

# Bab I

## Intro to Python

Python sebagai bahasa pemrograman yang populer dan komprehensif dengan menggabungkan kapabilitas, sintaksis kode yang jelas serta dilengkapi pustaka standar yang mempunyai fungsionalitas sangat besar. Python termasuk dari jajaran bahasa pemrograman tingkat tinggi seperti bahasa pemrograman C, C++, Java, Perl dan Pascal.

### A. Keywords

Keywords adalah kata yang dicadangkan oleh Python sehingga tidak boleh digunakan sebagai nama variabel, fungsi atau identifier lainnya. Keywords sendiri digunakan untuk mendefinisikan sintaks dan struktur dari bahasa Python. Keyword dalam Python merupakan case sensitive.

Ada 33 keyword dalam Python 3.7. Semua keywords selain True, False, dan None ditulis dalam lowercase dan harus ditulis apa adanya. Berikut adalah keywords dalam Python.

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

### B. Identifiers

Identifiers merupakan nama yang digunakan untuk mengidentifikasi class, function, variable, dll. Identifiers berguna untuk membedakan suatu entitas dengan entitas lainnya.

Aturan dalam menulis identifiers :

1. Identifiers dapat berupa kombinasi dari huruf dalam lowercase (a-z) atau uppercase (A-Z) atau digit (0-9) atau underscore (\_). Contoh : myClass, var\_1, this\_is\_a\_long\_variable
2. Identifiers tidak boleh diawali dengan digit.

```
1variable # Penulisan Salah  
variable1 # Penulisan Benar
```

3. Keywords tidak dapat digunakan sebagai identifiers.

```
True = 1
```

Output:

```
File "d:\Code\tes.py", line 1  
  True = 1  
    ^  
SyntaxError: invalid syntax
```

4. Tidak diperbolehkan menggunakan symbol special seperti !, @, #, \$, %, dll.

```
x$ = 1
```

Output:

```
File "d:\Code\tes.py", line 1  
  x$ = 1  
    ^  
SyntaxError: invalid syntax
```

5. Sebuah identifier dapat memiliki panjang berapapun.
6. Identifier Class dimulai dengan huruf uppercase sedangkan semua identifiers dimulai dengan huruf lowercase.
7. Identifier yang dimulai dengan underscore (`_<name>`) menandakan identifier tersebut merupakan identifier private.
8. Identifier yang dimulai dengan dua buah underscore (`__<name>`) menandakan identifier tersebut merupakan identifier yang sangat private.
9. Jika identifier tersebut juga diakhiri dengan dua buah underscore (`_<name>_`), identifier tersebut merupakan nama yang telah ditetapkan Python.

### Hal yang perlu diingat!

- Python merupakan bahasa pemrograman yang case-sensitive. Sehingga, Variable dan variabel tidak sama.

## C. Statements

Statements adalah instruksi atau pernyataan yang diberikan untuk dieksekusi oleh mesin. Penulisan statements dalam Python tidak diakhiri dengan tanda titik koma (;). Contohnya sebagai berikut:

```
x = 1
y = x
z = x + y
```

Dalam Python, akhir dari statement ditandai dengan baris baru. Tapi, kita dapat memperpanjang sebuah statement lebih dari beberapa baris secara eksplisit dengan menggunakan karakter forward slash (\). Contohnya :

```
x = 1 + 2 + 3 + \
    4 + 5 + 6 + \
    7 + 8 + 9
```

Multi-line statements juga terdapat dalam tanda kurung (), kurung siku [], kurung kurawal {}. Sebagai contoh, kita dapat mengimplementasikan contoh di atas menjadi :

```
x = (1 + 2 + 3 +
    4 + 5 + 6 +
    7 + 8 + 9)
colors = ['red',
          'blue',
          'green']
```

Kita juga dapat menyingkat penulisan statements menjadi satu baris menggunakan tanda titik koma ;.

```
a = 1; b = 2; c = 3
```

#### D. Indentation

Sebagian besar Bahasa pemrograman seperti C, C++, dan Java menggunakan kurung kurawal {} untuk mendefinisikan sebuah blok kode sedangkan Python menggunakan indentasi. Indentation sendiri adalah penulisan yang menjorok masuk ke dalam dari sebuah kode.

Sebuah blok kode (body dari sebuah function, loop, etc) dalam Python dimulai dengan indentasi dan diakhiri dengan baris yang tidak diindentasi. Jumlah spasi dari indentasi itu bebas, tetapi jumlah spasinya harus konsisten. Biasanya, 4 spasi digunakan sebagai indentasi dan lebih dipilih daripada tab. Sebagai contoh:

- Penulisan Benar

```
for i in range(1, 20):  
    if i == 3:  
        print("it's three")  
        Break
```

- Penulisan Salah

```
for i in range(1, 20):  
if i == 3:  
print("it's three")  
Break
```

Penggunaan dari indentasi dalam Python membuat kode terlihat rapi dan bersih sehingga menghasilkan sebuah kode yang terlihat mirip dan konsisten. Indentasi membuat kode tersebut menjadi lebih mudah dibaca. Sebagai contoh:

```
if True:  
    print('Yes')  
x = 10
```

akan lebih mudah dibaca daripada,

```
if True:print('Yes'); x = 10
```

Indentasi yang salah akan menghasilkan error **IndentationError**.

## E. Comments

Comments sangatlah penting dalam penulisan program. Comments membantu mendeskripsikan isi dari kode tersebut sehingga orang lain tidak sulit dalam memahami kode yang kita tulis. Penulisan comment dalam Python terbagi menjadi:

- Single-line comment

```
#This is a single-line comment
```

- Multi-line comment

```
"""  
This is  
a  
Multi-line comments """
```

## F. Variables & Constant

Variabel merupakan representasi dari alamat memori yang digunakan untuk menyimpan nilai dari data. Sintaks dari penulisan variabel adalah **name = value**. Contohnya,

```
x = 10
```

Di sini, kita telah membuat sebuah variabel bernama **x** dan telah memberinya value **10**. Variabel dapat kita anggap sebagai tas untuk menyimpan buku di dalamnya dan buku itu dapat diganti kapan saja. Hal ini berarti sebuah value dari variabel dapat diubah-ubah. Sebagai contoh:

```
x = 10
print(x)
x = 10.5
print(x)
```

Output:

```
10
10.5
```

Kita juga dapat memberikan beberapa nilai ke beberapa variabel sekaligus. Contoh:

```
x, y, z = 1, 3.2, "System Information"
print(x)
print(y)
print(z)
```

Output:

```
1
3.2
System Information
```

Jika kita ingin menetapkan value yang sama ke banyak variabel sekaligus, kita dapat melakukannya seperti:

```
x = y = z = "System Information"
print(x)
print(y)
print(z)
```

Output:

```
System Information
System Information
System Information
```

Constant merupakan sebuah tipe variabel yang valuenya tidak dapat diubah. Constant dapat kita anggap sebagai sebuah tas untuk menyimpan sebuah buku yang isinya tidak dapat diubah lagi. Contoh dari constant adalah

```
PI = 3.14  
GRAVITY = 9.8
```

Aturan dan Ketentuan dalam penamaan variabel dan constant:

1. Nama constant dan variabel harus memiliki kombinasi huruf kecil (a-z) atau huruf besar (A-Z) atau angka (0-9) atau garis bawah (\_)

```
snake_case  
MACRO_CASE  
camelCase  
CapWords
```

2. Buat nama yang masuk akal. Misalnya length lebih masuk akal daripada l.
3. Jika ingin menulis nama variabel yang lebih dari dua kata, gunakan garis bawah untuk memisahkannya.

```
car_name  
this_is_a_variable
```

4. Gunakan huruf kapital untuk mendeklarasikan sebuah constant.

```
PI  
G  
MASS  
SPEED_OF_LIGHT  
TEMP
```

5. Jangan gunakan simbol spesial seperti !, @, #, \$, %, etc.
6. Jangan memulai nama variabel dengan angka.

## G. Data Types

Data types adalah klasifikasi atau kategorisasi item data yang mewakili jenis nilai yang memberi tahu operasi apa yang dapat dilakukan pada data tertentu. Karena semuanya adalah objek dalam Python, tipe data sebenarnya adalah sebuah class dan variable adalah instance (object) dari class ini.

Ada berbagai macam tipe data di Python yang sering digunakan, sebagai berikut:

## 1. Python Numbers

Dalam Python, tipe data numerik mewakili data yang memiliki value numerik. Nilai numerik dapat berupa Integer, Float, dan bilangan Complex. Nilai – nilai ini didefinisikan sebagai kelas int, float, dan complex dalam Python.

- a. Integer – Nilai ini diwakili oleh kelas int. Integer berisi bilangan bulat positif atau negative (tanpa pecahan atau decimal).
- b. Float – Nilai ini diwakili oleh kelas float. Float adalah bilangan real dengan representasi floating point atau ditentukan oleh titik decimal. Secara opsional, karakter e atau E yang diikuti dengan bilangan bulat positif atau negative dapat ditambahkan untuk menentukan notasi ilmiah. Nilai float sendiri hanya akurat hingga 15 angka decimal.
- c. Complex number. Bilangan kompleks diwakili oleh kelas complex. Ini dispesifikasikan sebagai (bagian real) \_ (bagian imajiner). Misalnya  $2 + 3j$ .

Kita dapat menggunakan fungsi type() untuk mengetahui tipe dari tipe data tersebut.

```
x = 10
print("Type of x: ", type(x))

y = 10.2
print("Type of y: ", type(y))

z = 3 + 5j
print("Type of z: ", type(z))
```

Output:

```
Type of x: <class 'int'>
Type of y: <class 'float'>
Type of z: <class 'complex'>
```

## 2. Python Strings

Dalam Python, String adalah array byte yang mewakili karakter Unicode. String adalah kumpulan dari satu atau lebih karakter yang ditulis dalam tanda kutip tunggal '`<text>`', tanda kutip ganda "`<text>`", tanda kutip

tiga `'<text>'` atau `<text>''''`. Dalam Python, tidak ada tipe data **char** sehingga char adalah string dengan panjang satu. String sendiri diwakili oleh class `str`.

```
s = 'This is a single quotes string'
print(s)
s = "This is a double quotes string"
print(s)
# String dengan tanda kutip tiga dapat #
# membuat multi-line string
s = """This is a triple
    quotes string"""
print(s)
```

Output:

```
This is a single quotes string
This is a double quotes string
This is a triple
quotes string
```

Karena string merupakan sebuah array, maka operator slicing `[ ]` dapat digunakan untuk mengakses karakter dalam string.

```
s = "Hello World"
print(s[6])
print(s[6:11])
```

Output:

```
W
World
```

### 3. Python List

List adalah sebuah urutan item yang berurutan. List merupakan salah satu tipe data yang paling sering digunakan dalam Python dan sangat fleksibel yang berarti semua item dalam list tidak harus bertipe sama. Untuk mendeklarasi sebuah list

```
a = [1, 2.2, 'python']
```

### 4. Python Tuple

Tuple sendiri mirip dengan dengan list tetapi item dalam tuple tidak dapat diubah. Tuple setelah dibuat tidak dapat dimodifikasi. Ini dikarenakan

tuple digunakan untuk melindungi data dan biasanya lebih cepat daripada list yang dapat berubah secara dinamis. Tuple sendiri dideklarasikan menggunakan tanda kurung (<item>, <item>, ...) dan itemnya dipisah menggunakan tanda koma.

```
a = (1, 2.2, 'python')
```

## 5. Python Set

Set adalah sebuah koleksi dari berbagai item unik. Set didefinisikan oleh item yang dipisah oleh koma di dalam sebuah kurung kurawal {<item>, <item>, ...}. Item dari sebuah set tidak terurut. Karena set itu memiliki value unik maka set akan mengeliminasi value yang duplikat.

```
a = {1, 2.2, 'python'}
```

## 6. Python Dictionary

Dictionary adalah sebuah koleksi tak terurut dari pasangan key-value. Dictionary sering digunakan ketika berhadapan dengan data yang besar karena adanya key-value sehingga pengambilan data lebih optimal. Setiap pasangan key-value dari dictionary dipisahkan oleh tanda titik dua :, di mana setiap key dipisahkan oleh koma dan setiap key tidak boleh sama (unik). Dalam Python, dictionary dibuat dengan menaruh kumpulan pasangan key-value tersebut ke dalam sebuah kurung kurawal {key: value, key: value, ...}. Key dari dictionary ialah case sensitive, yang berarti jika sebuah key memiliki nama yang sama namun penulisannya berbeda maka akan dianggap sebagai key yang berbeda.

```
a = {'python': 1, 'Python': 1}
```

## 7. Python Boolean

Boolean adalah tipe data yang dapat menampung dua nilai, yaitu **True** dan **False**, operasi dengan *Logical Operator* juga akan menghasilkan nilai boolean, sehingga tipe data ini biasa digunakan dalam penyeleksian kondisi dan perulangan. Nilai **True** akan mengembalikan nilai **1** dan **False** akan mengembalikan nilai **0**.

```
is_raining = True  
is_walking = False  
x = (1 == True)
```

```
y = (1 == False)
a = True + 4
b = False + 10
```

Selain tipe data di atas, berikut adalah tipe data lengkap dari Python:

- **Numeric data types:** *int, float, complex*
- **String data types:** *str*
- **Sequence types:** *list, tuple, range*
- **Binary types:** *bytes, bytearray, memoryview*
- **Mapping data type:** *dict*
- **Boolean type:** *bool*
- **Set data types:** *set, frozenset*

## H. Conversion of Data Types

Sebuah proses pengubahan nilai dari suatu tipe data ke tipe data yang lainnya disebut konversi tipe data. Python sendiri memiliki 2 jenis konversi data:

- **Implicit Type Conversion**

Dalam konversi implisit, Python akan secara otomatis mengubah tipe data dari nilai tersebut. Proses ini tidak membutuhkan campur tangan dari user.

Sebagai contoh:

```
num_int = 345
num_flo = 3.45

new = num_int + num_flo

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

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

Output:

```
datatype of num_int: <class 'int'>
datatype of num_flo: <class 'float'>
Value of new: 348.45
datatype of new: <class 'float'>
```

Dari program di atas dapat dilihat bahwa kita akan melakukan operasi penjumlahan dari variabel integer **num\_int** dan float **num\_flo**. Hasil dari penjumlahan tersebut akan menghasilkan sebuah value yang bertipe data **float**, hal ini terjadi karena Python akan selalu mengkonversi tipe data kecil ke tipe data besar untuk menghindari adanya kehilangan data.

- **Explicit Type Conversion**

Dalam konversi eksplisit, user mengkonversi tipe data dari sebuah objek ke tipe data yang dibutuhkan dengan bantuan fungsi konversi tipe seperti **int()**, **float()**, **str()**, dll. Proses konversi tipe ini biasa disebut sebagai *typecasting*. Syntax : **<required\_datatypes>(expression)**.

```
num_int = 123
num_str = "456"

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

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

num_sum = num_int + num_str

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

Output:

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

Dari program di atas dapat dilihat bahwa kita akan melakukan operasi penjumlahan dari variabel integer **num\_int** dan string **num\_str**. Karena akan terjadi error bila tipe data integer dan string dijumlahkan langsung, maka perlu dilakukan konversi tipe data string ke integer dengan menggunakan fungsi **int()**. Setelah dilakukan konversi maka kedua variabel tersebut dapat dijumlahkan dan menghasilkan sebuah value yang bertipe data **integer**.

## I. Input

Untuk menambah fleksibilitas dalam program, kita mungkin mau mengambil input dari user. Python menyediakan fungsi `input()` untuk mengambil inputan dari user lalu menyimpannya dalam sebuah variabel. Syntax dari `input()` adalah:

```
input([prompt])
```

dimana `prompt` adalah sebuah string yang ingin ditampilkan secara opsional. Fungsi dari `input()` akan mengembalikan sebuah value string.

```
name = input("Input a name : ")
print(name)
```

Output:

```
Input a name : Sistem Informasi
Sistem Informasi
```

## J. Operators

Operator adalah sebuah symbol special dalam Python yang berguna untuk menjalankan komputasi aritmatika atau logika. Value yang dioperasikan oleh operator disebut operand.

Dalam Python ada beberapa operator, sebagai berikut:

### 1. Arithmetic Operators

Operator aritmatik digunakan untuk menjalankan operasi matematika seperti penjumlahan, pengurangan, perkalian, dll.

Operator	Arti	Contoh
+	Menjumlahkan dua operand atau unary plus	$x + y + 2$
-	Mengurangi operand kanan dari kiri atau unary minus	$x - y -- 2$
*	Mengalikan operand kiri dengan kanan	$x * y$
/	Membagikan operand kiri dengan kanan (akan menghasilkan float)	$x / y$
%	Modulus – hasil bagi dari operand kiri dengan kanan	$x \% y$

//	Floor Division – pembagian yang hasilnya menjadi bilangan bulat disesuaikan ke kiri pada garis bilangan	$x // y$
**	Pangkat – operand kiri dipangkatkan oleh operand kanan	$x ** y$

## 2. Comparison (Relational) Operators

Operator komparasi digunakan untuk membandingkan value dan akan mengembalikan nilai **True** atau **False** berdasarkan kondisinya.

Operator	Arti	Contoh
>	Lebih besar dari – True jika operand kiri lebih besar dari kanan	$x > y$
<	Kurang dari – True jika operand kiri lebih kecil dari kanan	$x < y$
==	Sama dengan – True jika kedua operand bernilai sama	$x == y$
!=	Tidak sama dengan – True jika kedua operand tidak bernilai sama	$x != y$
>=	Lebih besar atau sama dengan – True jika operand kiri lebih besar atau sama dengan kanan	$x >= y$
<=	Lebih kecil atau sama dengan – True jika operand kiri lebih kecil atau sama dengan kanan	$x <= y$

## 3. Logical (Boolean) Operators

Operator	Arti	Contoh
and	True jika kedua operand bernilai True	$x \text{ and } y$
or	True jika kedua atau salah satu operand bernilai True	$x \text{ or } y$
not	True jika operand bernilai False	$\text{not } x$

#### 4. Bitwise Operators

Operator bitwise bertindak pada operand seolah-olah mereka adalah string digit binary.

**Pada tabel di bawah :** misalkan  $x = 10$  (0000 1010) dan  $y = 4$  (0000 0100)

Operator	Arti	Contoh
&	Bitwise AND	$x \& y = 0$ (0000 0000)
	Bitwise Or	$x   y = 14$ (0000 1110)
~	Bitwise NOT	$\sim x = -11$ (1111 0101)
^	Bitwise XOR	$x \wedge y = 14$ (0000 1110)
>>	Bitwise right shift	$x >> 2 = 2$ (0000 0010)
<<	Bitwise left shift	$x << 2 = 40$ (0010 1000)

#### 5. Assignment Operators

Operator assignment digunakan dalam Python untuk menetapkan nilai ke variable.

Operator	Arti	Sama Dengan
=	$x = 5$	$x = 5$
+=	$x += 5$	$x = x + 5$
-=	$x -= 5$	$x = x - 5$
*=	$x *= 5$	$x = x * 5$
/=	$x /= 5$	$x = x / 5$
%=	$x \% = 5$	$x = x \% 5$
//=	$x //= 5$	$x = x // 5$
**=	$x ** = 5$	$x = x ** 5$
&=	$x \& = 5$	$x = x \& 5$
=	$x   = 5$	$x = x   5$
^=	$x \wedge = 5$	$x = x \wedge 5$

>>=	x >>= 5	x = x >> 5
<<=	x <<= 5	x = x << 5

## 6. Identity Operator

Operator	Arti	Contoh
is	True jika kedua operand identik	x is True
is not	True jika operand tidak identik	x is not True

## 7. Membership Operator

Operator	Arti	Contoh
in	True jika value/variable ditemukan dalam collections	x in True
not in	True jika value/variable tidak ditemukan dalam collection	x not in True

# Bab II

## Conditional Statement

*Conditional statement* (yang dalam bahasa Indonesia disebut pernyataan bersyarat atau pengkondisian) adalah fitur dari bahasa pemrograman yang melakukan perhitungan atau tindakan yang berbeda tergantung pada kondisi yang ditentukan programmer. Pengkondisian membuat program memiliki cabang yang masing-masing akan tereksekusi jika persyaratannya terpenuhi.

Pengkondisian biasanya dalam bentuk *if statement*, yaitu salah satu fitur utama dari bahasa pemrograman, tidak terkecuali Python. Hampir tidak ada bahasa pemrograman yang tidak memiliki *if statement* dan hampir tidak ada cara untuk memprogram tanpa cabang dalam aliran kode (setidaknya jika kode tersebut perlu memecahkan masalah yang kompleks).

Bahasa pemrograman seperti C, C++, dan Java setidaknya memiliki fitur conditional statement seperti *if statement* dan *switch case*, namun dalam bahasa Python *switch case* baru diadaptasi pada versi 3.10 keatas yang dikenal sebagai “*structural pattern matching*” atau “*match case*”. Artinya, pada versi sebelumnya pengkondisian hanya dapat dilakukan menggunakan *if statement*. Ekspresi terdiri dari satu atau beberapa operator perbandingan dan operator logika yang akan menghasilkan nilai True (benar) atau False (salah).

**Tabel Operator Perbandingan**

Operator	Arti	Contoh	Hasil
>	Lebih besar dari – True jika operand kiri lebih besar dari kanan	5 > 6	False
<	Kurang dari – True jika operand kiri lebih kecil dari kanan	5 < 6	True
==	Sama dengan – True jika kedua operand bernilai sama	5 == 5	True
!=	Tidak sama dengan – True jika kedua operand tidak bernilai sama	5 != 5	False
>=	Lebih besar atau sama dengan – True jika operand kiri lebih besar atau sama dengan kanan	5 >= 3	True

<=	Lebih kecil atau sama dengan – True jika operand kiri lebih kecil atau sama dengan kanan	5 <= 5	True
----	--	--------	------

**Tabel Operator Logika**

Operator	Arti	Contoh	Hasil
and	True jika kedua operand bernilai True	True and True	True
or	True jika kedua atau salah satu operand bernilai True	True or False	True
not	True jika operand bernilai False	not False	True

## A. If-else Statement

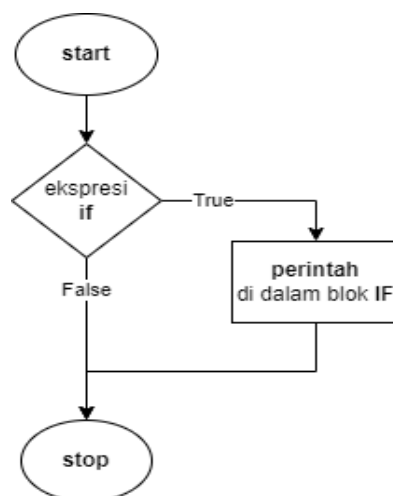
### 1. If statement

*If statement* adalah jenis pengkondisian yang paling mendasar, di mana kode dieksekusi apabila ekspresi terpenuhi atau bernilai *True* (benar). Pernyataan dari *if statement* harus memiliki *indent* minimal sepanjang satu spasi di awal tiap baris kode. Suatu pernyataan dapat berupa satu baris atau satu blok kode.

if ekspresi:

```
# Pernyataan/statement(dieksekusi jika ekspresi #
    bernilai True)
```

**Alur Program :**



**Contoh 1**

```
angka = -5  
  
if angka > 0:  
    print(angka, "adalah bilangan positif")  
  
print("pernyataan ini bernilai benar!")
```

**Output:**

```
> 5 adalah bilangan positif.  
> pernyataan ini bernilai True!.
```

**Contoh 2**

```
angka = 5  
  
if angka < 0:  
    print(angka, "adalah bilangan negatif") print("pernyataan  
ini bernilai False!")
```

**Output:**

```
> pernyataan ini bernilai False!.
```

**Contoh 1.3**

```
angka = -2  
  
if angka > 0:  
    print(angka, "adalah bilangan positif") if  
angka < 0:  
    print(angka, "adalah bilangan negatif")  
  
print("pernyataan ini bernilai benar!")
```

### Output:

> -2 adalah bilangan negatif.

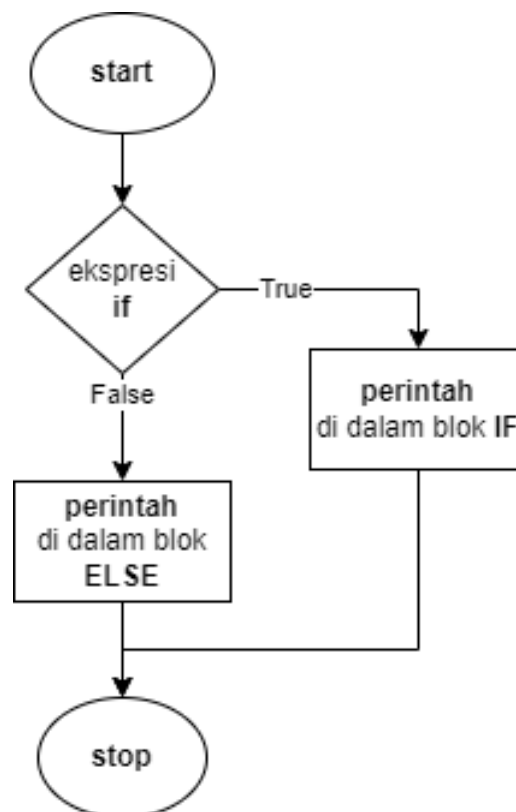
## 2. if-else statement

Pernyataan “else” digunakan ketika bagian benar dan salah dari kondisi tertentu ditentukan untuk dieksekusi. Ketika kondisinya benar, pernyataan di dalam blok if dieksekusi; jika kondisinya salah, program pada blok else akan dieksekusi.

Pernyataan if...else dalam Python memiliki sintaks berikut:

```
if ekspresi:  
  
    # pernyataan/statement(dieksekusi jika #  
    ekspresi bernilai True)  
else:  
  
    # pernyataan/statement(dieksekusi jika #  
    ekspresi bernilai False
```

### Alur program:



### Contoh 1

```
angka = 5  
if angka >= 0:  
    print("bilangan positif atau nol") else:  
    print(angka, "adalah bilangan negatif")
```

### Output:

```
> bilangan positif atau nol
```

### Contoh 2.2

```
x = 5  
y = 20  
if x == y:  
    print("x dan y bernilai sama")  
else:  
    print("x tidak sama dengan y")
```

### Output:

```
> x tidak sama dengan y
```

## 1. If-elif-else statement

Pernyataan elif memungkinkan Anda untuk memeriksa beberapa ekspresi dan mengeksekusi blok kode segera setelah salah satu kondisi mengevaluasi ke True. Dalam hal ini, kondisi if dievaluasi terlebih dahulu. Jika salah, pernyataan elif akan dieksekusi, jika itu juga salah, pernyataan else akan dieksekusi.

Pernyataan If...Elif..else dalam Python memiliki sintaks berikut:

if ekspresi:

#dieksekusi jika ekspresi bernilai True#

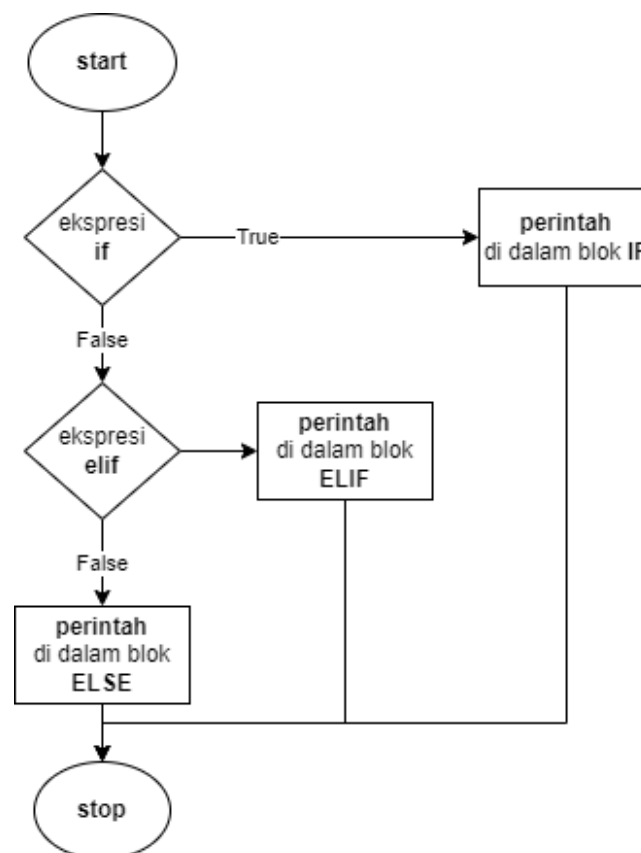
elif ekspresi:

#dieksekusi jika ekspresi (elif) bernilai True

else:

#dieksekusi jika ekspresi bernilai False

**Alur program:**



**Contoh 3**

angka = 0

if angka > 0:

```
print(angka, "adalah bilangan positif")
elif angka == 0:
    print(angka, "adalah bilangan nol")
else:
    print(angka, "adalah bilangan negatif")
```

**Output:**

```
> 0 adalah bilangan nol
```

## Contoh 2

```
angka = 0
if angka > 0:
    print(angka, "adalah bilangan positif")
elif angka == 0:
    print(angka, "adalah bilangan nol")
else:
    print(angka, "adalah bilangan negatif")
```

**Output:**

```
> 0 adalah bilangan nol
```

## 2. Nested IF Statement

Pernyataan IF bersarang adalah pernyataan di mana pernyataan If terletak di dalam pernyataan If lainnya. Ini digunakan ketika variabel harus diproses lebih dari sekali. Pernyataan if, if-else, dan if...elif...else dapat digunakan dalam program.

Pernyataan if bersarang dalam Python memiliki sintaks berikut:

if ekspresi:

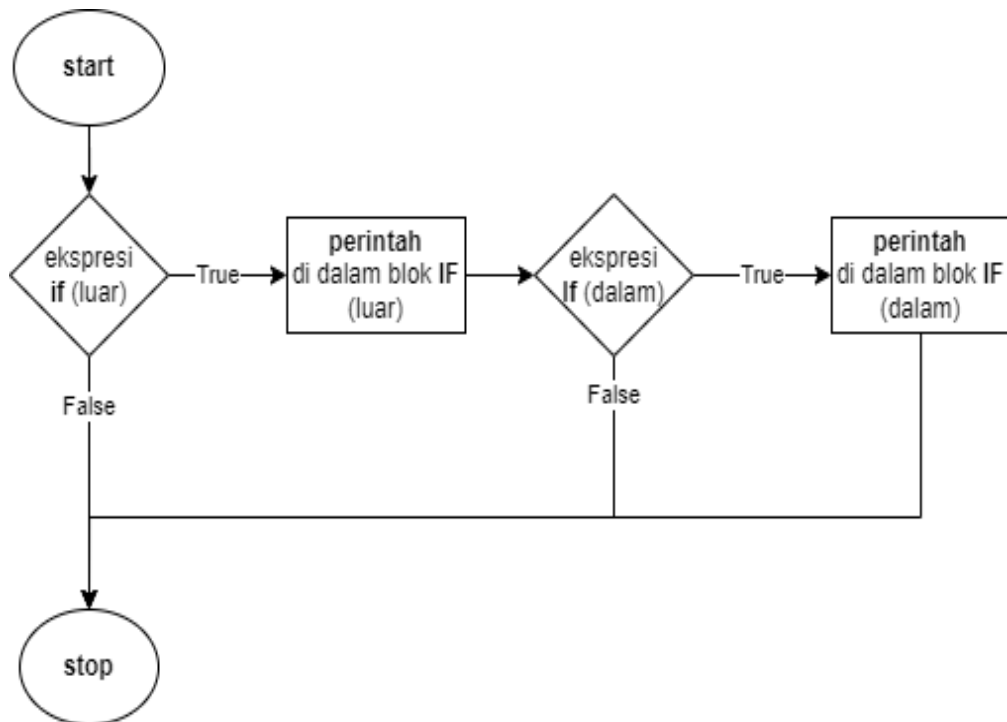
#dieksekusi jika ekspresi bernilai True# if

ekspresi:

# dieksekusi jika ekspresi if (luar) dan

# if (dalam) bernilai True

**Alur program:**



**Contoh 1**

angka = 0

if angka >= 0:

```
print(angka, "adalah bilangan positif") if
angka == 0:
    print(angka, "adalah bilangan nol")
else:
    print(angka, "adalah bilangan negatif")
```

**Output:**

```
> 0 adalah bilangan nol
```

**Contoh 2**

```
angka = 4
if angka >= 0:
    print(angka "adalah bilangan positif") if
(angka % 2) == 0:
        print(angka, "adalah bilangan genap
        positif")
else:
    print(angka, "adalah bilangan negatif")
```

**Output:**

```
> 0 adalah bilangan nol
```

**3. Shorthand if statement**

Shorthand if statement digunakan ketika hanya satu statement yang perlu dieksekusi di dalam blok if. Pernyataan ini dapat disertakan di baris yang sama dengan pernyataan If. Pernyataan

Short Hand if dalam Python memiliki sintaks berikut:

```
if kondisi: statement
```

**Contoh program:**

```
a = 123  
  
if a > 100: print(a, "lebih besar dari seratus")
```

**Output:**

```
> 123 lebih besar dari 100
```

#### 4. ShortHand if-else statement

Shorthand if-else digunakan untuk menyebutkan pernyataan If-else dalam satu baris di mana hanya ada satu pernyataan untuk dieksekusi di blok if dan else.

Short Hand if-else dalam Python memiliki sintaks berikut:

```
statement (True) if kondisi else statement (False)
```

**Contoh program:**

```
a = 123  
  
print(a, " besar") if a > 100 else print(a, " kecil")
```

**Output:**

```
> 123 besar
```

## B. Match Case

Pernyataan switch mengevaluasi ekspresi, mencocokkan nilai ekspresi terhadap serangkaian klausa kasus, dan mengeksekusi pernyataan setelah klausa kasus pertama dengan nilai yang cocok, hingga pernyataan break ditemukan. Klausa default dari pernyataan switch akan dilompati jika tidak ada kasus yang cocok dengan nilai ekspresi.

Short Hand if dalam Python memiliki sintaks berikut:

```
match variabel:

    case nilai-1:

        statement-1

    case nilai-2:

        statement-2

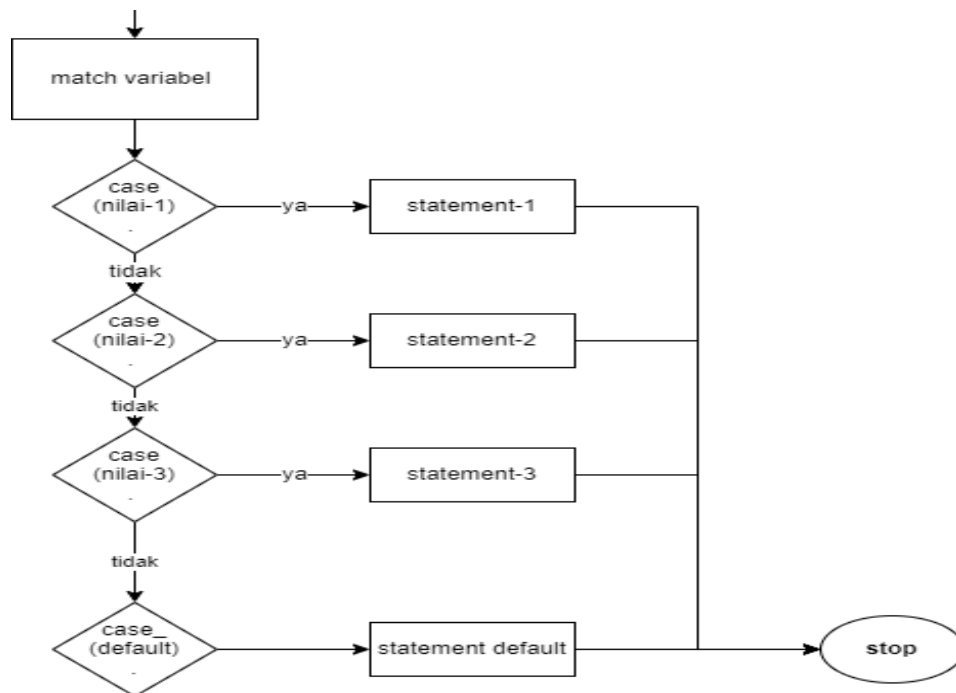
    case nilai-...:

        statement-...

    case _:

        statement default (jika tidak ada case yang sesuai)
```

**Alur program:**



**Contoh program:**

```
bahasa = "python"
match bahasa:
    case "JavaScript":
        print("kamu akan menjadi web developer.")

    case "Python":
        print("kamu akan menjadi Data Scientist")

    case "PHP":
        print("kamu akan menjadi backend developer")

    case "Solidity":
        print("kamu akan menjadi Blockchain developer")

    case "Java":
        print("kamu akan menjadi mobile app developer")

    case _:
        print("bahasa tidak penting, yang penting adalah
            mampu untuk menyelesaikan masalah.")
```

**Output:**

```
> kamu akan menjadi Data Scientist
```

# Bab III

## Looping

Looping atau perulangan merupakan cara yang digunakan untuk menjalankan perintah yang berulang untuk data berbentuk kelompok seperti list, tuple atau string. Dalam melakukan perintah looping biasanya digunakan perintah For Loop dan While Loop, serta penerapan pernyataan Break dan Continue sebagai pilihan tambahan dalam membuat perintah looping.

### A. For Loop

Perintah “For” digunakan dalam melakukan perulangan pada data kelompok yang sudah diketahui jumlah iterasinya dan akan berhenti jika iterasinya telah dieksekusi semuanya.

#### Contoh:

```
fruits = ["apel", "pisang", "mangga"]  
for x in fruits:  
    print("Ini adalah", x)
```

#### Output:

```
Ini adalah apel  
Ini adalah pisang  
Ini adalah mangga
```

### B. While Loop

Perintah “While” digunakan dalam melakukan perulangan pada data kelompok yang tidak diketahui jumlah pasti iterasinya. Oleh karena itu, biasanya perintah while akan terus melakukan perulangan selama suatu kondisi masih terpenuhi. Selain itu, pada looping juga kita bisa tambahkan kondisi “else” seperti layaknya pada Conditional Statement pada bab sebelumnya.

**Contoh:**

```
n = int(input("Enter n: "))

sum = 0
i = 1

while i <= n:
    sum = sum + i
    i = i+1
else: #Penambahan kondisi else optional
    print("The sum is", sum)
```

**Output:**

```
Enter n: 5
The sum is 15
```

**C. Break dan Continue**

Penggunaan “Break” dan “Continue” digunakan untuk menambahkan kondisi perintah pada looping sehingga dalam looping tersebut akan ada kondisi “If” yang ditambahkan. Break digunakan jika kita ingin menghentikan sebuah looping jika suatu kondisi telah terpenuhi, sedangkan continue digunakan untuk melewati iterasi sekarang dan melanjutkan langsung ke iterasi selanjutnya jika kondisinya terpenuhi.

**1. Break Statement**

```
fruits = ["apel", "pisang", "mangga"]
for x in fruits:
    if x == "pisang":
        Break
    print("Ini adalah", x)
```

**Output:**

```
Ini adalah apel
```

## 2. Continue Statement

Sama halnya pada “For Loop” statement di atas juga dapat digunakan pada “While Loop” dengan penulisan yang sama. Menambahkan conditional statement ke dalam looping yang dibuat.

```
fruits = ["apel", "pisang", "mangga"]  
for x in fruits:  
    if x == "pisang":  
        continue  
    print("Ini adalah", x)
```

**Output:**

```
Ini adalah apel  
Ini adalah mangga
```

## D. Nested Loop

Nested loop atau perulangan bersarang adalah sebuah perulangan yang di dalamnya terdapat perulangan yang lain. Baik itu perulangan While-For, While-While, For-For atau berbagai macam kombinasi Nested Loop.

**Contoh:**

```
adj = ["red", "tasty"]  
fruits = ["apple", "banana"]  
  
for x in adj: #Loop Pertama  
    for y in fruits: #Loop Kedua  
        print(x, y)
```

**Output:**

```
red apple  
red banana  
tasty apple  
tasty banana
```

## E. Try-Exception

Ketika terjadi kesalahan, atau exception, Python biasanya akan berhenti dan menghasilkan pesan kesalahan. Pengecualian ini dapat ditangani menggunakan pernyataan try:

### Contoh 1 :

```
x = 3
y = 'Empat'
try:
    z = x + y
except:
    z = str(x) + y

print(z)
```

3Empat

### Contoh 2:

```
x = 3
y = 'Empat'
try:
    z = x + y
except:
    z = y + str(x)
else:
    print('Tidak dapat menjumlahkan keduanya')

print(z)
```

Empat3

### Contoh 3:

```
x = 3
y = 'Empat'
try:
    z = x + y
except:
    z = y + str(x)
finally:
    print('!!! salah satunya dilakukan konversi type data')

print(z)
```

!!! salah satunya dilakukan konversi type data  
Empat3

# Bab IV

## Functions

Dalam konteks pemrograman, *function* atau fungsi didefinisikan sebagai sekumpulan baris perintah atau kode yang dikelompokkan menjadi satu kesatuan yang akan dieksekusi ketika dia dipanggil. Sebuah *function* bisa menerima parameter, bisa mengembalikan suatu nilai, dan bisa dipanggil berkali-kali secara independen. Tujuan dari *function* adalah untuk memecah program yang besar dan kompleks menjadi bagian-bagian kecil dengan tugasnya masing-masing.

### A. Pembuatan Function

Dalam bahasa pemrograman Python, *function* dibuat dengan sintaks sebagai berikut:

```
def <function_name>(parameters):  
  
    statements
```

#### Keterangan:

1. *def* adalah sintaks yang mendefinisikan bahwa blok kode program adalah sebuah *function*
2. <function\_name> adalah nama dari *function*
3. (parameters) adalah nilai-nilai (satu atau lebih) yang dikirimkan ke dalam *function* yang akan dieksekusi di dalam *function body*
4. *statements* adalah kumpulan perintah yang akan dieksekusi ketika *function* dipanggil

#### Catatan:

Blok kode program di dalam Python didefinisikan dengan indentasi

#### Contoh Function:

```
def hello_world():  
  
    print('Hello python! Hello World!')
```

## B. Pemanggilan Function

*Function* dipanggil dengan menuliskan nama fungsinya diikuti tanda kurung () sebagai berikut:

```
hello_world()
```

**Output:**

```
Hello python! Hello World!
```

Function juga dapat digunakan atau dipanggil berkali-kali:

```
hello_world()
```

```
hello_world()
```

```
hello_world()
```

**Output:**

```
Hello python! Hello World!
```

```
Hello python! Hello World!
```

```
Hello python! Hello World!
```

## C. Function dengan Parameter

Sebuah *function* dapat menerima satu atau lebih parameter atau argumen yang merupakan nilai yang dikirimkan ke dalam fungsi untuk dieksekusi dalam *function body*.

```
def welcome(name):  
    print(f'Hello {name}, welcome!')
```

**Keterangan:**

*name* adalah sebuah parameter yang dikirimkan ke dalam fungsi untuk ikut dieksekusi pada perintah *print()*

```
welcome('Human')
```

```
welcome('Robot')  
  
welcome('Alien')
```

**Output:**

```
Hello Human, welcome!  
  
Hello Robot, welcome!  
  
Hello Alien, welcome!
```

#### D. Function dengan Parameter Wajib

Parameter di dalam Python dapat bersifat wajib, artinya parameter tersebut harus diisi. Perhatikan contoh berikut:

```
def introduce(name, origin):  
  
    print(f"I'm {name} from {origin}")
```

Parameter *name* dan *origin* adalah parameter wajib. Jika *function* tersebut dipanggil dengan kedua parameternya:

```
introduce('Human', 'Earth')
```

**Output:**

```
I'm Human from Earth
```

Jika *function* tersebut dipanggil hanya dengan salah satu parameternya:

```
introduce('Human')
```

**Output:**

```
Exception has occurred: TypeError  
  
introduce() missing 1 required positional argument:  
'origin'
```

Terjadi *exception* karena terdapat parameter tidak lengkap atau hilang.

## E. Function dengan Parameter Opsional

Parameter di dalam Python juga dapat bersifat opsional, artinya parameter yang dapat tidak diisi karena memiliki nilai *default*. Perhatikan contoh berikut:

```
def introduce (name, origin='Somewhere'):  
    print(f"I'm {name} from {origin}")
```

Parameter *name* adalah parameter wajib sedangkan *origin* adalah parameter opsional. Jika *function* tersebut dipanggil dengan parameter wajibnya:

```
introduce('Human')
```

**Output:**

```
I'm Human from Somewhere
```

Jika *function* tersebut dipanggil dengan parameter wajib dan parameter opsionalnya:

```
introduce('Alien', 'Space')
```

**Output:**

```
I'm Alien from Space
```

## F. Function dengan Parameter Berurut dan Tidak Berurut

Sebuah *function* dapat memiliki parameter opsional yang lebih dari satu, artinya terdapat beberapa parameter yang dapat diisi ataupun tidak diisi karena memiliki nilai default. Perhatikan contoh berikut:

```
def introduce (name, origin='Somewhere', power='no  
power'):  
    print(f"I'm {name} from {origin}, I have {power}")
```

*name* adalah parameter wajib, sedangkan *origin* dan *power* adalah parameter opsional.

Jika *function* tersebut dipanggil dengan parameternya secara lengkap dan berurut:

```
introduce('Human', 'Earth', 'a mind')
```

**Output:**

```
I'm Human from Earth, I have a mind
```

Jika *function* tersebut dipanggil dengan parameternya secara lengkap tapi tidak berurut:

```
introduce('Human', 'a mind', 'Earth')
```

**Output:**

```
I'm Human from a mind, I have Earth
```

Jika *function* tersebut dipanggil dengan parameternya secara tidak lengkap tapi berurut:

```
introduce('Human', 'Earth')
```

**Output:**

```
I'm Human from Earth, I have no power
```

Jika *function* tersebut dipanggil dengan parameternya secara tidak lengkap dan tidak berurut:

```
def introduce('Human', 'a mind')
```

**Output:**

```
I'm Human from a mind, I have no power
```

Dari keempat program diatas diketahui bahwa yang menghasilkan output yang benar adalah function yang parameternya dipanggil secara berurutan. Sedangkan, *function* yang parameternya dipanggil secara tidak berurutan menghasilkan output yang salah.

Pemanggilan *function* dengan parameter yang tidak berurutan dapat dilakukan dengan mendefinisikan nama parameter beserta nilainya untuk menghasilkan output yang benar. Perhatikan contoh berikut:

```
introduce(name='Alien', power='a strength',
origin='Space')
```

**Output:**

```
I'm Alien from Space, I have a strength
```

## G. Function yang Mengembalikan dan Tidak Mengembalikan Nilai

Ditinjau dari segi pengembalian nilai, *function* dapat dikategorikan menjadi 2 bagian:

### 1. Function yang Mengembalikan Nilai

Function yang mengembalikan nilai adalah jenis *function* yang jika dipanggil akan mengembalikan nilai sebagai hasil dari pemanggilan *function* tersebut. Nilai tersebut dapat digunakan untuk perintah lebih lanjut seperti menyimpan nilai dalam variabel, melakukan perhitungan, atau mencetak nilai, dsb.

Perhatikan contoh berikut:

```
def add2values(a, b):
    result = a + b
    return result
```

Dalam sebuah *function*, untuk mengembalikan nilai digunakan sintaks *return*. Jika *statement return* telah dieksekusi pada sebuah *function*, maka semua proses yang ada di dalam blok code function tersebut akan berhenti dieksekusi.

```
#Case01 (just call the function)
add2values(1, 2)

#Case02 (store the function in a variable)
a = add2values(1, 2)

#Case3 (print the function)
print("1 + 2 =", add2values(1, 2))
```

**Output:**

```
#Output01
```

```
#Output02
```

```
#Output03
```

```
1 + 2 = 3
```

## 2. Fungsi yang Tidak Mengembalikan Nilai

Fungsi yang tidak mengembalikan nilai adalah fungsi yang ketika dipanggil hanya mengeksekusi perintah pada *body function* tanpa mengembalikan nilai yang ditandai dengan tidak adanya *statements return*.

Perhatikan contoh berikut:

```
def add2values(a, b):  
    result = a + b  
    print(a, '+', b, '=', result)  
  
add2values(1, 2)
```

**Output:**

```
1 + 2 = 3
```

## H. Function Rekursif

*Function* rekursif adalah *function* yang memanggil dirinya sendiri dalam *function body*-nya yang menyebabkan efek perulangan. Perulangan ini bisa berhenti ketika kondisi tertentu tercapai, atau bisa juga bersifat tak terbatas, atau mungkin bahkan bisa menimbulkan error karena pemanggilan fungsi yang tak ada habisnya.

*Function* rekursif dengan perulangan terbatas;

```
def show_intervals(start, end):  
    print(start)  
    start = start + 1  
  
    if(start <= end):  
        show_intervals(start, end)  
  
show_intervals(1, 10)
```

**Output:**

```
1  
2  
3  
.  
.  
.  
9  
10
```

*Function* rekursif dengan perulangan tanpa batas;

```
def show_multiple(value):  
    print(value)  
    value = value + value  
    show_multiple(value)  
  
show_multiple(2)
```

**Output:**

2

4

6

8

10

. . .

. . .

. . .

RecursionError: maximum recursion depth exceeded while  
calling a Python object