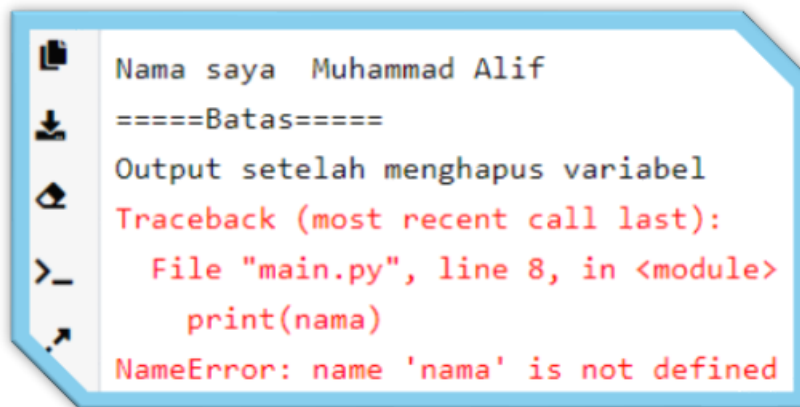
Fitur ini memungkinkan untuk menghapus sebuah variabel beserta nilai/valuenya. Dengan menggunakan fungsi `del(variabel)`. Berikut contoh penggunaannya:

```
nama = "Muhammad Alif"
print("Nama saya ", nama)

print("====Batas====")

del(nama)
print("Output setelah menghapus variabel")
print(nama)
```

➤ Sehingga menghasilkan **output** seperti ini:



D. Mengubah Variabel

Dalam variabel kita juga bisa mengubah value/nilai dari variabel, dengan cara mendeklarasikan kembali variabel yang ingin di ubah. Dalam konteks mengubah sebuah variabel sangat banyak pembahasan, misalnya konversi tipe data(*akan dibahas*), menambahkan nilai, dan lain lain.

I. Mengubah Isi Variabel (Value)

Berikut contoh dalam mengubah value dari sebuah variabel:

```
a = "String Pertama"
print(a)

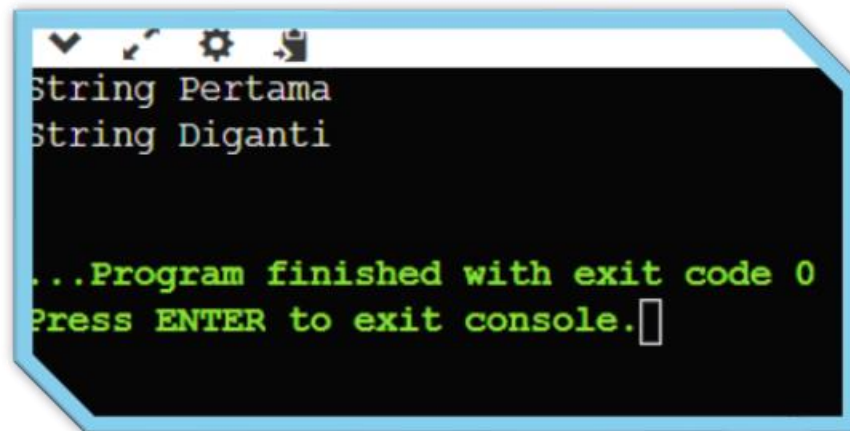
a = "String Diganti"
print(a)
```

Penjelasan :

Pada program tersebut awalnya memiliki sebuah variabel `a` yang memiliki value “String Pertama” saat di ubah menjadi “String Diganti”

Cara definisinya mirip seperti membuat variabel baru, tetapi dalam kasus ini mengganti sebuah nilai dari sebuah variabel yang artinya kita harus memanggil atau deklarasi nama variabel yang sama dengan yang ingin diganti.

- Hasil **output** dari kode tersebut adalah :



```
String Pertama
String Diganti

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

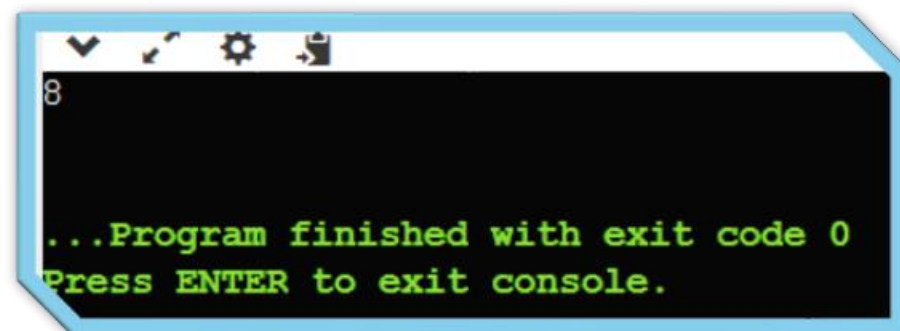
- Catatan : Kalian bisa menggunakan value lain, tidak hanya String, tetapi bisa juga tipe data lainnya. Seperti angka, boolean, float, dll.

II. Menambahkan Isi Variabel (Value)

Menambahkan nilai pada variabel yang sudah ada, dan mengubah valuenya yang berbeda dari nilai awalnya. Berikut contoh kodenya:

```
x = 5 # Nilai x Awal
x = x + 3 # Nilai x Setelah ada proses penjumlahan
print(x)
```

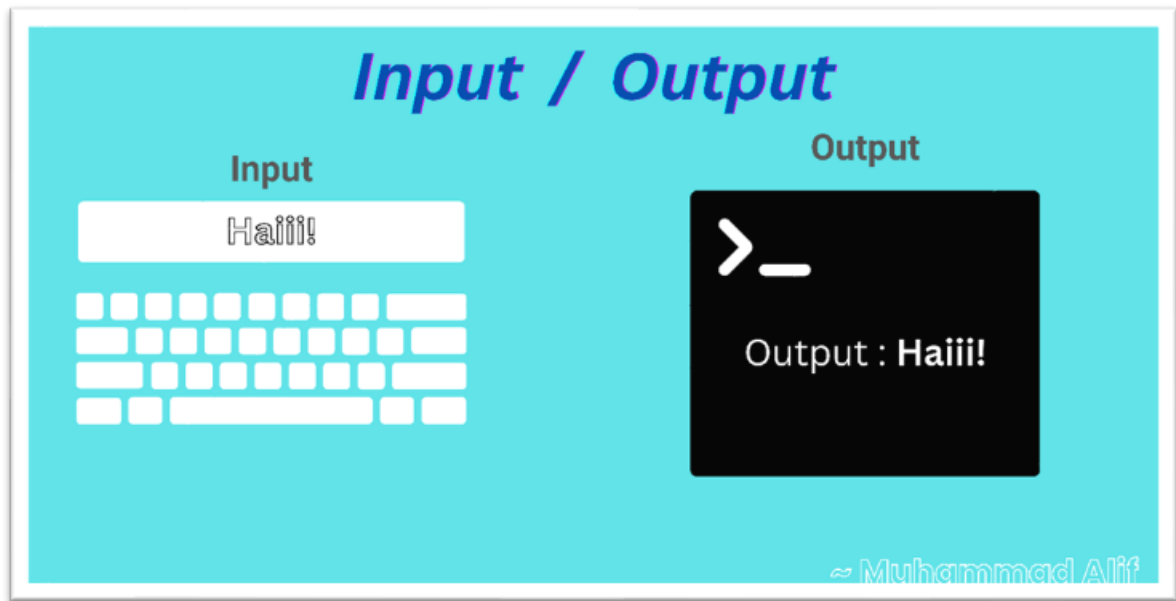
- Dan berikut hasil **output** dari kode tersebut:



```
8

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


2. I/O Python



A. Input

Fungsi ini sangat berguna untuk pengguna dan lebih interaktif juga ke sistem yang dibangun, karena kita bisa memberi inputan melalui keyboard dan menampilkan hasil inputan tersebut ke layar terminal atau program selama valid. Lainnya fungsi ini seperti kita berbicara ke sebuah sistem dengan cara memberikan data atau informasi.

Sebelum memulai mari kita bahas apa itu **Input**. *Input* adalah sebuah masukan (*interaksi dari pengguna ke sistem*) yang bisa diolah menjadi data atau informasi nantinya. Biasanya menggunakan keyboard sebagai alat masukan untuk interaksinya.

I. Cara Membuat Inputan

Untuk membuat sebuah inputan kita perlu fungsi `input()`, berfungsi dengan cara menerima satu argumen opsional (dalam bentuk string) yang akan ditampilkan di layar sebelum pengguna memberikan input. Berikut contoh implementasi :

```
input()
input("Dengan Teks : ")
inputan = input("Dengan Variabel : ")
```

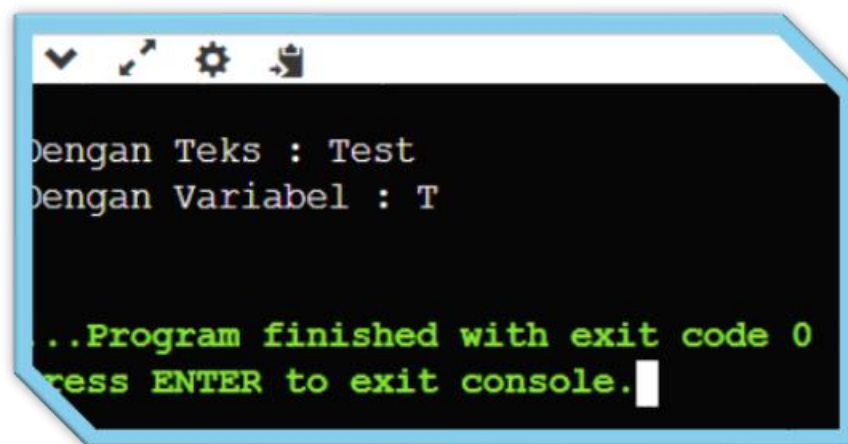
Penjelasan:

Pada line pertama atau kode baris pertama, itu hanya inputan biasa yang hanya menampilkan interface apa yang diketik oleh pengguna.

Pada line kedua ditambahkan sebuah teks tambahan untuk memperjelas inputan untuk pengguna nantinya.

Pada line ketiga inputan tersebut dimasukkan kedalam sebuah variabel sebagai value, dari bentuk fungsi tersebut nantinya baru kita bisa memberikan output kepada pengguna.

Output :

A screenshot of a console window with a black background and a blue border. The window contains the following text: "Dengan Teks : Test", "Dengan Variabel : T", and "...Program finished with exit code 0". Below the last line, there is a prompt "Press ENTER to exit console." followed by a white cursor. The window has standard OS icons (checkmark, arrows, gear, and a trash can) in the top left corner.

```
Dengan Teks : Test
Dengan Variabel : T

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

II. Macam-Macam Jenis Inputan

Inputan pada python juga memiliki beberapa jenis, maksudnya adalah inputan tersebut hanya menerima tipe data tertentu. Misal inputannya hanya menerima berupa angka, huruf, angka desimal, dll.

a. String

Pada jenis inputan ini hanya menerima tipe data string saja. Menggunakan fungsi **str(input())**

Contoh :

```
nama = str(input("Masukkan nama Anda: "))
```

b. Integer

Inputan yang hanya menerima inputan numerik, jika non-numerik maka akan menampilkan error. Menggunakan fungsi **int(input())**

Contoh :

```
umur = int(input("Masukkan usia Anda: "))
```

c. Float

Inputan yang hanya menerima inputan angka desimal, kurang lebih sama dengan integer jika ada non-numerik atau non-angka desimal maka akan terjadi error. Menggunakan fungsi **float(input())**

Contoh :

```
nilai_ujian = float(input("Masukkan nilai ujian Anda: "))
```

d. Boolean

Pada inputan ini hanya menerima inputan berupa True dan False saja. Menggunakan fungsi **bool(input())**

Contoh :

```
jawaban = input("Apakah Anda suka programming? (ya/tidak): ")
suka_programming = jawaban == "ya" #Boolean
if suka_programming: # True
    pass
else: #False
    pass
```

Penjelasan:

pass, merupakan fungsi sementara jadi jika dikompilasi tidak akan terjadi apapun.

B. Output

Output biasa disebut bagaimana program itu “berbicara” kepada pengguna untuk memberikan informasi atau data. Output biasa menggunakan fungsi **print()**.

Setelah pembahasan inputan, disini kita akan mencoba menampilkan hasil inputan dari pengguna ke program, berikut contohnya:

```
nama = str(input("Nama saya : "))
umur = int(input("Umur saya : "))

print("Nama Saya : " + nama + " Umur saya adalah " , umur)
```

Penjelasan :

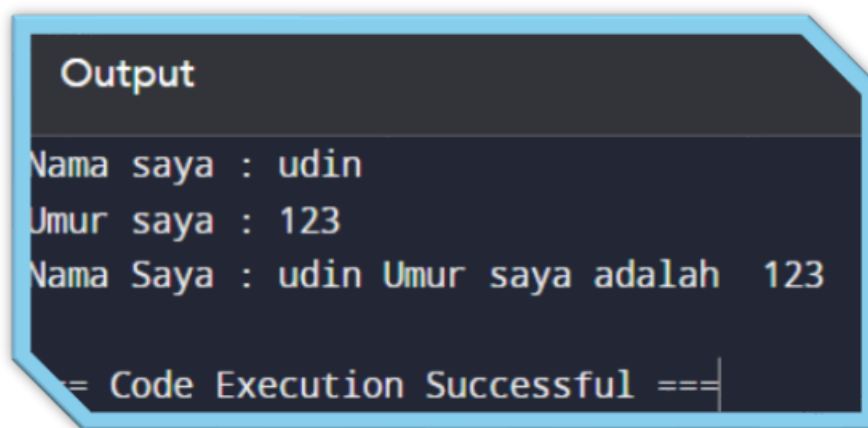
Jadi, kita memiliki 2 inputan yaitu string untuk memasukan nama, integer untuk memasukkan umur. Pada masing masing inputan sudah diberikan format tipe data inputannya.

Seperti, nama diberikan fungsi str, umur menggunakan int untuk menerima inputan numerik. Pada kode baris 1 dan 2 kita memiliki variabel nama dan umur, saat user sudah melakukan inputan, inputan tersebut akan menjadi sebuah *value* untuk kedua variabel tersebut.

Setelah itu diberikan outputnya, menggunakan fungsi **print()**

Output:

:



```
Output
Nama saya : udin
Umur saya : 123
Nama Saya : udin Umur saya adalah 123
== Code Execution Successful ==
```

I. Jenis Jenis Output :**a. String Formatting****Penjelasan :**

%s = Berfungsi sebagai placeholder untuk nilai string nantinya dari variabel.

% = Operator untuk memasukkan nilai variabelnya

(variabel) = variabel yang akan dimasukkan ke placeholder %s

```
nama = "Muhammad Alif"
umur = 20
print("Nama: %s, Umur: %s tahun" % (nama, umur))
```

- Catatan : Saat memberikan placeholder %s, dan variabel pada string ("") itu seperti berurutan, jadi urutannya kiri ke kanan. Dan akan tampil **sesuai dengan posisi variabel yang di masukkan.**

b. String Interpolation

Sedikit mirip dengan formatting sebelumnya, tetapi ini ada tambahan metode pada stringnya yaitu `.format(variabel)` dan placeholdernya atau hasil dari variabel tersebut akan ditampilkan pada posisi `{}`

```
nama = "Muhammad Alif"
umur = 20
print("Nama: {}, Umur: {} tahun".format(nama, umur))
```

- Catatan : Saat memberikan placeholder `{}`, dan variabel pada string (`"`) itu seperti berurutan, jadi urutannya kiri ke kanan. Dan akan tampil **sesuai dengan posisi variabel yang di masukkan**.

c. Literal String Interpolation

Sedikit berbeda dengan formatting lainnya, ini bisa langsung memanggilnya tanpa menggunakan metode seperti sebelumnya. Cukup memasukkan nama variabel pada `{variabel}` value dari variabel tersebut akan ditampilkan.

Tetapi pada awal fungsi `print()` diberikan huruf `(f)` untuk memanggil variabel didalam string langsung dengan menempatkan di `{}`.

```
nama = "Muhammad Alif"
umur = 20
print(f>Nama: {nama}, Umur: {umur} tahun")
```

3. Tipe Data

Tipe Data, python menggunakan istilah "tipe data" untuk menggambarkan jenis nilai atau informasi yang dapat disimpan dan diolah oleh program. Setiap nilai yang digunakan memiliki tipe data yang menentukan sifat dan perilaku nilai.

Tipe Data	Contoh	Penjelasan
Boolean	<code>True</code> atau <code>False</code>	Menyatakan benar <code>True</code> yang bernilai <code>1</code> , atau salah <code>False</code> yang bernilai <code>0</code>
String	<code>"Ayo belajar Python"</code>	Menyatakan karakter/kalimat bisa berupa huruf angka, dll (diapit tanda <code>"</code> atau <code>'</code>)

Tipe Data	Contoh	Penjelasan
Integer	25 atau 1209	Menyatakan bilangan bulat
Float	3.14 atau 0.99	Menyatakan bilangan yang mempunyai koma atau desimal
Hexadecimal	9a atau 1d3	Menyatakan bilangan dalam format heksa (bilangan berbasis 16)
Complex	1 + 5j	Menyatakan pasangan angka real dan imajiner
List	['xyz', 786, 2.23]	Data untaian yang menyimpan berbagai tipe data dan isinya bisa diubah-ubah
Tuple	('xyz', 768, 2.23)	Data untaian yang menyimpan berbagai tipe data tapi isinya tidak bisa diubah
Dictionary	{'nama': 'adi','id':2}	Data untaian yang menyimpan berbagai tipe data berupa pasangan penunjuk dan nilai

■ Contoh Penggunaan

a. String

```
nama = "Muhammad Alif"
print("Nama :", nama)
```

b. Integer

```
angka1 = 25
angka2 = 0
angka3 = -1

print("Nilai Angka 1:", angka1)
print("Nilai Angka 2:", angka2)
print("Nilai Angka 3:", angka3)
```

c. Boolean

```
benar = True
salah = False

print("Benar:", benar)
print("Salah:", salah)
```

d. Float

```
pi = 3.14

print("Nilai pi:", pi)
```

e. List

```
c_list = [1, 'teks', True, 3.14]

print("List :", c_list)
```

f. Tuple

```
bio = (20, 'Muhammad Alif', True)

print("Informasi:", info)
```

g. Dictionary

```
siswa = {'nama': 'Muhammad Alif', 'umur': 20}

print("Data siswa:", siswa)
print("Data Siswa Nama :", siswa["nama"])
print("Data Siswa Umur :", siswa["umur"])
```

h. Set

```
bilangan = {1, 2, 3, 4, 5}
huruf = {'a', 'b', 'c', 'd', 'e'}
boolean = {True, False, True}

print("Set bilangan:", bilangan)
print("Set huruf:", huruf)
print("Set boolean:", boolean)
```

4. Operasi Dasar



Operasi Dasar pada Python mengacu pada kumpulan tindakan atau manipulasi data yang dapat dilakukan dalam bahasa pemrograman Python. Ini mencakup berbagai operasi, seperti penjumlahan, pengurangan, perkalian, pembagian, pemangkatan, serta operasi perbandingan, penugasan, dan logika. **Operand**, yang merupakan identitas atau nilai yang dioperasikan oleh operator, memainkan peran penting dalam operasi ini. Dengan menggunakan *operand*, operator dapat melakukan tindakan tertentu, seperti membandingkan nilai, melakukan perhitungan matematika, atau menentukan alur program berdasarkan logika yang diterapkan.

A. Operasi Arithmetic (Aritmatika)

Operator matematika dalam bahasa pemrograman Python adalah simbol atau tanda yang digunakan untuk melakukan berbagai operasi aritmatika. Mulai dari penjumlahan, pengurangan, perkalian, pembagian, hingga pemangkatan dan operasi lainnya. Dalam Python, terdapat beberapa operator matematika dasar, seperti:

Jenis Aritmatika	Penjelasan	Contoh
Penjumlahan	Digunakan untuk menjumlahkan kedua operand	$a + b$
Pengurangan	Digunakan untuk mengurangi operand pertama dengan kedua	$a - b$
Perkalian	Digunakan untuk mengalikan kedua operand	$a * b$

Pembagian	Digunakan untuk membagi operand pertama dengan kedua	a / b
Modulus	Mengembalikan sisa pembagian operand pertama dengan kedua	a % b
Perpangkatan	Menghitung hasil pangkat operand pertama ke operand kedua	a ** b
Pembagian Bulat	Mengembalikan hasil pembagian bilangan bulat	a // b

Berikut contoh penggunaan dari operasi aritmatika :

```
# Variabel
angka1 = 10
angka2 = 5

# Penjumlahan
hasil_penjumlahan = angka1 + angka2
print("Hasil penjumlahan:", hasil_penjumlahan)

# Pengurangan
hasil_pengurangan = angka1 - angka2
print("Hasil pengurangan:", hasil_pengurangan)

# Perkalian
hasil_perkalian = angka1 * angka2
print("Hasil perkalian:", hasil_perkalian)

# Pembagian
hasil_pembagian = angka1 / angka2
print("Hasil pembagian:", hasil_pembagian)

# Modulus
hasil_modulus = angka1 % angka2
print("Hasil modulus:", hasil_modulus)

# Perpangkatan
hasil_perpangkatan = angka1 ** angka2
print("Hasil perpangkatan:", hasil_perpangkatan)

# Pembagian Bulat
hasil_pembagian_bulat = angka1 // angka2
print("Hasil pembagian bulat:", hasil_pembagian_bulat)
```

Output :

```
Hasil penjumlahan: 15
Hasil pengurangan: 5
Hasil perkalian: 50
Hasil pembagian: 2.0
Hasil modulus: 0
Hasil perpangkatan: 100000
Hasil pembagian bulat: 2

Code Execution Successful ===
```

B. Operasi Comparion (Perbandingan)

Operasi Perbandingan dalam Python digunakan untuk membandingkan dua nilai atau ekspresi dan menghasilkan nilai kebenaran (True atau False) berdasarkan hasil perbandingannya. Berikut adalah tabel operasi perbandingan beserta penjelasannya:

Jenis Perbandingan	Penjelasan	Contoh
==	Digunakan untuk memeriksa apakah dua nilai sama.	a == b
!=	Digunakan untuk memeriksa apakah dua nilai tidak sama.	a != b
>	Digunakan untuk memeriksa apakah nilai sebelah kiri lebih besar dari nilai sebelah kanan.	a > b
<	Digunakan untuk memeriksa apakah nilai sebelah kiri lebih kecil dari nilai sebelah kanan.	a < b
>=	Digunakan untuk memeriksa apakah nilai sebelah kiri lebih besar dari atau sama dengan nilai sebelah kanan.	a >= b
<=	Digunakan untuk memeriksa apakah nilai sebelah kiri lebih kecil dari atau sama dengan nilai sebelah kanan.	a <= b

Berikut contoh penggunaan dari operasi perbandingan :

```
# Variabel
a = 10
b = 5

# Cek dulu apakah a sama dengan b
sama_dengan = a == b
print("Apakah a sama dengan b?", sama_dengan)

# Cek dulu apakah a tidak sama dengan b
tidak_sama_dengan = a != b
print("Apakah a tidak sama dengan b?", tidak_sama_dengan)

# Cek dulu apakah a lebih besar dari b
lebih_besar = a > b
print("Apakah a lebih besar dari b?", lebih_besar)

# Cek dulu apakah a lebih kecil dari b
lebih_kecil = a < b
print("Apakah a lebih kecil dari b?", lebih_kecil)

# Cek dulu apakah a lebih besar dari atau sama dengan b
lebih_besar_sama_dengan = a >= b
print("Apakah a lebih besar dari atau sama dengan b?",
lebih_besar_sama_dengan)

# Cek dulu apakah a lebih kecil dari atau sama dengan b
lebih_kecil_sama_dengan = a <= b
print("Apakah a lebih kecil dari atau sama dengan b?",
lebih_kecil_sama_dengan)
```

Output :

```
Apakah a sama dengan b? False
Apakah a tidak sama dengan b? True
Apakah a lebih besar dari b? True
Apakah a lebih kecil dari b? False
Apakah a lebih besar dari atau sama dengan b? True
Apakah a lebih kecil dari atau sama dengan b? False

== Code Execution Successful ==
```

C. Operasi Assignment (Penugasan)

Operasi Assignment dalam Python mirip dengan memberikan nilai kepada variabel. Misalnya, kita bisa memberikan nilai 10 kepada variabel a dengan menggunakan operator = seperti ini: a = 10. Ini artinya, variabel a sekarang memiliki nilai 10 dan bisa digunakan dalam program. Berikut beberapa jenis assignmentnya :

Jenis Assignment	Penjelasan	Contoh
=	digunakan untuk memberikan nilai dari sebelah kanan ke variabel di sebelah kiri.	a = 10
+=	digunakan untuk menambahkan nilai dari sebelah kanan ke variabel di sebelah kiri.	a += 5
-=	digunakan untuk mengurangi nilai dari sebelah kanan ke variabel di sebelah kiri.	a -= 3
*=	digunakan untuk mengalikan nilai dari sebelah kanan dengan variabel di sebelah kiri.	a *= 2
/=	digunakan untuk membagi nilai dari variabel di sebelah kiri dengan nilai di sebelah kanan.	a /= 4
%=	digunakan untuk mengambil sisa bagi antara nilai variabel di sebelah kiri dengan nilai di sebelah kanan.	a %= 3
//=	digunakan untuk melakukan pembagian bulat antara nilai variabel di sebelah kiri dengan nilai di sebelah kanan.	a //= 3
**=	digunakan untuk mengangkat nilai variabel di sebelah kiri dengan nilai di sebelah kanan.	a **= 2

Contoh penggunaan untuk operasi penugasan :

```
# Operator =
a = 10 # Memberikan nilai 10 kepada variabel a

# Operator +=
a += 5 # Menambahkan nilai 5 ke variabel a
print("Nilai a setelah ditambah 5:", a)

# Operator -=
a -= 3 # Mengurangkan nilai 3 dari variabel a
print("Nilai a setelah dikurangkan 3:", a)

# Operator *=
a *= 2 # Mengalikan nilai a dengan 2
print("Nilai a setelah dikalikan 2:", a)

# Operator /=
a /= 4 # Membagi nilai a dengan 4
print("Nilai a setelah dibagi 4:", a)

# Operator %=
a %= 3 # Mengambil sisa bagi nilai a dengan 3
print("Sisa bagi a dengan 3:", a)

# Operator //=
a //= 3 # Melakukan pembagian bulat nilai a dengan 3
print("Pembagian bulat a dengan 3:", a)

# Operator **=
a **= 2 # Meningkatkan nilai a dengan 2
print("Nilai a pangkat 2:", a)
```

Output :

```
Nilai a setelah ditambah 5: 15
Nilai a setelah dikurangkan 3: 12
Nilai a setelah dikalikan 2: 24
Nilai a setelah dibagi 4: 6.0
Sisa bagi a dengan 3: 0.0
Pembagian bulat a dengan 3: 0.0
Nilai a pangkat 2: 0.0
```

Code Execution Successful ==

D. Operasi Membership (Keanggotaan)

Operasi keanggotaan (membership) dalam Python digunakan untuk menguji apakah suatu nilai atau objek terdapat dalam sebuah urutan seperti list, tuple, atau string. Berikut adalah tabel operasi keanggotaan beserta penjelasannya:

Jenis Membership	Penjelasan	Contoh
in	Melakukan cek apakah nilai terdapat dalam urutan	5 in [1, 2, 3, 4, 5]
not in	Melakukan cek apakah nilai tidak terdapat dalam urutan	6 not in [1, 2, 3]

Contoh penggunaan operasi membership :

```
# Contoh jika menggunakan angka yang ada di dalam list
menggunakan "in"
print(5 in [1, 2, 3, 4, 5])

# Contoh jika menggunakan angka yang tidak ada di dalam list
menggunakan "not in"
print(6 not in [1, 2, 3, 4, 5])

# Contoh jika cek angka yang ada didalam list menggunakan "not
in"
print(5 not in [1,2,3,4,5])
```

Output :

```
True
True
False

== Code Execution Successful ==
```

E. Operasi Identity (Identitas)

Operasi identitas (is dan is not) digunakan untuk memeriksa apakah dua variabel memiliki identitas yang sama atau berbeda. Identitas dalam konteks ini merujuk pada lokasi memori tempat objek disimpan. Jika dua variabel memiliki identitas yang sama, itu berarti keduanya merujuk ke objek yang sama di dalam memori.

Jenis Identitas	Penjelasan	Contoh	Hasil
is	True jika dua variabel memiliki identitas yang sama	x is y	False
is not	True jika dua variabel memiliki identitas yang berbeda	x is not y	True

Berikut contoh penggunaan operasi :

```
x = 5
y = 5
z = 6

# Menguji apakah dua variabel memiliki identitas yang sama
print(x is y)
print(x is z)

# Menguji apakah dua variabel memiliki identitas yang berbeda
print(x is not z)
print(x is not y)
```

Output :

```
True
False
True
False

== Code Execution Successful ==
```

Penjelasan Singkat :

Variabel x dan y memiliki nilai yang sama, yaitu 5. Karena angka kecil seperti 5 di-cache oleh Python dalam memori, keduanya merujuk ke objek yang sama. Itu sebabnya, operasi is mengembalikan nilai True.

Sementara itu, variabel x memiliki nilai 5, sedangkan variabel z memiliki nilai 6. Karena kedua nilai ini berbeda, keduanya merujuk ke objek yang berbeda di memori. Sehingga, operasi is not mengembalikan nilai True.

F. Operasi Bitwise

Operasi bitwise memanipulasi bit-bit individu dalam representasi biner dari operandnya. Ini mencakup operasi seperti AND, OR, XOR, shift left, dan shift right. Berikut tabel operasi bitwise

Jenis Bitwise	Penjelasan	Contoh
& (AND)	Menghasilkan 1 jika kedua operandnya memiliki nilai 1 atau True.	a & b
(OR)	Menghasilkan 1(True) jika salah satu kedua nilai operand memiliki angka 1 atau True.	a b
^ (XOR)	Menghasilkan 1(True) jika salah satu kedua nilai operandnya memiliki angka 1 tetapi berbeda dengan OR yang jika keduanya tetap ada angka 1 tetap menghasilkan 1, pada XOR harus salah satunya bernilai 1 (True).	a ^ b
>> (Shift Right)	Menggeser ke kanan dari suatu nilai bit sejumlah dari nilai (n)	a >> n
<< (Shift Left)	Menggeser ke kiri dari suatu nilai bit sejumlah dari nilai (n)	a << n
~ (NOT)	Mengembalikan atau Versi Terbalik dari sebuah bit. Misal, 1 = 0 dan 0 = 1.	~a

Contoh penggunaan operasi bitwise :

```
a = 5    # Representasi biner: 0101
b = 3    # Representasi biner: 0011

# AND bitwise
hasil_and = a & b    # 0101 & 0011 = 0001 (1 dalam desimal)
print(hasil_and)

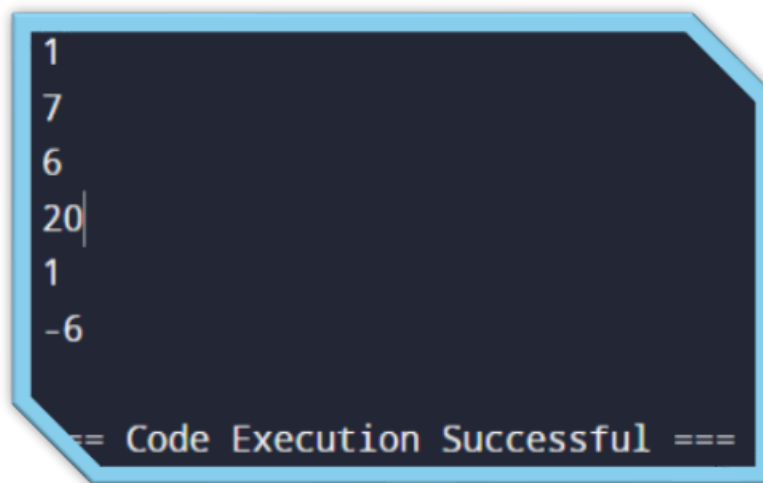
# OR bitwise
hasil_or = a | b     # 0101 | 0011 = 0111 (7 dalam desimal)
print(hasil_or)

# XOR bitwise
hasil_xor = a ^ b    # 0101 ^ 0011 = 0110 (6 dalam desimal)
print(hasil_xor)
```



```
# Shift kiri
hasil_shift_left = a << 2 # 0101 geser kiri 2 bit = 10100 (20
dalam desimal)
print(hasil_shift_left)
# Shift kanan
hasil_shift_right = a >> 2 # 0101 geser kanan 2 bit = 0001 (1
dalam desimal)
print(hasil_shift_right)
# NOT bitwise
hasil_not = ~a # ~0101 = 1010 (nilai negatif dalam desimal)
print(hasil_not)
```

Output :



```
1
7
6
20
1
-6

== Code Execution Successful ==
```

G. Operasi Logical (Logika)

Operator logika adalah elemen penting dalam pemrograman yang digunakan untuk membandingkan dua nilai boolean dan menghasilkan nilai boolean berdasarkan hasil perbandingan tersebut. Operator ini meliputi AND, OR, dan NOT. Operator AND menghasilkan TRUE hanya jika kedua operannya TRUE, sementara operator OR menghasilkan TRUE jika salah satu atau kedua operannya TRUE. Sementara itu, operator NOT digunakan untuk membalikkan nilai operand. Penggunaan operator logika ini memungkinkan pengembang untuk membuat struktur kontrol dan pengambilan keputusan yang efisien dalam alur program, meningkatkan kinerja, dan menjadikan kode lebih mudah dipahami.

Jenis Logika	Penjelasan	Contoh
and	Akan bernilai TRUE jika kedua pernyataan operandnya sama (True)	x and y
or	Akan bernilai TRUE setidaknya satu operandnya bernilai True.	x or y
not	Mengembalikan kebalikan dari nilai operand.	not x

Contoh penggunaan operasi logika :

```
# variabel
x = 5
y = 10

# Operator AND
result_and = (x > 0) and (y < 15)
print("Kedua kondisi terpenuhi (AND):", result_and)
result_and = (x > 0) and (y > 15)
print("Kedua kondisi terpenuhi (AND):", result_and)

# Operator OR
result_or = (x < 0) or (y < 15)
print("Salah satu kondisi terpenuhi (OR):", result_or)
result_or = (x < 0) or (y > 15)
print("Salah satu kondisi terpenuhi (OR):", result_or)

# Operator NOT
result_not = not(x == 5)
print("Nilai (NOT):", result_not)
result_not = not(x == 0)
print("Nilai (NOT):", result_not)
```

Output :

```
Kedua kondisi terpenuhi (AND): True
Kedua kondisi terpenuhi (AND): False
Salah satu kondisi terpenuhi (OR): True
Salah satu kondisi terpenuhi (OR): False
Nilai (NOT): False
Nilai (NOT): True

- Code Execution Successful ==
```

Penjelasan Alur :

- Jadi, penggunaan AND pada program diatas variabel “x” memiliki value = 5, sedangkan “y” memiliki value = 10.

Pada saat menggunakan operator AND kedua nilai tersebut masing masing memiliki fungsi untuk cek nilai perbandingan yaitu “>” dan “<” untuk kasus ini :

Operand 1 = Apakah X lebih dari 0 ? Ya (lebih dari 0 karena nilai yang ditetapkan adalah 5)

Operand 2 = Apakah Y kurang dari 15 ? Ya (kurang dari 15 karena nilai yang ditetapkan adalah 10)

Karena kedua nilai tersebut True / Ya maka tanggapan dari operator AND akan menghasilkan **True**.

- Untuk penggunaan OR nilai x dan y masih sama dengan sebelumnya,

Kenapa pada operasi pertama bernilai True? Karena salah satu dari kondisi tersebut ada yang memenuhi persyaratan dari operasi perbandingan.

Pada operand apa yang menghasilkan perbandingan tersebut dapat terpenuhi? Pada operand ke 2 yaitu:

Apakah Y lebih kecil dari 15? Ya, sedangkan operand 1 Apakah X lebih kecil dari 0 ? tidak

Walaupun seperti itu, pada hasil akhir penggunaan operasi OR akan menghasilkan **True**.

- Terakhir, penggunaan NOT kita memiliki nilai x yang sama dengan sebelumnya yaitu 5

Pada contoh tersebut kita membandingkan menggunakan operasi perbandingan (==) untuk kedua nilai atau operand.

Saat menggunakan operasi Not maka operasi tersebut hanya Mengembalikan atau membuat hasil terbalik dari yang diharapkan.

Seperti pada program, jika kita tidak menggunakan fungsi Not di dalamnya maka otomatis nilai tersebut akan bernilai **True**, kenapa?

Karena $x == 5$, yang dimana nilai dari variabel x adalah 5 jadinya $5 == 5$? Ya.

Jika diberikan fungsi Not, maka hasilnya akan terbalik dan akan menjadi **False**.

Seperti ini jika diberikan fungsi Not : **not(x == 5)** .

5. Struktur Kontrol: Percabangan



Pengecekan kondisi dan pernyataan if-else adalah bagian penting dari pemrograman karena mereka mengontrol aliran eksekusi. Pernyataan if-else dalam Python memungkinkan pengambilan keputusan dalam kode. Ini menentukan apakah akan mengeksekusi blok pernyataan berdasarkan hasil pemeriksaan kondisi. Pernyataan dalam blok if akan dijalankan jika kondisinya benar. Jika tidak, pernyataan dalam blok else akan dijalankan.

Di Python percabangan ini memiliki beberapa jenis yang perlu diketahui :

- If
- If – Else
- If – Else If(Elif) – Else
- Switch
- Try Except (error handling)

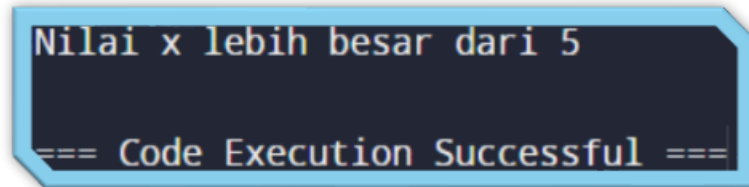
Dari 5 jenis tersebut kita akan membahasnya satu per satu agar lebih mudah dimengerti apa saja kegunaannya dari masing masing jenis:

A. If

Pada statement “if” hanya melakukan melakukan sebuah cek kondisi saja dan menjalankan sebuah pertanyaan selama bernilai True. Misal pada program dibawah :

```
x = 10
if x > 5:
    print("Nilai x lebih besar dari 5")
```

Output :



```
Nilai x lebih besar dari 5
=== Code Execution Successful ===
```

Penjelasan Alur :

Jadi, pada program tersebut kita punya variabel x dengan nilai 10, setelah itu kita punya sebuah statement percabangan “if” dengan operasi perbandingan “>”

Pada percabangan “if” tersebut kita membandingkan nilai x ini apakah lebih besar 5 ? Ya, karena 10 lebih besar dari angka 5.

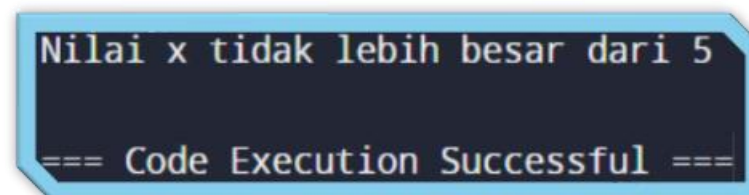
B. If – Else

Sekarang kita memiliki 2 percabangan yaitu if dan else, kita sebut saja blok percabangan. Jika pernyataan pertama bernilai True maka blok “If” akan dieksekusi atau dijalankan sedangkan jika pernyataan saat dicek kondisinya False, maka blok “else” akan berjalan atau dieksekusi.

Pada contoh dibawah adalah penggunaan if else:

```
x = 3
if x > 5:
    print("Nilai x lebih besar dari 5")
else:
    print("Nilai x tidak lebih besar dari 5")
```

Output :



```
Nilai x tidak lebih besar dari 5
=== Code Execution Successful ===
```

Penjelasan Alur :

Pada program diatas, kita punya variabel x dengan nilai/value 3, setelah itu ada percabangan dengan blok “if” melakukan perbandingan:

Apakah x(3) lebih besar dari angka 5 ? Tidak, karena 3 itu kurang dari 5.

Karena dari sini sudah jelas, pernyataan akhirnya adalah False, dan karena hasil akhir tersebut blok “else” akan berperan untuk memberikan output yang bernilai False.

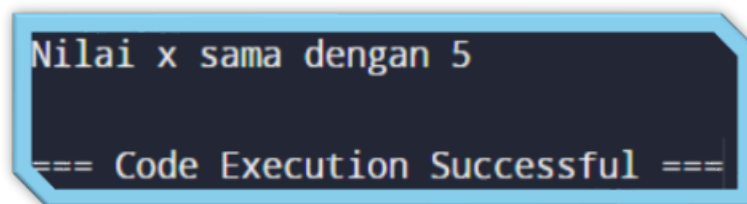
C. If – Elif – Else

Digunakan ketika ada lebih dari dua kondisi yang ingin dievaluasi. Pertama, kondisi dalam if akan dievaluasi. Jika kondisi tidak benar, maka blok kode dalam elif akan dievaluasi. Jika tidak ada kondisi yang benar, blok kode dalam else akan dievaluasi.

Berikut contoh penggunaan dari percabangan if elif else:

```
x = 5
if x > 5:
    print("Nilai x lebih besar dari 5")
elif x == 5:
    print("Nilai x sama dengan 5")
else:
    print("Nilai x kurang dari 5")
```

Output :



```
Nilai x sama dengan 5
=== Code Execution Successful ===
```

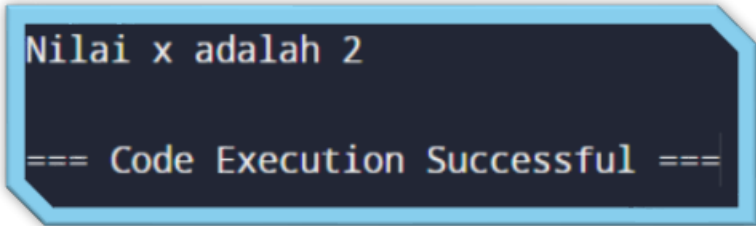
D. Switch

Memungkinkan kita untuk mengevaluasi nilai dari suatu ekspresi dan menjalankan blok kode yang sesuai dengan pola atau pattern yang cocok dengan nilai tersebut. Berikut contoh penggunaannya:

```
x = 2

match x:
    case 1:
        print("Nilai x adalah 1")
    case 2:
        print("Nilai x adalah 2")
    case _:
        print("Nilai x tidak sesuai dengan pola yang ada")
```

Output :



```
Nilai x adalah 2
```

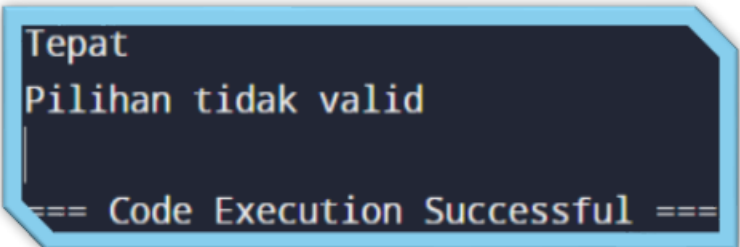
```
=== Code Execution Successful ===
```

➤ Catatan : Fungsi ini hanya ada di versi python 3.10.

Berikut adalah contoh yang bisa dipakai di versi python manapun:

```
def switch(angka):  
    case = {  
        1: 'Tepat',  
        2: 'Salah',  
        3: 'Mungkin'  
    }  
    return case.get(angka, "Pilihan tidak valid")  
  
print(switch(1))  
print(switch(4))
```

Output :



```
Tepat  
Pilihan tidak valid
```

```
=== Code Execution Successful ===
```

Alur Penjelasan :

Fungsi **switch_case** ini memanfaatkan kecerdasan Python dengan menggabungkan fungsi dan struktur data dictionary. Dengan menggunakan dictionary sebagai pengganti struktur switch case konvensional, program ini mengambil keuntungan dari kemampuan Python yang fleksibel untuk mencocokkan kunci dengan nilai yang sesuai. Sehingga, bahkan tanpa struktur switch case bawaan, kita dapat dengan mudah mengonversi angka menjadi kata-kata yang lebih deskriptif dengan bantuan fungsi ini.

E. Try Except

try dan **except** adalah duo dinamis dalam Python yang membantu dalam menangani kesalahan dan situasi tak terduga. Saat kita ingin melindungi bagian kode yang berpotensi menyebabkan masalah, kita melampirkannya ke blok **try**, yang bertindak sebagai perisai pertahanan pertama. Jika ada kesalahan di dalamnya, **except** hadir untuk menangkapnya dengan gesit, memberikan kita kesempatan untuk mengatasi masalah dengan penanganan kesalahan yang tepat. Dengan kemitraan yang solid, **try** dan **except** memberikan fleksibilitas dan keandalan dalam menangani kegagalan, menjadikan pengembangan Python lebih santai dan aman.

Ada beberapa jenis juga yang perlu diketahui dalam *try except* :

Jenis Exception	Penjelasan
AssertionError	Terjadi ketika pernyataan assert gagal.
AttributeError	Terjadi ketika atribut atau referensi atribut tidak ditemukan.
EOFError	Terjadi ketika fungsi input() mencapai akhir file tanpa adanya data.
FloatingPointError	Terjadi ketika operasi floating-point gagal.
GeneratorExit	Terjadi ketika generator atau iterator dihentikan sebelum selesai.
ImportError	Terjadi ketika modul yang diimpor tidak ditemukan atau gagal diimpor.
IndexError	Terjadi ketika pengindeksan berada di luar rentang (misalnya, daftar).
KeyError	Terjadi ketika kunci tidak ditemukan dalam kamus.
KeyboardInterrupt	Terjadi ketika pengguna menghentikan program dengan menekan Ctrl+C.
MemoryError	Terjadi ketika program kehabisan memori.
NameError	Terjadi ketika variabel lokal atau global tidak ditemukan.
NotImplementedError	Terjadi ketika metode atau fungsi abstrak belum diimplementasikan.
OSError	Terjadi ketika operasi sistem gagal (misalnya, file tidak ditemukan).
OverflowError	Terjadi ketika hasil operasi aritmatika terlalu besar untuk direpresentasikan.
RecursionError	Terjadi ketika kedalaman rekursi melebihi batas yang ditentukan.
StopIteration	Terjadi ketika metode next() dari iterator tidak memiliki item selanjutnya.
SyntaxError	Terjadi ketika terdapat kesalahan sintaks dalam kode Python.
IndentationError	Terjadi ketika ada masalah dengan indentasi dalam kode Python.
TabError	Terjadi ketika campuran tab dan spasi digunakan untuk indentasi.
SystemError	Terjadi ketika ada kesalahan internal dalam interpreter Python.

TypeError	Terjadi ketika operasi atau fungsi diberikan argumen dengan tipe yang tidak sesuai.
ValueError	Terjadi ketika fungsi menerima argumen dengan nilai yang tidak valid.
ZeroDivisionError	Terjadi ketika pembagian oleh nol dilakukan.

Contoh penggunaan try except seperti berikut:

```
try:
    angka = int(input("Masukkan angka: "))
    hasil = 10 / angka
    print("Hasil pembagian 10 oleh", angka, "adalah", hasil)
except ZeroDivisionError:
    print("Tidak bisa melakukan pembagian dengan angka nol.")
except ValueError:
    print("Masukan tidak valid. Harap masukkan angka.")
except Exception as e:
    print("Terjadi kesalahan:", e)
```

Contoh Beberapa Output :

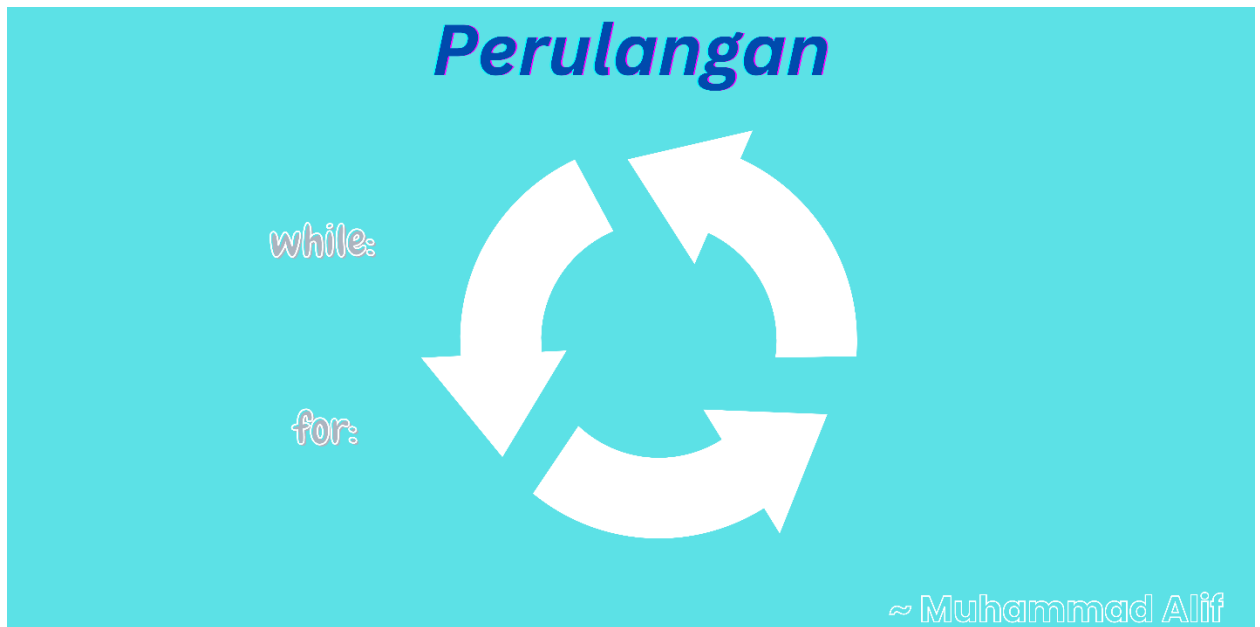
```
Masukkan angka: 0
Tidak bisa melakukan pembagian dengan angka nol.

=== Code Execution Successful ===
```

```
Masukkan angka: A#335tgse
Masukan tidak valid. Harap masukkan angka

=== Code Execution Successful ===
```

6. Struktur Kontrol: Perulangan



Perulangan for dikenal sebagai counted loop, sementara perulangan while disebut uncounted loop. Perbedaannya terletak pada penggunaannya: for digunakan untuk iterasi sejumlah perulangan yang sudah diketahui sebelumnya, sementara while cocok digunakan untuk kondisi perulangan yang mungkin tidak terbatas dan bergantung pada suatu syarat tertentu.

Perulangan atau loop merupakan teknik untuk mengulang-ulang eksekusi suatu blok kode, atau mengiterasi elemen milik tipe data kolektif (contohnya: list). Berikut beberapa contoh penggunaan for dan while:

A. For

Contoh Menggunakan Range :

```
# Contoh penggunaan perulangan for untuk mencetak angka dari 1 hingga 5
for i in range(1, 6):
    print(i)
```

Output :

```
1
2
3
4
5
== Code Execution Successful ==
```

Penjelasan Alur :

Pada program diatas, perulangan for menggunakan fungsi range rumusnya itu n-1 jadi pada value kedua pada fungsi tersebut dikurangi 1 misalnya *range(1,11)* artinya rangenya berisi 1 sampai 10.

Contoh Output Masing – Masing Karakter :

```
message = "Halo Dunia!"  
for char in message:  
    print(char)
```

Output :



B. While

Contoh Menghitung Sampai Batas Yang Ditentukan :

```
count = 0  
while count < 5:  
    print("Perulangan ke-", count+1)  
    count += 1
```

Output :

```
Perulangan ke- 1  
Perulangan ke- 2  
Perulangan ke- 3  
Perulangan ke- 4  
Perulangan ke- 5
```

Contoh Inputan Berulang sampai Kondisi Memenuhi :

```
answer = ""  
while answer.lower() != "ya":  
    answer = input("Apakah Anda ingin keluar? (ya/tidak): ")  
print("Terima kasih!")
```

Output :

```
Apakah Anda ingin keluar? (ya/tidak): tidak  
Apakah Anda ingin keluar? (ya/tidak): tidakt  
Apakah Anda ingin keluar? (ya/tidak): tidak  
Apakah Anda ingin keluar? (ya/tidak): ya  
Terima kasih!
```

```
== Code Execution Successful ==
```

7. String

String

“ Aku adalah String. ”

~ Muhammad Alif

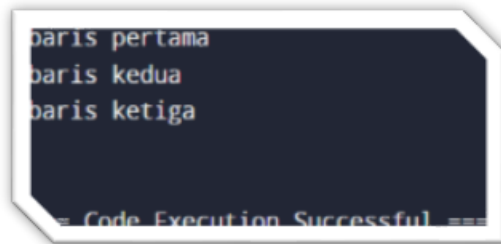
String dalam Python adalah tipe data yang digunakan untuk merepresentasikan teks atau urutan karakter. Dalam Python, string didefinisikan dengan tanda kutip tunggal (' '), tanda kutip ganda (" "), atau tanda kutip tiga (''' ' atau '''' '''). String dapat berisi huruf, angka, simbol, atau kombinasi dari semuanya.

A. Multiline String

Multiline yang dimaksud adalah bisa membuat sebuah string dalam banyak baris, pada string biasa itu hanya bisa membuat dalam 1 baris saja, sedangkan multiline bisa beberapa baris. Seperti contoh berikut :

```
text = """baris pertama
baris kedua
baris ketiga
"""
print(text)
```

Output :



```
baris pertama
baris kedua
baris ketiga

== Code Execution Successful ==
```

- Catatan : Kamu bisa bebas memberi tab atau indentasi sebanyak yang kamu mau pada multiline tersebut.

B. String Special Character

Pada pembahasan ini, hanya fitur bawaan python yang cukup berguna dan ini melibatkan penggunaan ascii, digunakan juga untuk mengatur posisi string/teks, menambahkan karakter khusus dalam string, dan lain lain. Maka dari itu ini membuat kita fleksibel untuk memanipulasi sebuah string.

Berikut tabel mengenai apa saja hal yang terdapat pada String Special Character:

Jenis Special Character	Penjelasan
\	Karakter backslash digunakan untuk menyisipkan karakter khusus atau mengabaikan arti khusus dari karakter di sebelumnya. Misalnya, \\ digunakan untuk menyisipkan karakter backslash itu sendiri.
\b	Digunakan untuk backspace
\n	digunakan untuk membuat baris baru dalam teks. Ketika \n digunakan, kursor maju ke baris berikutnya, sehingga teks selanjutnya akan ditulis pada baris baru.
\r	digunakan untuk membuat baris baru, tetapi dengan perbedaan bahwa kursor kembali ke awal baris yang sama setelah menggunakan karakter ini. Dengan kata lain, saat \r digunakan, kursor kembali ke posisi awal baris saat ini, dan teks selanjutnya yang ditulis akan menempa teks yang sudah ada di baris yang sama.
\t	Menghasilkan karakter tab horizontal, yang digunakan untuk membuat tabulasi horizontal dalam string.
\{oktal} atau \{hex}	Pada penggunaan backslash tersebut bisa membuat karakter dari oktal atau hexadecimal.

Contoh penggunaan String Special Character :

```
# Contoh penggunaan karakter backslash (\)
print("Ini adalah karakter backslash: \\")

# Contoh penggunaan karakter backspace (\b)
print("Ini adalah karakter sebelum backspace.\bIni adalah setelah backspace.")

# Contoh penggunaan karakter baris baru (\n)
print("Baris pertama.\nBaris kedua.")

# Contoh penggunaan karakter carriage return (\r)
print("Ini adalah teks sebelum carriage return.\rIni adalah teks setelah carriage return.")

# Contoh penggunaan karakter tab horizontal (\t)
print("Ini\tadalah\tteks\tdengan\ttab")

# Contoh penggunaan karakter oktal (\{oktal}) dan heksadesimal (\x{hex})
print("Karakter oktal: \122")
print("Karakter heksadesimal: \x41")
```

Output :

```
Ini adalah karakter backslash: \
Ini adalah karakter sebelum backspace. Ini adalah setelah backspace.
Baris pertama.
Baris kedua.
Ini adalah teks sebelum carriage return.
Ini adalah teks setelah carriage return.
Ini adalah teks    dengan tab
Karakter oktal: R
Karakter heksadesimal: A
--- Code Execution Successful ---
```

C. String Formatting

a. Formatting

Penjelasan :

%s = Berfungsi sebagai placeholder untuk nilai string nantinya dari variabel.
% = Operator untuk memasukkan nilai variabelnya

(variabel) = variabel yang akan dimasukkan ke placeholder %s

```
nama = "Muhammad Alif"
umur = 20
print("Nama: %s, Umur: %s tahun" % (nama, umur))
```

- Catatan : Saat memberikan placeholder %s, dan variabel pada string ("") itu seperti berurutan, jadi urutannya kiri ke kanan. Dan akan tampil *sesuai dengan posisi variabel yang di masukkan*.

b. String Interpolation

Sedikit mirip dengan formatting sebelumnya, tetapi ini ada tambahan metode pada stringnya yaitu `.format(variabel)` dan placeholdernya atau hasil dari variabel tersebut akan ditampilkan pada posisi {}

```
nama = "Muhammad Alif"
umur = 20
print("Nama: {}, Umur: {} tahun".format(nama, umur))
```

- Catatan : Saat memberikan placeholder {}, dan variabel pada string ("") itu seperti berurutan, jadi urutannya kiri ke kanan. Dan akan tampil *sesuai dengan posisi variabel yang di masukkan*.

c. Literal String Interpolation

Sedikit berbeda dengan formatting lainnya, ini bisa langsung memanggilnya tanpa menggunakan metode seperti sebelumnya. Cukup memasukkan nama variabel pada { variabel } value dari variabel tersebut akan ditampilkan.

Tetapi pada awal fungsi print() diberikan huruf (f) untuk memanggil variabel didalam string langsung dengan menempatkan di {}.

```
nama = "Muhammad Alif"
umur = 20
print(f>Nama: {nama}, Umur: {umur} tahun")
```

D. Penggabungan String

Penggabungan yang dimaksud adalah seperti kita punya 2 atau lebih string, setelah itu di gabung menjadi 1 string.

Berikut beberapa contoh yang bisa dicoba:

1. Penggabungan Menggunakan Operator (+)

```
string1 = "Hello"
string2 = "World"
gabung = string1 + " " + string2
```



```
print(gabung)
```

Output :

```
Hello World
```

```
== Code Execution Successful ==
```

2. Menggunakan Concatenation

Jadi pada metode ini kita menggabungkan 2 string yang terpisah tanpa adanya separator ataupun operator.

```
string = "Hello " "World"  
print(string)
```

Output :

```
Hello World
```

```
== Code Execution Successful ==
```

E. Method String

Method string pada Python adalah fungsi-fungsi yang terkait dengan objek string yang memungkinkan manipulasi dan pengolahan string. Method-string memungkinkan Anda untuk melakukan berbagai operasi pada string, seperti pencarian, pemotongan, pemisahan, pemformatan, dan banyak lagi. Ini dapat memungkinkan kita untuk memanipulasi teks dengan cara yang efisien dan mudah dibaca. Beberapa method string yang umum digunakan di Python antara lain:

Jenis Method String	Penjelasan
capitalize()	Mengubah karakter pertama string menjadi huruf besar.
lower()	Mengubah semua karakter dalam string menjadi huruf kecil.
upper()	Mengubah semua karakter dalam string menjadi huruf besar.
swapcase()	Mengubah huruf besar menjadi huruf kecil dan sebaliknya.

title()	Mengubah setiap kata dalam string menjadi huruf besar.
count(substring)	Menghitung berapa kali sebuah substring muncul dalam string.
find(substring)	Mengembalikan indeks dari pertama kali kemunculan sebuah substring dalam string.
replace(old, new)	Mengganti setiap kemunculan substring tertentu dengan substring baru.
split(separator)	Memecah string menjadi sebuah list berdasarkan separator yang diberikan.
join(iterable)	Menggabungkan elemen-elemen dalam sebuah iterable menjadi string.
strip()	Menghapus whitespace atau karakter tertentu dari awal dan akhir string.
startswith(prefix)	Memeriksa apakah string dimulai dengan prefix tertentu.
endswith(suffix)	Memeriksa apakah string diakhiri dengan suffix tertentu.

Sekarang kita membuat contoh kegunaan dari setiap method yang ada diatas:

```

string = "hello world"
print(string.capitalize())

string = "HELLO WORLD"
print(string.lower())

string = "hello world"
print(string.upper())

string = "Hello World"
print(string.swapcase())

string = "hello world"
print(string.title())

string = "hello world"
print(string.count('l'))

string = "hello world"
print(string.find('world'))

string = "hello world"
print(string.replace('world', 'universe'))

string = "hello world"
print(string.split())

my_list = ['hello', 'world']
print(' '.join(my_list))

string = "  hello world  "
print(string.strip())

```

```
string = "hello world"
print(string.startswith('hello'))

string = "hello world"
print(string.endswith('world'))
```

Output :

```
Hello world
hello world
HELLO WORLD
hELLO WORLD
Hello World
3|
6
hello universe
['hello', 'world']
hello world
hello world
True
True

=== Code Execution Successful ===
```

8. RegEx



Regular expression, atau sering disingkat sebagai regex, adalah sebuah urutan karakter yang membentuk sebuah pola pencarian. Pola ini kemudian dapat digunakan untuk pencarian teks dan manipulasi teks berdasarkan aturan yang telah ditentukan.

Jenis Fungsi RegEx	Penjelasan
re.search	Mencari pola dalam teks dan mengembalikan objek dengan detail pencocokan. Berguna untuk menemukan kata pertama yang cocok dengan pola dalam string.
re.findall	Mengembalikan daftar semua kemunculan pola dalam teks, bukan hanya yang pertama. Berguna untuk menemukan semua kemunculan suatu pola dalam dokumen.
re.match	Mencocokkan pola di awal string. Jika pola tidak ditemukan di awal string, mengembalikan None. Cocok untuk pencarian di awal string.
re.sub	Mengganti teks dengan pola tertentu dalam string. Berguna untuk mengganti semua kemunculan pola dengan teks lain, seperti mengubah format tanggal.
re.split	Memisahkan string menjadi list berdasarkan pola yang diberikan. Berguna dalam membersihkan dan mempersiapkan data untuk analisis lebih lanjut.
re.compile	Mengkompilasi pola RegEx menjadi objek RegEx. Memungkinkan penggunaan pola

	yang sama dalam banyak operasi pencarian, meningkatkan efisiensi kode.
--	--

Berikut beberapa contoh program mengenai penggunaan regex:

1. Mencari awalan kata dalam sebuah kalimat berdasarkan 1 karakter

```
import re

text = "Hari ini cerah, waktu yang cocok untuk pergi kekampus Universitas Mulawarman."
pattern = r"\bc\w+"
words_with_p = re.findall(pattern, text)
print(f"Kata-kata dengan awalan 'c':", words_with_p)
```

Output :

```
Kata-kata dengan awalan 'c': ['cerah', 'cocok']

=== Code Execution Successful ===
```

2. Mencari sebuah kata berdasarkan kata kunci yang diberikan dari 1 kalimat

```
import re

teks = "Perkenalkan nama aku adalah muhammad alif."
pola = r"nama"

hasil = re.search(pola, teks)
if hasil:
    print("String ditemukan:", hasil.group())
else:
    print("String tidak ditemukan.")
```

Output :

```
String ditemukan: nama

=== Code Execution Successful ===
```

9. List

List

[value, value2]

~ Muhammad Alif

List merupakan salah satu struktur data yang digunakan untuk menyimpan kumpulan elemen atau nilai-nilai. Dalam Python, list ditandai dengan tanda kurung siku ([]) dan elemen-elemennya dipisahkan oleh koma. List dapat berisi nilai-nilai dengan jenis data yang berbeda-beda, dan dapat diubah (mutable), artinya kita dapat menambah, menghapus, atau mengubah elemen-elemennya setelah list dibuat.

Contoh List :

```
contoh_list = [1, True, 3.2, 'halo', "Alif", 6.0]
```

Untuk mengakses sebuah list, itu kita sebut **Index**, index sendiri itu selalu dimulai dari 0 untuk list dan beberapa tipe data kolektif lainnya yang akan dibahas selanjutnya.

Pada contoh list diatas, ini adalah beberapa penjelasan bagaimana mengakses sebuah elemen list:

1. Elemen pertama: Indeks 0 dari list contoh_list adalah 1. Ini adalah nilai integer.
2. Elemen kedua: Indeks 1 dari list contoh_list adalah True. Ini adalah nilai boolean.
3. Elemen ketiga: Indeks 2 dari list contoh_list adalah 3.2. Ini adalah nilai float.
4. Elemen keempat: Indeks 3 dari list contoh_list adalah 'halo'. Ini adalah nilai string.
5. Elemen kelima: Indeks 4 dari list contoh_list adalah "Alif". Ini juga adalah nilai string.
6. Elemen keenam: Indeks 5 dari list contoh_list adalah 6.0. Ini adalah nilai float.

Ada beberapa hal juga bisa dibahas tentang list seperti bagaimana penggunaannya dan beberapa hal tentang manipulasi data dari list bagaimana :

A. Melihat Isi Menggunakan Perulangan

Untuk melihatnya kita bisa menggunakan for atau while, berikut contohnya:

1. For

```
daftar_angka = [1, 2, 3, 4, 5, 7]

for angka in daftar_angka:
    print(angka)
```

Output :

```
1
2
3
4
5
7

== Code Execution Successful ==
```

2. While

```
daftar_angka = [1, 2, 3, 4, 5]
indeks = 0
while indeks < len(daftar_angka):
    print(daftar_angka[indeks])
    indeks += 1
```

Output :

```
1
2
3
4
5

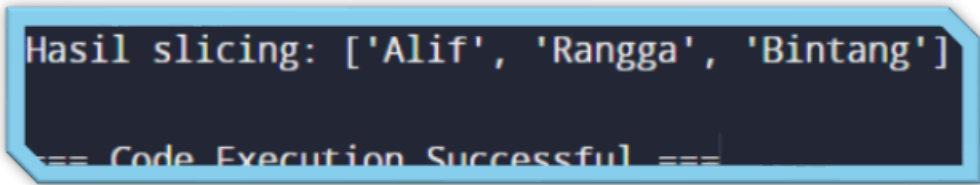
== Code Execution Successful ==
```

B. Slicing pada List

Penggunaan slicing, seperti pada umumnya saja seperti berikut :

```
nama = ["Ardi", "Alif", "Rangga", "Bintang", "Steven"]  
print("Hasil slicing:", nama[1:4])
```

Output :



```
Hasil slicing: ['Alif', 'Rangga', 'Bintang']  
=== Code Execution Successful ===
```

Penjelasan Singkat :

nama[1:4]: Ini adalah sintaks slicing yang digunakan untuk mengambil potongan dari list nama. Di sini, kita mengambil elemen dari indeks ke-1 hingga ke-3. Perlu diingat bahwa dalam slicing, indeks awal (1) diikutsertakan, sedangkan indeks akhir (4) tidak diikutsertakan. Jadi, ini akan mengambil elemen pada indeks 1, 2, dan 3 dari list nama.

C. Penggunaan Syntax List

Penggunaan list selanjutnya adalah syntax method atau beberapa metode yang membuat kita melakukan manipulasi data seperti menambah elemen, menghapus, sorting/menugurkan data, dan masih banyak lagi.

Jenis Method List	Penjelasan
append()	Menambahkan elemen ke akhir list.
remove()	Menghapus elemen tertentu dari list.
sort()	Mengurutkan elemen-elemen dalam list.
index()	Mengembalikan indeks elemen tertentu.
len()	Mengembalikan panjang (jumlah elemen) list.
count()	Mengembalikan jumlah kemunculan nilai tertentu dalam list.
reverse()	Membalik urutan elemen dalam list.
extend()	Menambahkan elemen dari iterable ke akhir list.
pop()	Menghapus dan mengembalikan elemen terakhir list.

Dari jenis jenis diatas, berikut beberapa contoh penggunaannya:

```
list_127 = [1, 3, 2, 4, 3, 5]

# Menambahkan elemen baru ke akhir list
list_127.append(6)
print("Setelah menambahkan elemen baru:", list_127)

# Menghapus elemen tertentu dari list
# Catatan : hanya menghapus urutannya dari kiri ke kanan.
list_127.remove(3)
print("Setelah menghapus elemen 3:", list_127)

# Mengurutkan elemen dalam list
list_127.sort()
print("Setelah diurutkan:", list_127)

# Mengembalikan indeks elemen tertentu
indeks = list_127.index(4)
print("Indeks dari elemen 4:", indeks)

# Mengembalikan panjang list
panjang = len(list_127)
print("Panjang list:", panjang)

# Menghitung jumlah kemunculan nilai tertentu dalam list
jumlah = list_127.count(3)
print("Jumlah kemunculan nilai 3:", jumlah)

# Membalik urutan elemen dalam list
list_127.reverse()
print("Setelah dibalik:", list_127)

# Menambahkan elemen dari iterable ke akhir list
list_127.extend([7, 8, 9])
print("Setelah menambahkan elemen dari iterable:", list_127)

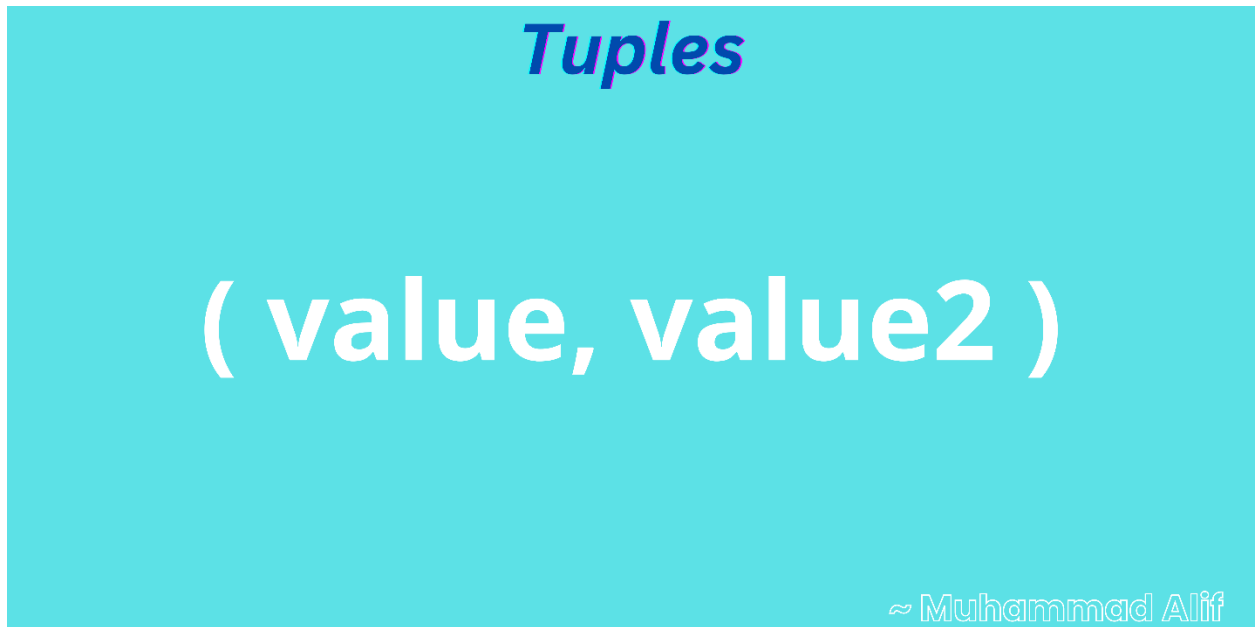
# Menghapus dan mengembalikan elemen terakhir list
terakhir = list_127.pop()
print("Elemen terakhir yang dihapus:", terakhir)
print("List setelah pop:", list_127)
```

Output :

```
Setelah menambahkan elemen baru: [1, 3, 2, 4, 3, 5, 6]
Setelah menghapus elemen 3: [1, 2, 4, 3, 5, 6]
Setelah diurutkan: [1, 2, 3, 4, 5, 6]
Indeks dari elemen 4: 3
Panjang list: 6
Jumlah kemunculan nilai 3: 1
Setelah dibalik: [6, 5, 4, 3, 2, 1]
Setelah menambahkan elemen dari iterable: [6, 5, 4, 3, 2, 1, 7, 8, 9]
Elemen terakhir yang dihapus: 9
List setelah pop: [6, 5, 4, 3, 2, 1, 7, 8]
```

Code Execution Successful

10.Tuples



Tuple memang mirip seperti list tapi yang membedakannya menggunakan “()” sebagai identitasnya dan juga ini tidak dapat diubah berbeda dengan list yang bisa di manipulasi.

Fungsi Build-in pada Tuple :

Jenis Build-in Tuple	Penjelasan
len(tuple)	Memberikan total panjang tuple.
max(tuple)	Mengembalikan item dari tuple dengan nilai maks.
min(tuple)	Mengembalikan item dari tuple dengan nilai min.
tuple(seq)	Mengubah seq menjadi tuple

Apakah ada cara lain agar bisa melakukan perubahan data pada tuple? Bisa.

```
# Tuple awal
tuple_awal = (1, 2, 3, 4, 5)

# Menambahkan elemen baru ke tuple
tuple_baru_1 = tuple_awal + (6,)
print("Tuple setelah menambahkan elemen baru:", tuple_baru_1)

# Menghapus elemen tertentu dari tuple
indeks_hapus = 2
tuple_baru_2 = tuple([elemen for indeks, elemen in
enumerate(tuple_awal) if indeks != indeks_hapus])
```

```
print("Tuple setelah menghapus elemen dengan indeks", indeks_hapus,
":", tuple_baru_2)

# Menggabungkan dua tuple
tuple_baru_3 = (7, 8, 9)
tuple_gabungan = tuple_awal + tuple_baru_3
print("Tuple hasil penggabungan:", tuple_gabungan)
```

Output :

```
Tuple setelah menambahkan elemen baru: (1, 2, 3, 4, 5, 6)
Tuple setelah menghapus elemen dengan indeks 2 : (1, 2, 4, 5)
Tuple hasil penggabungan: (1, 2, 3, 4, 5, 7, 8, 9)

=== Code Execution Successful ===
```

11.Sets

Sets

{value, value2}

~ Muhammad Alif

Sets, pada tipe data kolektif ini mirip seperti list, hanya saja tidak dapat duplikat elemen didalamnya jadi hanya bisa unik. Pembedanya dengan tipe data kolektif lain adalah menggunakan kurung kurawal {}. Ini juga bersifat unordered atau tidak memiliki index.

Cara Membuat Sets

```
sets = {1, 2, 3}
```

Kegunaan Sets, pada umumnya bisa langsung menghapus elemen yang duplikasi diset. Selain itu ada beberapa cara membuat set dan juga cara mengaksesnya.

```
buat_set = {1,2,3}

# Membuat list berada didalam set
buat_set = set([1,2,3])
```

Jadi, fungsi set() ini cara lain untuk membuat tipe data kolektif sets, dan uniknya pada fungsi tersebut kita hanya bisa membuat tipe data list didalamnya. Untuk cara pengaksesan dari tipe data sets ini hanya bisa menggunakan perulangan for terutama

Berikut beberapa cara penggunaan sets :

```
# Membuat set kosong
empty_set = set()
print("Set kosong:", empty_set)

# Membuat set dengan elemen
my_set = {1, 2, 3, 4, 5, 5}
print("Set dengan elemen:", my_set)

# Menambahkan elemen ke dalam set
my_set.add(6)
print("Set setelah ditambahkan elemen:", my_set)

# Menghapus elemen dari set
my_set.remove(3)
print("Set setelah menghapus elemen:", my_set)

# Menampilkan panjang set
print("Panjang set:", len(my_set))

# Iterasi melalui set
print("Elemen dalam set:")
for item in my_set:
    print(item)

# Menggunakan operator in untuk memeriksa keanggotaan
print("Apakah 2 ada dalam set?", 2 in my_set)
print("Apakah 7 ada dalam set?", 7 in my_set)
```

Built-in yang dimiliki oleh sets pada python sebagai berikut :

Jenis Method Sets	Penjelasan
add()	Menambahkan elemen baru ke dalam set.
remove()	Menghapus elemen tertentu dari set. Jika elemen tidak ada, akan menimbulkan KeyError.
discard()	Menghapus elemen tertentu dari set. Jika elemen tidak ada, tidak akan menimbulkan error.
clear()	Menghapus semua elemen dari set.

copy()	Mengembalikan salinan dari set.
pop()	Menghapus dan mengembalikan elemen acak dari set. Jika set kosong, akan menimbulkan KeyError.
update()	Menambahkan elemen-elemen dari set lain atau iterable ke dalam set.
union() atau	Mengembalikan gabungan (union) dari dua set.
intersection() atau &	Mengembalikan irisan (intersection) dari dua set.
difference() atau -	Mengembalikan selisih (difference) antara dua set.
symmetric_difference() atau ^	Mengembalikan symmetric difference antara dua set.
issubset()	Memeriksa apakah set tersebut adalah subset dari set lain.
issuperset()	Memeriksa apakah set tersebut adalah superset dari set lain.

Contoh penggunaan pada daftar tabel diatas :

```
# Method add()
my_set = {1, 2, 3}
my_set.add(4)
print("Set setelah menambahkan elemen 4:", my_set)

# Method remove()
my_set.remove(2)
print("Set setelah menghapus elemen 2:", my_set)

# Method clear()
my_set.clear()
print("Set setelah di-clear:", my_set)

# Method copy()
my_set = {1, 2, 3}
new_set = my_set.copy()
print("Salinan set baru:", new_set)

# Method pop()
popped_element = my_set.pop()
print("Elemen yang di-pop:", popped_element)

# Method update()
my_set.update({4, 5})
print("Set setelah di-update:", my_set)

# Method union()
set1 = {1, 2, 3}
set2 = {3, 4, 5}
union_set = set1.union(set2)
print("Union dari set1 dan set2:", union_set)

# Method intersection()
```

```

intersection_set = set1.intersection(set2)
print("Intersection dari set1 dan set2:", intersection_set)

# Method difference()
difference_set = set1.difference(set2)
print("Difference dari set1 dan set2:", difference_set)

# Method symmetric_difference()
symmetric_difference_set = set1.symmetric_difference(set2)
print("Symmetric difference dari set1 dan set2:",
symmetric_difference_set)

# Method issubset()
print("Apakah set1 subset dari set2?", set1.issubset(set2))

# Method issuperset()
print("Apakah set1 superset dari set2?", set1.issuperset(set2))

```

Output :

```

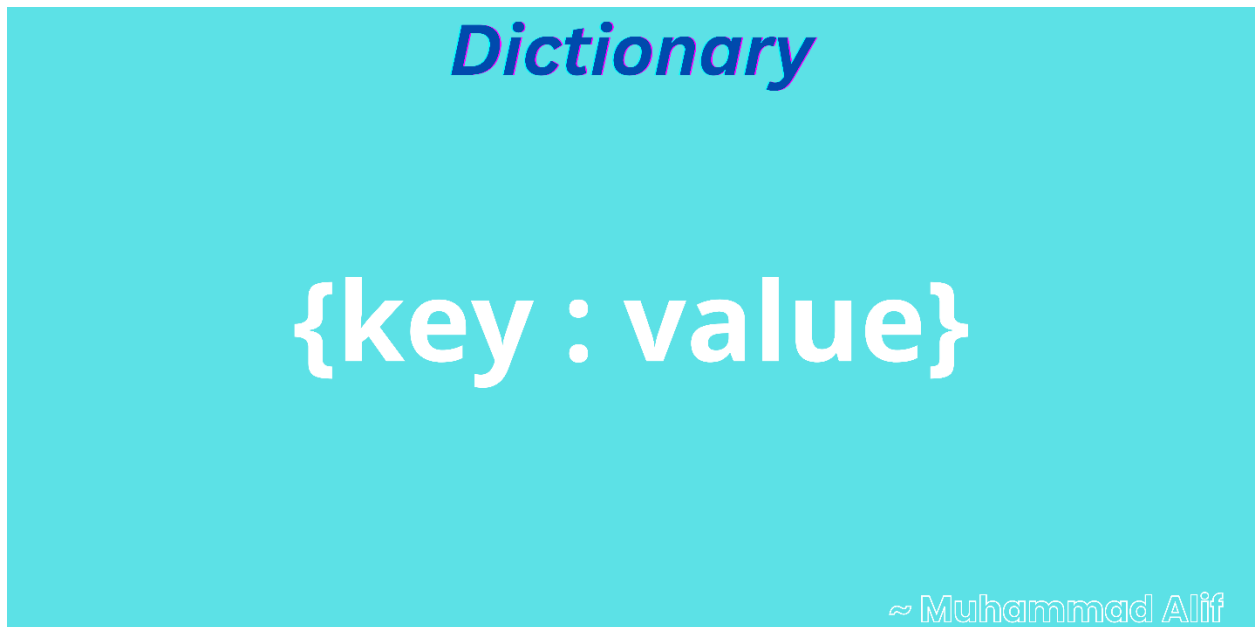
Set setelah menambahkan elemen 4: {1, 2, 3, 4}
Set setelah menghapus elemen 2: {1, 3, 4}
Set setelah di-clear: set()
Salinan set baru: {1, 2, 3}
Elemen yang di-pop: 1
Set setelah di-update: {2, 3, 4, 5}
Union dari set1 dan set2: {1, 2, 3, 4, 5}
Intersection dari set1 dan set2: {3}
Difference dari set1 dan set2: {1, 2}
Symmetric difference dari set1 dan set2: {1, 2, 4, 5}
Apakah set1 subset dari set2? False
Apakah set1 superset dari set2? False

Code Execution Successful ==

```

Selain sets, ada juga fungsi **frozenset()** kurang lebih dengan set biasa tetapi ini immutable tidak bisa diubah elemennya.

12.Dictionary



Dictionary, tipe data kolektif yang cukup unik dari yang lainnya karena hanya memiliki 2 nilai yaitu *key* & *value*. Hal menarik lainnya adalah tipe data ini memiliki sifat *unordered*, *changeable/mutable*, dan *unique*. Maksudnya adalah tidak memiliki indeks, bisa di ubah, dan nilai pada keynya tidak bisa 2 atau lebih.

Mari bahas terlebih dahulu contoh unique :

Unique, artinya unik dan pastinya berbeda dengan yang lain. Di Tipe Data ini memiliki sifat yang dimana key tidak bisa lebih dari 1, misal:

```
mahasiswa = {  
    "nama": "Muhammad Alif",  
    "nama": "Muhammad Alif"  
}
```

Selain itu, berikut cara membuat sebuah dictionary ada 2 cara yaitu menggunakan fungsi bawaan atau method bawaan:

1. Struktur Dictionary

```
nama_dict = {  
    "key": "value"  
}
```

2. Method dict()

```
# Fungsi dict()
dictionary_method = dict(nama="Muhammad Alif", umur=20)
# Bentuk Dictionary :
# dictionary_method = {
#     "nama" : "Muhammad Alif",
#     "umur" : 20
# }
```

3. Menggunakan Tipe Data Kolektif Campuran

Pada konteks campuran kali ini kita membuat dari list ke dictionary :

```
bentuk_list = ["test", "lala"]
keys = range(1, len(bentuk_list) + 1)
berubah_dict = dict(zip(keys, bentuk_list))
print(berubah_dict)
```

Output :



```
{1: 'test', 2: 'lala'}
```

=== Code Execution Successful ===

4. Dictionary bisa berisi berbagai struktur data tipe data kolektif

```
_dict = {
    "list": [1, 2, 3, 4, 5],
    "tuple": (6, 7, 8, 9, 10),
    "set": {11, 12, 13, 14, 15}
}
```

Contoh Penggunaan :

a. Memanggil Indeks List

```
_dict = {
    "list": [1, 2, 3, 4, 5],
    "tuple": (6, 7, 8, 9, 10),
    "set": {11, 12, 13, 14, 15}
}

print("Nilai list:", _dict["list"])
```



```
# Memanggil Indeks pada Dictionary List
print("Nilai pertama list:", _dict["list"][0])
```

b. Memanggil Indeks Tuple

```
_dict = {
    "list": [1, 2, 3, 4, 5],
    "tuple": (6, 7, 8, 9, 10),
    "set": {11, 12, 13, 14, 15}
}

print("Nilai tuple:", _dict["tuple"])
# Memanggil Indeks pada Dictionary Tuple
print("Nilai pertama dari tuple:", _dict["tuple"][0])
```

c. Memanggil Indeks Set

```
_dict = {
    "list": [1, 2, 3, 4, 5],
    "tuple": (6, 7, 8, 9, 10),
    "set": {11, 12, 13, 14, 15}
}

print("Nilai set:", _dict["set"])
# Memanggil Indeks pada Dictionary Sets
# Karena Unordered dan tidak bisa dipanggil salah satu
elemennya, salah satunya caranya adalah melakukan konversi
pada tipe data tersebut menjadi list dengan cara :

jadi_list = list(_dict["set"])
print("Nilai pertama dari set pertama : ", jadi_list[0])
```

13.Function

Function

def()

~ Muhammad Alif

Function salah satu fungsi yang sangat membantu dan terorganisir, karena function pada python kita bisa memberikan script atau beberapa baris kode yang hanya bisa dijalankan jika kita memanggilnya. Dan dalam fungsi tersebut bisa diberikan input maupun output, karakteristiknya biasa menggunakan **def()**. Kode yang dianggap termasuk dalam function itu harus berada diIndentasinya.

Bentuk dari function sebagai berikut :

```
def nama_fungsi(parameter1, parameter2, ...):  
    # Badan fungsi dimulai di sini  
    # Lakukan operasi atau tugas yang diinginkan  
    # Gunakan parameter yang diberikan jika diperlukan  
    # Akhir dari badan fungsi  
  
# Memanggil fungsi  
nama_fungsi(nilai_parameter1, nilai_parameter2, ...)
```

Hal yang perlu diperhatikan dalam function :

- **Positional Argument**, pada parameter jika membuat parameter urutannya harus sama jika dipanggil fungsinya. Misal:

```
def fungsi(a,b):  
    print(f'Aku adalah a : {a}')
```

```
    print(f'Aku adalah b : {b}')
```

```
fungsi("Hai","Test")
```

```
    #a      b
```

Nah, urutannya juga harus sama dengan parameter pada function yang dibuat. Sehingga ini adalah outputnya :

```
Aku adalah a : Hai
Aku adalah b : Test

=== Code Execution Successful ===
```

- **Keyword Argument**, fungsi ini kamu bisa menyebutkan nama parameter dan memberikan nilainya sehingga memungkinkan kamu tidak perlu sesuai urutan memanggil nama parameternya seperti contoh Positional Argument.

Sebagai contoh berikut:

```
def salam(nama, pesan):
    #func #P1      #P2
    print(f"Hai teman, {nama}! {pesan}")
        #nama      #pesan
salam(nama="Alif", pesan="hari ini aku bermain Mobile
Legends")
salam(pesan="kuliah pagi menyenangkan!", nama="Steven")
```

Output :

```
Hai teman, Alif! hari ini aku bermain Mobile Legends
Hai teman, Steven! kuliah pagi menyenangkan!

=== Code Execution Successful ===
```

- **Optional Argument**, fungsi ini kamu bisa memberikan nilai default atau nilai awal jika saat dipanggil kamu tidak perlu mengisi parameter lainnya(tidak wajib). Tapi kamu juga bisa memberikan nilai pada parameter lainnya jika ingin mengubah nilai awal/defaultnya.

Seperti contoh berikut :

```
def salam(nama, pesan="hari yang indah bermain mobile
legend!"):
    print(f"Hello, {nama}! {pesan}")

salam("Alif") # Hanya menggunakan parameter nama, sehingga
output "pesan" akan sesuai dengan yang ada di function yaitu
"good morning!"
```

```
salam("Steven", "Main yuk?") # Mengganti atau memberikan
nilai pada parameter pesan.
```

Output :

```
Hello, Alif! hari yang indah bermain mobile legend!
Hello, Steven! Main yuk?

=== Code Execution Successful ===
```

Contoh Program Function Hampir Mencakup Modul :

```
import re

# Dictionary untuk menyimpan data siswa
siswa = []

# Fungsi untuk menambahkan data siswa baru
def tambah_data(nama, umur, kelas):
    try:
        umur = int(umur)
        student = {"nama": nama, "umur": umur, "kelas": kelas}
        siswa.append(student)
        print(f>Data siswa {nama} berhasil ditambahkan.")
    except ValueError:
        print("Usia harus berupa bilangan bulat.")

# Fungsi untuk menampilkan data seluruh siswa
def tampilkan_data():
    if not siswa:
        print("Belum ada data siswa.")
    else:
        print("Data siswa:")
        for idx, student in enumerate(siswa, start=1):
            print(f"{idx}. Nama: {student['nama']}, Usia: {student['umur']},
Kelas: {student['kelas']}")

# Fungsi untuk mencari mahasiswa berdasarkan nama dengan regex
def cari_mahasiswa(nama):
    found = False
    for student in siswa:
        if re.search(nama, student['nama'], re.IGNORECASE):
            print(f"Mahasiswa dengan nama '{nama}' ditemukan: {student}")
            found = True
    if not found:
        print(f"Mahasiswa dengan nama '{nama}' tidak ditemukan.")

# Program utama
while True:
```

```

print("\nMenu:")
print("1. Tambah Data Siswa")
print("2. Tampilkan Data Siswa")
print("3. Cari Mahasiswa")
print("4. Keluar")
pilihan = input("Pilih menu: ")

if pilihan == '1':
    nama = input("Masukkan nama siswa: ")
    umur = input("Masukkan usia siswa: ")
    kelas = input("Masukkan kelas siswa: ")
    tambah_data(nama, umur, kelas)
elif pilihan == '2':
    tampilkan_data()
elif pilihan == '3':
    nama_cari = input("Masukkan nama mahasiswa yang ingin dicari: ")
    cari_mahasiswa(nama_cari)
elif pilihan == '4':
    print("Terima kasih!")
    break
else:
    print("Menu tidak valid. Silakan coba lagi.")

```

Output :

1. Menu Utama

```

Menu:
1. Tambah Data Siswa
2. Tampilkan Data Siswa
3. Cari Mahasiswa
4. Keluar|

```

2. Tambah Data Siswa

```

Menu:
1. Tambah Data Siswa
2. Tampilkan Data Siswa
3. Cari Mahasiswa
4. Keluar|
Pilih menu: 1
Masukkan nama siswa: Muhammad Alif
Masukkan usia siswa: 20
Masukkan kelas siswa: C 2022
Data siswa Muhammad Alif berhasil ditambahkan.

```

3. Lihat Data Siswa

```
Menu:
1. Tambah Data Siswa
2. Tampilkan Data Siswa
3. Cari Mahasiswa
4. Keluar
Pilih menu: 2
Data siswa:
1. Nama: Muhammad Alif, Usia: 20, Kelas: C 2022
```

4. Cari Data Siswa

```
Menu:
1. Tambah Data Siswa
2. Tampilkan Data Siswa
3. Cari Mahasiswa
4. Keluar
Pilih menu: 3
Masukkan nama mahasiswa yang ingin dicari: Alif
Mahasiswa dengan nama 'Alif' ditemukan: {'nama': 'Muhammad Alif', 'umur': 20,
'kelas': 'C 2022'}
```

Opini terkait pembelajaran :

Untuk opini saya sendiri mengenai apa yang telah diajari di kampus adalah kita dapat mengevaluasi diri tentang program dasar, terutama untuk bahasa program **Python** itu sendiri karena dari banyak orang yang telah maju kedepan untuk uji kemampuan banyak yang masih salah. Sehingga membuktikan kalau orang tersebut sudah melupakan atau memang kurang mampu dalam memahami pemrograman.

Sistem Pembelajaran itu sendiri menurut saya sudah cukup baik, mungkin lebih bagus lagi jika interaktif mahasiswa yang lebih paham agar mahasiswa lain dapat mengerti juga dengan pemahaman mahasiswa yang maju tersebut untuk memperbaiki kesalahan orang yang telah memberikan jawaban sebelumnya. Dan setelah itu dijelaskan kembali oleh dosen untuk diluruskan jika ada yang salah dalam penjelasan mahasiswa yang maju untuk memperbaiki temannya tersebut.

Apa yang saya dapat setelah mempelajari kembali Python? Saya awalnya memang banyak lupa tentang fungsional yang dimiliki oleh python dengan adanya pembelajaran kembali ini membuat saya mengingat kembali dan menambah wawasan lagi karena ada beberapa hal yang ternyata saya belum ketahui jika ditelusuri lebih dalam.

Kedepannya ingin seperti apa? Kedepannya, saya berharap dapat berkontribusi dengan cara mengajarkan apa yang telah saya pelajari kepada adek tingkat atau mahasiswa yang mungkin mengalami kesulitan serupa. Saya percaya bahwa saling berbagi pengetahuan dan pengalaman adalah kunci untuk meningkatkan pemahaman dan kemampuan dalam pemrograman. Selain itu, saya juga berencana untuk membagikan modul laporan ini kepada mereka sebagai referensi untuk pembelajaran mereka ke depannya.

Tetapi kekurangan yang saya dapat adalah, transisi dari bahasa program java ke python, memang ini kurang relevan dengan yang diajarkan oleh dosen sebelumnya. Begitu juga pada praktikum sehingga perlu penyesuaian kembali untuk memahami struktur bahasa program tersebut.