# Daftar Isi

[**Daftar Isi 1**](#_heading=h.arclthjkzcdp)

[**Konteks Modul 2**](#_heading=h.b7bo7h6tp569)

[Sasaran 2](#_heading=h.k537bq8vli0)

[Topik 2](#_heading=h.v6ew6gbhq9lj)

[Requirement 2](#_heading=h.lm19xpwxmhxz)

[**Control Flow 3**](#_heading=)

[**Percabangan (if, if-else, dan elif) 3**](#_heading=)

[1. Pengenalan percabangan 3](#_heading=h.g1voddewudvo)

[2. Percabangan dengan blok if 6](#_heading=h.20b9o37c0p6y)

[3. Percabangan dengan blok if-else 7](#_heading=h.odnt9y7ktt72)

[4. Percabangan dengan blok elif 8](#_heading=h.l5u63m39miai)

[5. Percabangan dengan blok nested if 11](#_heading=h.uwgboyxw12ub)

[6. Logika Operator 13](#_heading=h.dwrmu17h490o)

[**Pengulangan/Iterasi (for dan while) 14**](#_heading=)

[1. Pengenalan pengulangan 14](#_heading=h.go5vmca0n30j)

[2. Pengulangan dengan blok for 14](#_heading=h.9j4i501eyul3)

[3. Pengulangan dengan blok while 24](#_heading=h.c8ruyd5iew6t)

[4. Pengulangan bertingkat 32](#_heading=h.i7uiv4ibp4tb)

[**Fungsi (def) 41**](#_heading=h.umsjkeex6mfk)

[1. Pengenalan Fungsi 41](#_heading=h.27sx4i2modj4)

[2. Fungsi dengan arguments dan parameters 42](#_heading=h.1ucn3tdw2c92)

[3. Fungsi dengan return types 46](#_heading=h.4uc8ix3d3kfe)

[4. Fungsi recursive 52](#_heading=h.ok8sx7y0y88)

# Konteks Modul

### Sasaran

1. Peserta memahami konsep *Flow* logika pemrograman
2. Peserta mengetahui berbagai jenis *Flow* logika pemrograman seperti percabangan, pengulangan dan fungsi
3. Peserta bisa melakukan penulisan *Flow* pemrograman seperti percabangan, pengulangan dan fungsi dengan menggunakan bahasa pemrograman *Python*

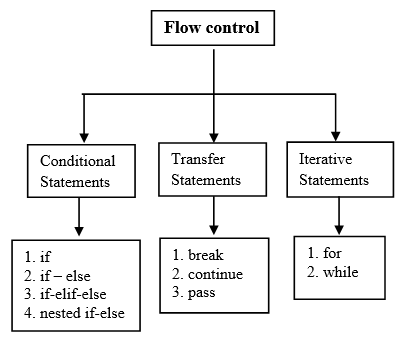
### Topik

1. *Control Flow*
2. Percabangan (*if*, *if-else*, dan *elif*)
3. Pengulangan/iterasi (*for* dan *while*)
4. Fungsi (*def*)

### Requirement

1. Google Collab
2. Support all os laptop device
3. Contoh *hands-on*: [lihat disini](https://drive.google.com/file/d/1xkIvtJynJc6JGg7cJFCDGi9gyAfn7no2/view?usp=sharing)

# *Control Flow*



Mengontrol aliran program (*controls flow*) merupakan mekanisme untuk mengatur urutan atau aliran eksekusi dari sebuah kode program. Terdapat tiga macam aliran program di Python, yaitu sequential *control flow, decision control flow*, dan *loop control flow*

Konsep loops sangat penting di bahasa pemrograman manapun. Terkadang, kita akan sering berhadapan dengan situasi di mana kita perlu mengeksekusi sepotong kode berulang kali, tetapi kita tidak ingin menulis baris kode yang sama beberapa kali. Loop dapat melakukan hal tersebut.

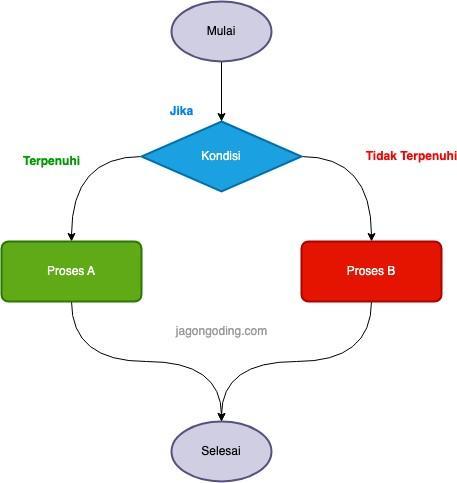
Singkatnya, *loop* atau iterasi merupakan kemampuan untuk menjalankan atau mengeksekusi kode program berulang kali. Salah satu contoh iterasi yang telah kita pelajari sebelumnya adalah rekursif. Terdapat dua jenis *loops* dalam *Python*, yaitu *while* *loop* dan *for loop*.

# Percabangan (*if*, i*f-else*, dan *elif*)

## Pengenalan percabangan

Percabangan dalam dunia pemrograman adalah proses penentuan keputusan atau dalam bahasa inggris ini biasa disebut sebagai *conditional statement*.

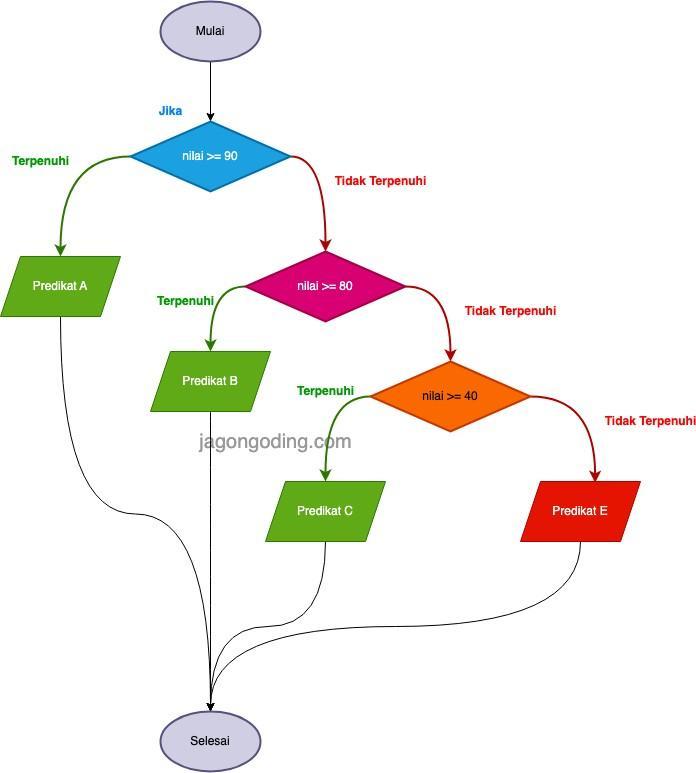
Konsepnya sederhana: kita menginstruksikan komputer untuk melakukan suatu aksi tertentu (X), hanya jika suatu kondisi Y terpenuhi. Atau, kita memerintahkan komputer untuk melakukan aksi tertentu (A) jika suatu kondisi tidak terpenuhi.



Premisnya adalah “Jika-Maka”. Contoh:

* Jika nilai siswa lebih dari atau sama dengan 90, maka dia dapat predikat A.
* Jika nilai siswa lebih dari atau sama dengan 80, maka dia dapat predikat B.
* Jika nilai siswa lebih dari atau sama dengan 40, maka dia dapat predikat C.
* Dan seterusnya.

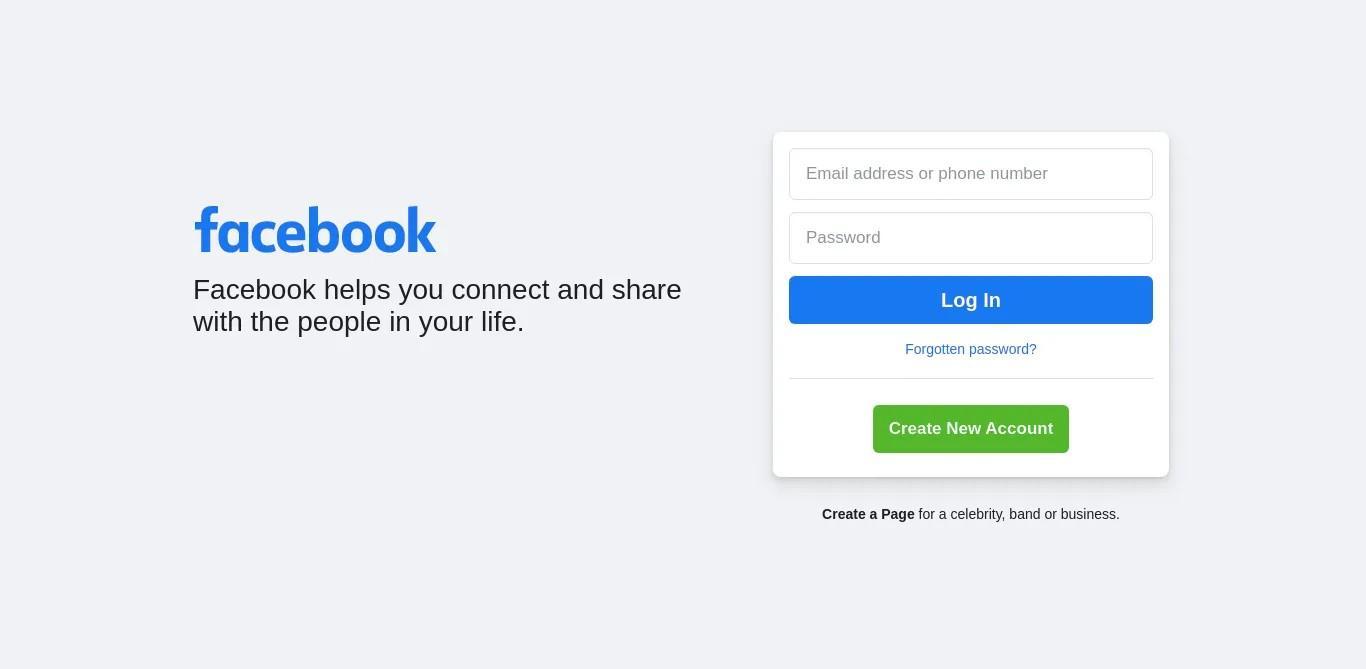
Ilustrasi:



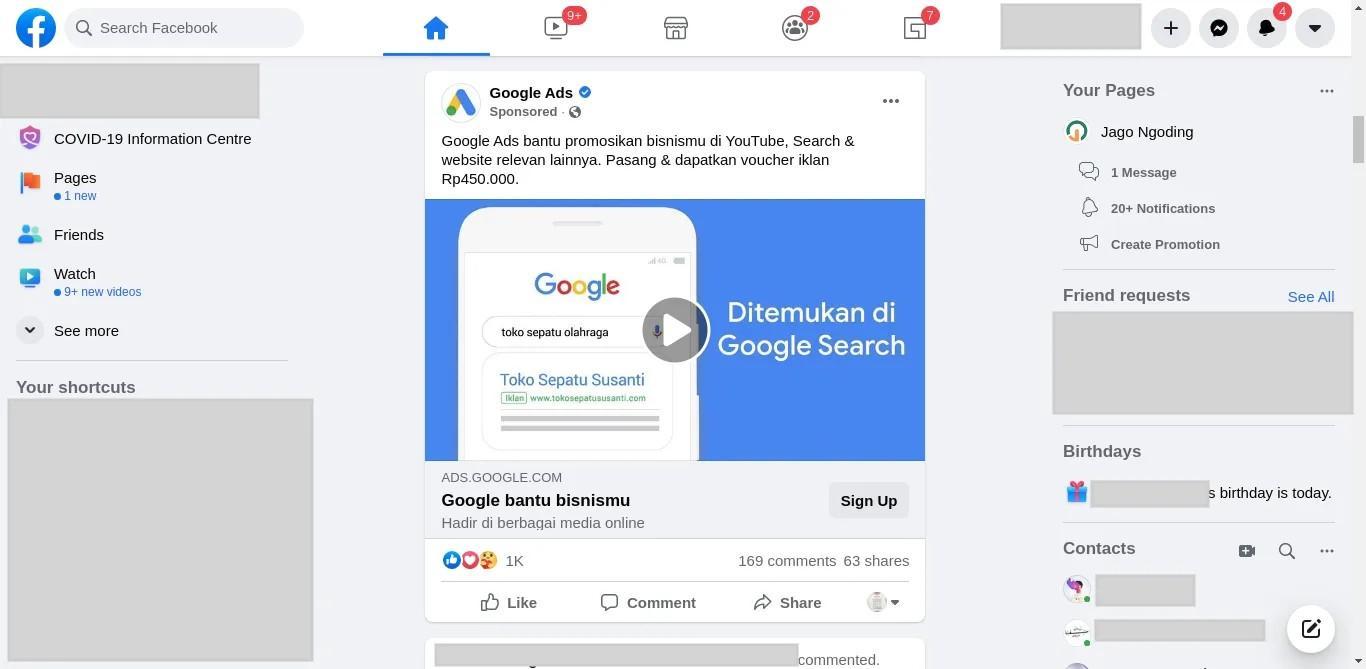
Contoh kehidupan nyata:

Contoh dari kehidupan nyata yang saya yakin semuanya pernah mengalami, adalah saat mengakses facebook.com.

1. Ketika mengakses facebook dan kita dalam kondisi belum login, kita akan mendapatkan halaman ini:



1. Sedangkan jika kita dalam kondisi login, maka kita akan mendapatkan halaman seperti ini:



Itu adalah contoh di kehidupan nyata di mana jika suatu kondisi terpenuhi, maka suatu aksi akan dijalankan (dalam hal ini adalah menampilkan halaman A), dan jika kondisi tidak terpenuhi maka yang ditampilkan adalah halaman B.

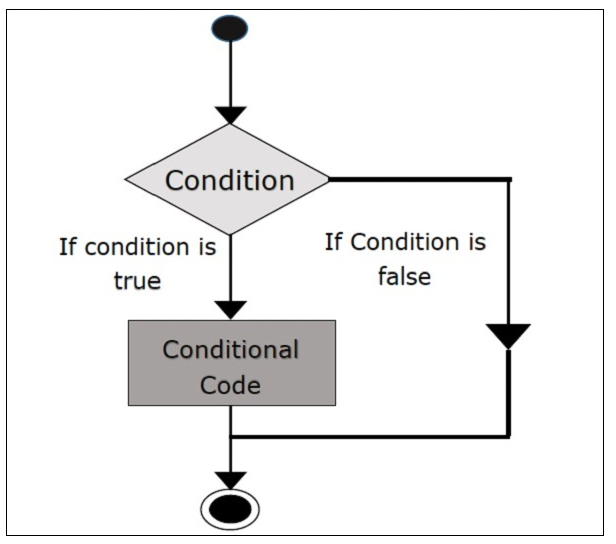
Bisa kita simpulkan, bahwa di setiap program atau aplikasi, baik yang berbasis web, desktop, mobile, maupun CLI tidak akan pernah lepas dari yang namanya percabangan. Sehingga, logika percabangan adalah hal yang sangat dasar dan sangat penting sekali untuk dikuasai ketika kita mempelajari sebuah bahasa pemrograman tertentu.

Dalam bahasa pemrograman python, syntax atau statement yang digunakan untuk melakukan percabangan adalah:

* if
* elif
* dan else

Di mana if merupakan kondisi utama, sedangkan elif adalah kondisi kedua atau ketiga hingga ke-x, sedangkan else adalah kondisi terakhir di mana semua kondisi sebelumnya tidak ada yang terpenuhi.

## Percabangan dengan blok *if*



Blok kode *if* pada python, strukturnya seperti ini:

**if** kondisi**:** statements**()**

Bagian kondisi adalah sebuah variabel / atau nilai yang bertipe data *boolean*. Baik berupa nilai *True/False* secara langsung, ataupun sebuah ekspresi logika.

Jika kondisi bernilai *True*, maka *statements()*akan dieksekusi oleh sistem. Tulis dan perhatikan kode program berikut:

**if** **True**:

print**(**'Kode program ini akan dieksekusi'**)**

**if** **False**:

print**(**'Kode program ini tidak akan dieksekusi'**)**

print**(**'Kode program ini akan selalu dieksekusi karena tidak termasuk pada percabangan'**)**

Jika kita jalankan, kode program di atas hanya akan memunculkan output:

Kode program ini akan dieksekusi

Kode program ini akan selalu dieksekusi

Kenapa?

Karena kondisi *if* yang kedua tidak bernilai *True*, sehingga statemen yang ada di dalamnya pun tidak akan pernah dieksekusi oleh sistem.

Dan *print()* yang ke-3 akan selalu dieksekusi karena ia berada di luar blok kode *if*.

Ingat bahwa blok kode di dalam python ditentukan oleh indentasi seperti yang telah kita bahas pada [Aturan Penulisan Sintaks Python](https://jagongoding.com/python/dasar/aturan-sintaks-python)



Selain menggunakan boolean secara langsung, kita juga bisa menggunakan ekspresi logika untuk percabangan.

Perhatikan contoh berikut:

**if 5 > 10:** *#* ❌

print**(**'Nilai 5 lebih dari 10'**)**

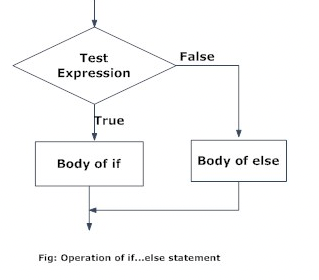
**if 10 > 5:** *#* ✅

print**(**'Nilai 10 lebih dari 5'**)**

Jika dijalankan, program di atas hanya akan menampilkan output:

Nilai **10** lebih dari **5**

## Percabangan dengan blok *if-else*



Yang kedua adalah blok if else. Blok if else ini biasa dinamakan percabangan, karena memiliki setidaknya 2 cabang:

1. Cabang if
2. Cabang else

Perhatikan contoh berikut:

nilai **= 50**

print**(**'Nilai anda adalah:'**,** nilai**,** '\n'**)**

**if** nilai **>= 70:**

print**(**'Selamat, anda lulus!'**)**

**else:**

print**(**'Maaf, anda tidak lulus.'**)**

Jika dijalankan, program di atas akan menghasilkan output:

Nilai anda adalah:**50**

Maaf, anda tidak lulus.

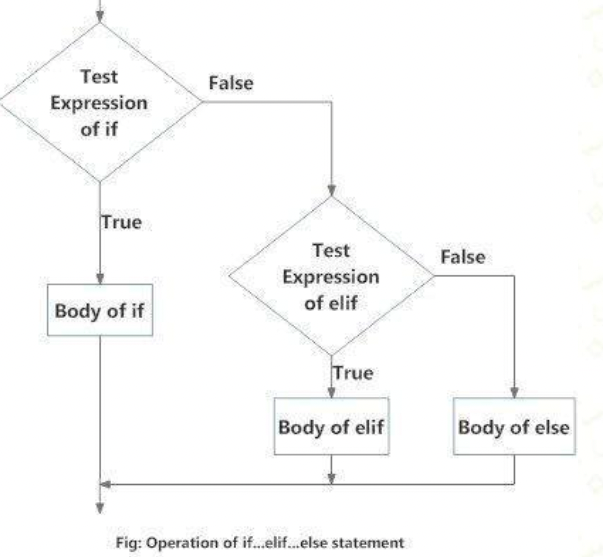
Anda bisa mengganti nilai = 50 menjadi [inputan user](https://jagongoding.com/python/dasar/percabangan/(/python/dasar/input)) seperti ini.

nilai **=** int**(**input**(**'Masukkan nilai anda: '**))**

…

Lalu jalankan kembali program di atas.

## Percabangan dengan blok *elif*



Sebagaimana pohon, cabangnya tidak hanya 2, tapi bisa 3, 4, 5 bahkan lebih.

Begitu juga pada logika kita. Kita bisa membuat lebih dari 2 cabang logika. Dan pada python, untuk membuat lebih dari 2 cabang, kita bisa menggunakan blok kode if..elif..else.

Agar lebih jelas, mari kita coba praktikkan.

Contoh yang paling umum digunakan untuk kasus percabangan if..elif..else adalah menentukan grade nilai suatu siswa.

Jika nilainya sekian, dia dapat predikat A. Sedangkan jika nilainya sekian maka predikatnya adalah B, dan seterusnya.

Berikut ini rules yang akan kita gunakan:

1. Predikat A untuk nilai >= 90
2. Predikat B untuk nilai >= 80 < 90
3. Predikat C untuk nilai >= 60 < 80
4. Predikat D untuk nilai >= 40 < 60
5. Selain itu, maka predikat E.

Dari 5 rule di atas, kita akan menggunakan satu if, 3 elif, dan 1 else. Perhatikan contoh berikut:

nilai **=** int**(**input**(**'Masukkan nilai: '**))**

**if** nilai **>= 90:**

print**(**'Predikat A'**)**

**elif** nilai **>= 80:**

print**(**'Predikat B'**)**

**elif** nilai **>= 60:**

print**(**'Predikat C'**)**

**elif** nilai **>= 40:**

print**(**'Predikat D'**)**

**else:**

print**(**'Predikat E'**)**

Jalankan program lalu memasukkan angka 80, maka kita akan mendapatkan output predikat B:

Masukkan nilai: 80

Predikat B

Kenapa Pakai elif? Kan Bisa Pakai if saja?

Oke. Sebelum kita jawab pertanyaan di atas, mari kita coba ubah kode program di atas menjadi menggunakan if semua, tanpa elif sama sekali.

nilai **=** int**(**input**(**'Masukkan nilai: '**))**

**if** nilai **>= 90:**

print**(**'Predikat A'

**if** nilai **>= 80:**

print**(**'Predikat B'**)**

**if** nilai **>= 60:**

print**(**'Predikat C'**)**

**if** nilai **>= 40:**

print**(**'Predikat D'**)**

**else:**

print**(**'Predikat E'**)**

Jalankan program, lalu coba masukkan angka 80. Sistem akan memberikan kita output seperti berikut:

Masukkan nilai:

**80**

Predikat B

Predikat C

Predikat D

Kenapa?

Karena setiap kali kita membuat blok if, itu artinya kita membuat satu pohon percabangan.

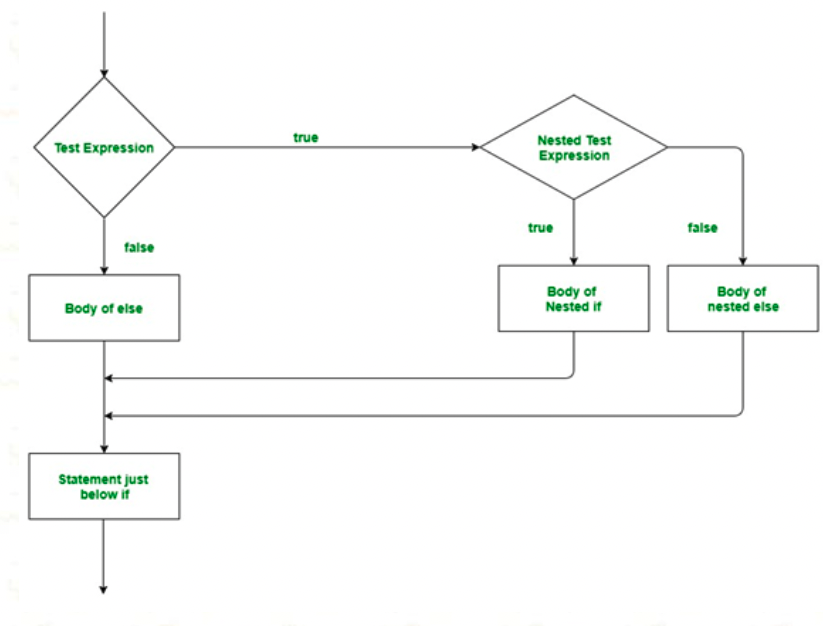
Dan pada pemrograman, pohon percabangan selalu hanya mengeluarkan satu kondisi yang terpenuhi.

Jika ada lebih dari satu kondisi yang terpenuhi, maka kondisi yang pertamalah yang akan dieksekusi. Tidak semuanya.

Oleh karena itu, di kode program yang pertama, kita sejatinya hanya membuat satu pohon percabangan, dan kondisi yang dieksekusi hanya 1, meskipun yang terpenuhi lebih dari 1.

Berbeda dengan contoh yang kedua, kita sejatinya sedang membuat 4 buah pohon percabangan, oleh karena itu outputnya pun akan berbeda sesuai dengan banyaknya pohon percabangan yang kita buat.

## Percabangan dengan blok *nested if*



1. Percabangan Satu Baris

Pada python, kita bisa menggunakan if..else dalam satu baris. Biasanya, pada bahasa pemrograman lainnya, ini disebut sebagai ternary (meskipun python cukup unik dari segi sintaksisnya).

Perhatikan contoh berikut:

nilai **=** int**(**input**(**'Masukkan nilai: '**))**

status **=** 'lulus' **if** nilai **>= 70 else** 'tidak lulus' print**(**status**)**

Kode program di atas adalah shortcut dari kode seperti di bawah:

**if** nilai **>= 70:**

status **=** 'lulus'

else:

status **=** 'tidak lulus'

print**(**status**)**

1. Percabangan Bertingkat

Percabangan bertingkat adalah sebuah istilah untuk if di dalam if. Kalau dalam dunia per-bakso-an, percabangan bertingkat adalah bakso beranak. Alias di dalam pentol ada pentol yang lainnya, Silakan coba dan jalankan kode program di bawah ini:

nilai **=** int**(**input**(**'Masukkan nilai: '**))**usia **=** int**(**input**(**'Masukkan usia: '**))**

**if** nilai **>= 75:  
 if (**usia **< 15):**

print**(**'Selamat adek, kamu lulus!'**)**

**else:**

print**(**'Selamat kakak, kamu lulus!'**)**

**else:**

**if (**usia **< 15):**

print**(**'Mohon maaf dek, coba lagi ya!'**)**

**else:**

print**(**'Mohon maaf kak, coba lagi ya!'**)**

Kode program di atas akan memeriksa terlebih dahulu apakah nilai yang dimasukkan adalah lulus atau tidak. Setelah itu, program akan memeriksa usia, apakah dia akan disapa dengan “kakak” atau kah dengan “adek”.

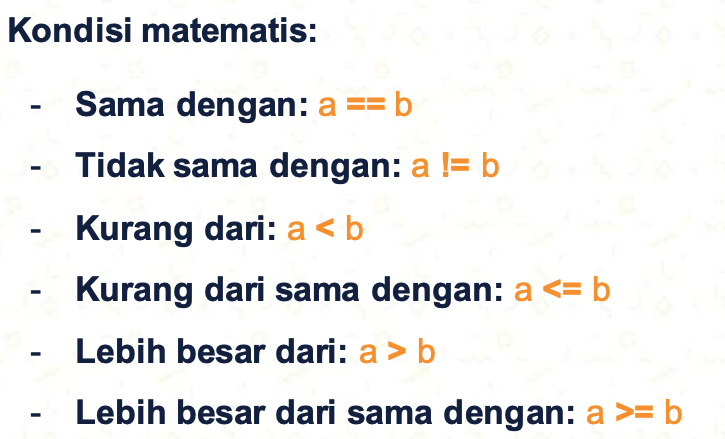
1. Contoh Program Percabangan Python

Selain contoh-contoh yang telah kita coba di atas, teman-teman bisa melihat contoh-contoh lainnya di seri [Latihan Program Python](https://jagongoding.com/python/latihan-logika/overview). Di situ ada berbagai macam contoh program untuk melatih logika, dan hampir setiap program menggunakan percabangan if-else seperti:

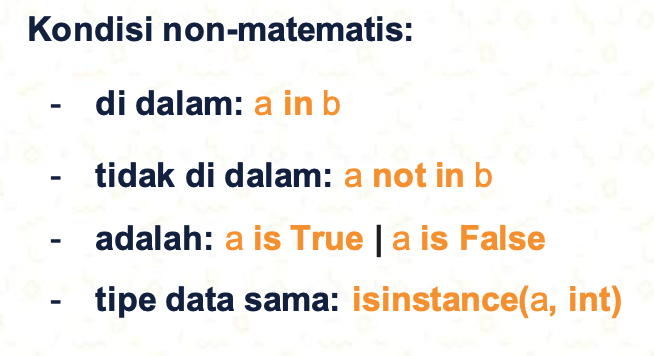
* [Angka Terbesar dari 3 Angka](https://jagongoding.com/python/latihan-logika/angka-terbesar-dari-3-angka/)
* [Memeriksa Bilangan Ganjil Genap](https://jagongoding.com/python/latihan-logika/bilangan-ganjil-genap/)
* [Menghitung Jumlah Huruf Vokal](https://jagongoding.com/python/latihan-logika/menghitung-huruf-vokal/)
* M[emeriksa Tahun Kabisat](https://jagongoding.com/python/latihan-logika/tahun-kabisat/)
* dan lain-lain

## Logika Operator

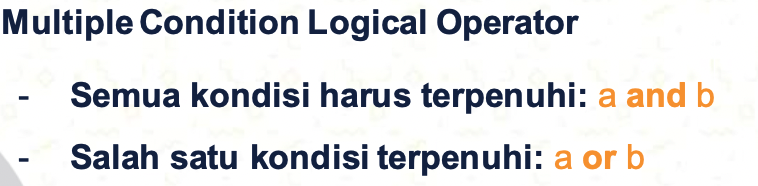
1. Kondisi Operator Matematis



1. Kondisi Operator Non-matematis



1. Kondisi *Multiple Logical Operator*

**

1. Contoh Penggunaan

Berikut ini contohnya untuk operator keanggotaan:

buah\_yang\_tersedia **= [**'jeruk'**,** 'mangga'**,** 'melon'**]**

buah\_yang\_dicari **=** input**(**'Masukkan nama buah dalam huruf kecil: '**)**

**if (**buah\_yang\_dicari **in** buah\_yang\_tersedia**):**

print**(**'Buah yang anda cari tersedia!'**)**

**else**:

print**(**'Buah yang anda cari tidak tersedia!'**)**

Contoh output ketika saya ketik ‘mangga’:

Masukkan nama buah dalam huruf kecil: mangga   
Buah yang anda cari tersedia!

# Pengulangan/Iterasi (*for* dan *while*)

## Pengenalan pengulangan

Perulangan dalam dunia pemrograman adalah baris kode atau instruksi yang dieksekusi oleh komputer secara berulang-ulang sampai suatu kondisi tertentu terpenuhi. Konsep perulangan ini didukung di semua bahasa pemrograman modern, termasuk di antaranya adalah python.

Dengan perulangan, kita bisa mengeksekusi suatu kode program berkali-kali dengan jumlah tertentu, atau selama sebuah kondisi tertentu terpenuhi.

Sebenarnya hampir sama seperti [percabangan python](https://jagongoding.com/python/dasar/percabangan) yang kita bahas pada pertemuan sebelumnya. Hanya saja bedanya:

* Kalau percabangan, blok kode yang memenuhi kondisi tertentu hanya akan dieksekusi satu kali saja.
* Sedangkan perulangan, ia akan dilakukan seterusnya berulang-ulang dengan jumlah tertentu atau selama kondisi tertentu terpenuhi.

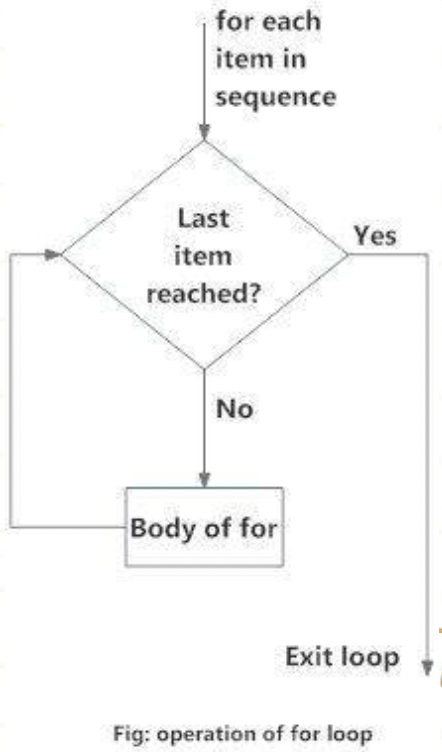
Pada python, kita bisa melakukan perulangan dengan beberapa cara diantaranya:

* [Perulangan for](https://jagongoding.com/python/dasar/perulangan-for)
* [Perulangan while](https://jagongoding.com/python/dasar/perulangan-while)
* F[ungsi rekursif](https://jagongoding.com/python/dasar/fungsi-rekursif)

Ketiga jenis perulangan tersebut akan kita bahas dalam 3 artikel yang berbeda.

Dan pada pertemuan kali ini, kita akan membahas perulangan for pada python terlebih dahulu. Silakan siapkan dulu kopi panasnya.

## Pengulangan dengan blok *for*



Perulangan for pada python adalah perintah yang digunakan untuk melakukan iterasi dari sebuah nilai sequence atau data koleksi pada python seperti [List](https://jagongoding.com/python/dasar/list), [Tuple](https://jagongoding.com/python/dasar/tuple), [String](https://jagongoding.com/python/dasar/tipe-data-dan-variabel/#tipe-data-string-teks) dan lain-lain.

For pada python memiliki perilaku yang berbeda dengan for pada kebanyakan bahasa pemrograman yang lain, karena pada python ia sangat berkaitan dengan data *sequence* atau data kolektif. Mungkin kalau dibandingkan dengan bahasa lain, for pada python lebih dikenal sebagai foreach.

**Syntax For**

Berikut ini adalah struktur sintaks metode for:

**for** nilai **in** sequence**:**

*# blok kode for*

Jadi, ada 3 bagian penting.

1. sequence: adalah sebuah nilai yang bersifat *iterable* alias bisa diulang-ulang. Di antara [tipe data](https://jagongoding.com/python/dasar/tipe-data-dan-variabel/) yang bersifat *sequence* atau *iterable* adalah:
   * [list](https://jagongoding.com/python/dasar/list/)
   * [tuple](https://jagongoding.com/python/dasar/tuple/)
   * [string](https://jagongoding.com/python/dasar/tipe-data-dan-variabel/#tipe-data-string-teks)
   * dan lain sebagainya
2. nilai: adalah setiap item yang diekstrak dari sequence
3. Blok kode: yaitu statemen-statemen atau perintah-perintah tertentu yang akan dieksekusi secara berulang.

**For dengan list**

Langsung saja ke contoh dan praktik.

Buka tools kalian, lalu buat sebuah variabel dengan [tipe data list](https://jagongoding.com/python/dasar/list/). Setelah itu kita akan menampilkan tiap item dari list yang telah kita buat. Perhatikan contoh berikut:

listKota **= [**

'Jakarta'**,** 'Surabaya'**,** 'Depok'**,** 'Bekasi'**,** 'Solo'**,** 'Yogyakarta'**,** 'Semarang'**,** 'Makassar'

]

**for** kota **in** listKota**:**

print**(**kota**)**

Jika dieksekusi, program di atas akan menghasilkan output:

Jakarta Surabaya Depok Bekasi Solo Yogyakarta Semarang Makassar

Mengetahui urutan iterasi for dengan list

Untuk mengetahui urutan iterasi for dengan list, kita bisa menggunakan fungsi enumerate.

Fungsi tersebut akan mengekstrak 2 buah nilai:

1. yang pertama adalah index: yaitu urutan iterasi yang ke berapa
2. dan item yang mana itu adalah nilai dari list itu sendiri.

Perhatikan contoh berikut:

listKota **= [**

'Jakarta'**,** 'Surabaya'**,** 'Depok'**,** 'Bekasi'**,** 'Solo'**,** 'Jogjakarta'**,** 'Semarang'**,** 'Makassar'

]

**for** i**,** kota **in** enumerate**(**listKota**):**

print**(**i**,** kota**)**

Kode program di atas sama saja seperti sebelumnya, kita hanya menambahkan fungsi enumerate() dan mem-*passing* variabel listKota sebagai parameter. Kita juga mengekstrak dua buah nilai yang kita kasih nama i dan kota.

Jila dijalankan, berikut adalah output yang kita dapat:

1. Jakarta
2. Surabaya
3. Depok
4. Bekasi
5. Solo
6. Jogjakarta
7. Semarang
8. Makassar

Perhatikan, urutannya dimulai dari 0, bukan dari angka 1.

**For dengan fungsi range()**

Selain dengan list, kita juga bisa menggunakan for dengan fungsi range(). Perhatikan contoh berikut:

*## 0 sampai 4*

**for** i **in** range**(5):** print**(**"Perulangan ke -"**,** i**)**

Output:

Perulangan ke - **0**Perulangan ke - **1**Perulangan ke - **2**Perulangan ke - **3**Perulangan ke - **4**

Dengan fungsi range, kita bisa melakukan perulangan dari 0, sampai kurang dari nilai range yang kita definisikan (yaitu 5 dalam contoh di atas). Sehingga hasil perulangan yang didapatkan adalah 0 sampai 4.

Kita bisa memulai range dari selain 0

Perhatikan contoh berikut:

*## 10 sampai 15*

**for** i **in** range**(10, 16):**

print**(**'i ='**,** i**)**

Perulangan di atas akan menghasilkan output:

i = 10

i = 11

i = 12

i = 13

i = 14

i = 15

Kita juga bisa mendefinisikan kelipatannya:

Perhatikan contoh berikut:

*## Bilangan genap kelipatan 2*

**for** i **in** range**(2, 12, 2):**

print**(**'i ='**,** i**)**

Pada contoh di atas, sistem akan melakukan perulangan dimulai dari angka 2, hingga kurang dari 12 dengan interval/kelipatan sebanyak 2.

Hasilnya:

i = 2

i = 4

i = 6

i = 8

i = 10

Untuk bilangan ganjil, kita mulai saja dari angka 1:

*## Bilangan ganjil kelipatan 2*

**for** bilangan\_ganjil **in** range**(1, 12, 2):**

print**(**bilangan\_ganjil**)**

Output:

1

**3**

5

**7**

9

**11**

Oiya, nama variabel-nya tidak harus i. Bisa terserah kita.

**For dengan tuple**

[Tuple](https://jagongoding.com/python/dasar/tuple/) adalah di antara [tipe data](https://jagongoding.com/python/dasar/tipe-data-dan-variabel/) yang bersifat *iterable*, sehingga kita juga bisa memperlakukannya sebagai objek perulangan mengguankan for.

Perhatikan contoh di bawah:

tupleBuah **= (**'Mangga'**,** 'Jeruk'**,** 'Apel'**,** 'Pepaya'**)**

**for** buah **in** tupleBuah**:**

print**(**buah**)**

Output:

Mangga Jeruk Apel Pepaya

**For dengan string**

[String](https://jagongoding.com/python/dasar/tipe-data-dan-variabel/#tipe-data-string-teks) pun demikian, bersifat iterable, sehingga bisa kita jadikan objek perulangan. Perhatikan contoh berikut:

**for** karakter **in** "Indonesia"**:**

print**(**karakter**)**

Jika dijalankan, output-nya:

I

n d o n

e s i a

Pada python, kita bisa menginterupsi dan juga men-skip suatu iterasi pada perulangan. Terdapat 2 perintah yang bisa kita gunakan, yaitu:

* break untuk interupsi (memberhentikan paksa) sebuah perulangan
* continue untuk menskip ke iterasi selanjutnya

Perhatikan contoh berikut:

**for** i **in** range**(10, 20):**

*# skip jika i == 15*

**if (**i **== 15):**

continue

print**(**i**)**

Output:

10

11

12

13

14

16 <-- Habis 14 langsung 16

17

18

19

Perhatikan output di atas, pada saat i == 15, perintah print(i) tidak dieksekusi dan justru di-skip ke iterasi berikutnya.

Atau…

Kita justru bisa memberhentikan paksa suatu perulangan sekalipun belum sampai ke iterasi yang terakhir.

**for** i **in** range**(10, 20):**

*# hentikan jika i == 15*

**if (**i **== 15):**

break

print**(**i**)**

Jika dijalankan:

10

11

12

13

14 <-- print terakhir sebelum terjadi break pada i == 15

Sistem akan memberhentikan perulangan ketika i == 15 dan belum sempat melakukan perintah print().

For … else?

Hmmm. Hampir mirip dengan if ... else. Tapi tugasnya berbeda.

Perulangan for jika ditambahkan blok else, maka perintah yang ada pada blok else hanya akan dieksekusi ketika perulangan selesai secara natural –*tanpa interupsi*.

Perhatikan contoh berikut:

listKota **= [**

'Jakarta'**,** 'Surabaya'**,** 'Depok'**,** 'Bekasi'**,** 'Solo'**,** 'Jogjakarta'**,** 'Semarang'**,** 'Makassar'

]

**for** kota **in** listKota**:**

print**(**kota**)**

else:

print**(**'Tidak ada lagi item yang tersisa'**)**

Jika dijalankan, program di atas akan menghasilkan output seperti ini:

Jakarta  
Surabaya  
Depok  
Bekasi  
Solo  
Jogjakarta  
Semarang  
Makassar

Tidak ada lagi item yang tersisa

For … Else + Break

Jika kita gabungkan for ... else dengan break, maka blok else hanya akan dieksekusi jika perintah break tidak dieksekusi.

Kita bisa memanfaatkan for ... else + break untuk pencarian sebuah item pada list. Perhatikan contoh berikut:

listKota **= [**'Jakarta'**,** 'Surabaya'**,** 'Depok'**,** 'Bekasi'**,** 'Solo'**,** 'Jogjakarta'**,** 'Semarang'**,** 'Makassar'

]

kotaYangDicari **=** input**(**'Ketik nama kota yang kamu cari: '**)**

**for** i**,** kota **in** enumerate**(**listKota**):**

*# kita ubah katanya ke lowercase agar # menjadi case insensitive*

**if** kota**.**lower**() ==** kotaYangDicari**.**lower**():** print**(**'Kota yang anda cari berada pada indeks'**,** i**) break**

else:

print**(**'Maaf, kota yang anda cari tidak ada'**)**

Program di atas akan meminta user untuk menginputkan nama kota yang ingin dicari. Jika kotanya maka akan kita kasih info indeks-nya berapa (dalam listKota), dan jika tidak ada maka perintah print() yang ada di blok else akan dieksekusi.

Coba jalankan. Kemudian kita input kata solo, ini hasilnya:

Ketik nama kota yang kamu cari:

solo

Kota yang anda cari berada pada indeks **4**

Jika kita cari pakai kota yang tidak ada di dalam list, begini hasilnya:

Ketik nama kota yang kamu cari:  
pontianak   
Maaf, kota yang anda cari tidak ada

Nah, harusnya sekarang sudah lebih jelas bagaimana cara for ... else bekerja, dan kapan blok kode else akan dieksekusi. Dia hanya akan dieksekusi ketika perulangan mencapai titik akhirnya (alias sudah tidak ada iterasi lagi yang tersisa).

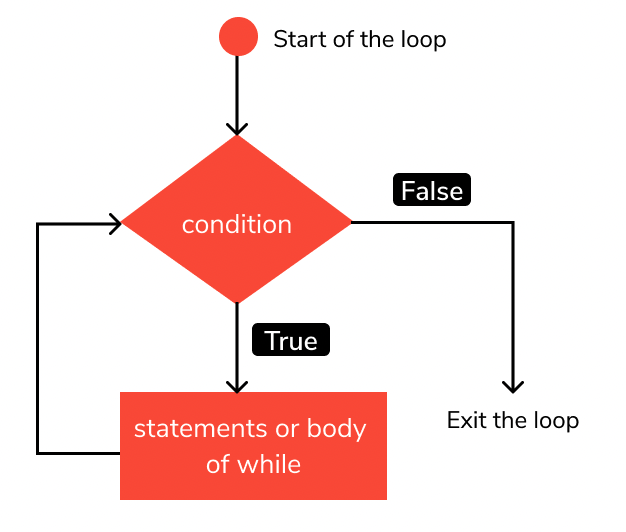
Ada pun jika sebuah perulangan for dihentikan paksa dengan perintah break, maka perintah yang ada pada blok else tidak akan dieksekusi.

**Contoh Program Perulangan (For) Python**

Selain contoh-contoh yang telah kita coba di atas, teman-teman bisa melihat contoh-contoh lainnya di seri [Latihan Program Python](https://jagongoding.com/python/latihan-logika/overview). Di situ ada berbagai macam contoh program untuk melatih logika yang menggunakan perulangan for seperti:

* [Membuat Deret Fibonacci](https://jagongoding.com/python/latihan-logika/fibonacci-non-rekursif/)
* [Memeriksa Bilangan Ganjil Genap](https://jagongoding.com/python/latihan-logika/bilangan-ganjil-genap/)
* [Menentukan Bilangan Prima](https://jagongoding.com/python/latihan-logika/bilangan-prima/)
* [Menghitung Jumlah Huruf Vokal](https://jagongoding.com/python/latihan-logika/menghitung-huruf-vokal/)
* [Menghitung Pangkat Secara Manual](https://jagongoding.com/python/latihan-logika/perpangkatan/)
* [Prigrim Mingibih Hirif Vikil](https://jagongoding.com/python/latihan-logika/mengubah-huruf-vokal/)
* dan lain-lain

## Pengulangan dengan blok *while*



Perulangan while pada python adalah proses pengulangan suatu blok kode program selama sebuah kondisi terpenuhi. Singkatnya, perulangan while adalah perulangan yang bersifat *indefinite* alias tidak pasti, atau bahkan tidak terbatas.

Sebuah blok kode akan dilakukan terus-menerus selama suatu kondisi terpenuhi. Jika suatu kondisi ternyata tidak terpenuhi pada iterasi ke 10, maka perulangan akan berhenti. Jika kondisi yang sama pada saat yang berbeda ternyata berhenti pada iterasi ke 100, maka perulangan akan berhenti pada jumlah tersebut.

**Penulisan Sintaks While**

Kita bisa menulis sintaks while dengan cara berikut:

**while <**kondisi**>:**

*# blok kode yang akan diulang-ulang*

Terdapat 3 komponen utama:

* Yang pertama adalah keyword while, ini harus kita isi.
* Yang kedua adalah <kondisi>: ini bisa berupa variabel boolean atau ekspresi logika.
* Dan yang terakhir adalah blok (atau kumpulan baris) kode yang akan diulang-ulang kondisi terpenuhi.

Kita langsung pada contohnya saja.

Perulangan Tanpa Batas

Perulangan while sangat berkaitan dengan variabel boolean, atau *logical statement*. Karena penentuan kapan suatu blok kode akan diulang-ulang ditinjau dari True or False dari suatu pernyataan logika.

Sehingga jika suatu kondisi itu selalu benar, maka perulangannya pun akan selalu di eksekusi. Perhatikan contoh berikut:

while (1 + 2 == 3):

print**(**'Halo dunia!'**)**

Jika dieksekusi, sistem akan mencetak tulisan “Halo dunia!” berkali-kali tanpa henti.

…

Halo dunia!

Halo dunia!

Halo dunia!

Halo dunia!

...

Kita bisa memaksanya berhenti dengan menekan tombol Ctr + C jika menggunakan CLI, atau dengan cara menekan tombol stop jika menggunakan IDE atau sejenisnya.

**Kenapa perulangan di atas dieksekusi terus menerus?**

Karena kita memerintahkan komputer untuk menulis “Hello World” selama satu ditambah dua sama dengan tiga.

Pertanyaannya: apakah satu ditambah dua sama dengan tiga terus-menerus atau tidak? Jawabannya iya! Oleh karena itu sistem melakukan iterasi tak terbatas.

Contoh perulangan while seperti for + range

Nah, timbul pertanyaan.

Lalu bagaimana caranya agar kita bisa memberhentikan perulangan while?

Caranya gampang, kita buat kondisinya bersifat dinamis (alias bisa berubah-ubah).

Di dalam contoh berikut, kita akan menampilkan angka 1 sampai dengan angka 5 menggunakan perulangan while.

i **= 1**

while i <= 5:

print**(**i**)**

i **+= 1**

Kode program di atas akan menghasilkan output seperti berikut:

1

2

3

4

5

**Penjelasan**

Pada kode program di atas, kita menginstruksikan sistem untuk:

1. Melakukan perulangan selama variabel i kurang dari atau sama dengan 5.
2. Setiap kali iterasi, sistem akan menampilkan nilai dari i.
3. Dan yang terakhir, pada setiap iterasi, sistem akan menambahkan nilai i dengan angka 1.

Kalian bisa memodifikasi kode program di atas misalkan untuk:

* menampilkan bilangan prima dari 1 sampai 100
* menampilkan angka kelipatan 4 dari 1 sampai 100
* menampilkan angka ganjil dari 1 sampai 27
* dan sebagainya

Contoh perulangan while untuk list

Untuk menampilkan semua item pada list, cara yang paling clean adalah dengan menggunakan metode for seperti yang telah kita bahas sebelumnya.

Meskipun begitu, kita tetap bisa menggunakan perulangan while untuk bermain-main dengan list.

Perhatikan contoh berikut:

listKota **= [**

'Jakarta'**,** 'Surabaya'**,** 'Depok'**,** 'Bekasi'**,** 'Solo'**,** 'Jogjakarta'**,** 'Semarang'**,** 'Makassar'

]

*# bermain index*

i **= 0**

**while** i **<** len**(**listKota**):**

print**(**listKota**[**i**]**i **+= 1**

Jika dijalankan, hasilnya akan terlihat seperti ini:

Jakarta Surabaya Depok Bekasi Solo Jogjakara Semarang Makassar

Kita juga bisa menggunakan fungsi list.pop(). Perhatikan kode program berikut:

*# bermain pop*

**while** listKota**:** print**(**listKota**.**pop**(0))**

Kode program di atas juga akan menghasilkan output yang sama seperti yang kita lakukan dengan pendekatan indeks.

Contoh perulangan while dengan inputan

Kita juga bisa menggunakan while dengan inputan.

Perhatikan contoh di bawah. Pada contoh ini kita akan meminta user untuk memasukkan angka ganjil lebih dari 50. Jika user justru memasukkan nilai genap atau nilai yang kurang dari 50, maka sistem akan meminta user untuk menginputkan kembali.

a **=** int**(**input**(**'Masukkan bilangan ganjil lebih dari 50: '**))**

while a % 2 != 1 or a <= 50:

a **=** int**(**input**(**'Salah, masukkan lagi: '**))**

print**(**'Benar'**)**

Contoh Output:

Masukkan bilangan ganjil lebih dari 50: 1 Salah, masukkan lagi: 2

Salah, masukkan lagi: 3 Salah, masukkan lagi: 10 Salah, masukkan lagi: 50

Salah, masukkan lagi: 52

Salah, masukkan lagi: 54

Salah, masukkan lagi: 55

Benar

Contoh perulangan while dengan continue

Sama dengan perulangan for, kita juga bisa menggunakan perintah continue pada perulangan while.

Apa itu perintah continue?

Perintah continue berfungsi untuk men-skip iterasi sekarang ke iterasi selanjutnya. Contoh:

listKota **= [**

'Jakarta'**,** 'Surabaya'**,** 'Depok'**,** 'Bekasi'**,** 'Solo'**,** 'Jogjakarta'**,** 'Semarang'**,** 'Makassar'

]

*# skip jika i adalah bilangan genap # dan i lebih dari 0*

i **= -1**

**while** i **<** len**(**listKota**):** i **+= 1**

if i % 2 == 0 and i > 0:

print**(**'skip'**)**

continue

*# tidak dieksekusi ketika continue dipanggil*

print**(**listKota**[**i**])**

Output:

Jakarta Surabaya

skip <-- i sama dengan 2

Bekasi

skip <-- i sama dengan 4

Jogjakarta

skip <-- i sama dengan 6

Makassar

skip <-- i sama dengan 8

Pada output di atas, ketika i-nya adalah bilangan genap yang lebih dari satu, perintah

print(listKota[i]) tidak dieksekusi dan justru di-skip.

Contoh perulangan while dengan break

Kita juga bisa menggunakan perintah break pada perulangan while. Perintah break itu sebenarnya mirip dengan perintah continue.

Bedanya:

Ketika perintah break dipanggil, maka perulangan akan dihentikan secara paksa. Perhatikan contoh berikut:

listKota **= [**

'Jakarta'**,** 'Surabaya'**,** 'Depok'**,** 'Bekasi'**,** 'Solo'**,** 'Jogjakarta'**,** 'Semarang'**,** 'Makassar'

]

kotaYangDicari **=** input**(**'Masukkan nama kota yang dicari: '**)**

i **= 0**

**while** i **<** len**(**listKota**):**

**if** listKota**[**i**].**lower**() ==** kotaYangDicari**.**lower**():** print**(**'Ketemu di index'**,** i**)**

break

print**(**'Bukan'**,** listKota**[**i**])** i **+= 1**

Contoh output:

Masukkan nama kota yang dicari:  
bekasi  
Bukan Jakarta

Bukan Surabaya  
Bukan Depok  
Ketemu di index 3

while … else

Sama seperti for, kita juga bisa menggunakan blok kode else pada perulangan while.

Tugasnya pun sama: yaitu untuk mendefinisikan suatu tugas yang akan dieksekusi ketika perulangan telah selesai secara natural tanpa dihentikan secara paksa.

Kita coba ubah program pencarian kota di atas dengan menambahkan blok kode else seperti berikut:

listKota **= [**

'Jakarta'**,** 'Surabaya'**,** 'Depok'**,** 'Bekasi'**,** 'Solo'**,** 'Jogjakarta'**,** 'Semarang'**,** 'Makassar'

]

kotaYangDicari **=** input**(**'Masukkan nama kota yang dicari: '**)**

i **= 0**

**while** i **<** len**(**listKota**):**

**if** listKota**[**i**].**lower**() ==** kotaYangDicari**.**lower**():**

print**(**'Ketemu di index'**,** i**)**

break

print**(**'Bukan '**,** listKota**[**i**])** i **+= 1**

else:

print**(**'Maaf, kota yang anda cari tidak ditemukan.'**)**

Coba eksekusi lalu masukkan kota yang tidak ada pada variabel listKota. Berikut ini contoh output yang saya dapatkan:

Masukkan nama kota yang dicari:

sidoarjo  
Bukan Jakarta

Bukan Surabaya Bukan Depok Bukan Bekasi Bukan Solo

Bukan Jogjakarta Bukan Semarang Bukan Makassar

Maaf, kota yang anda cari tidak ditemukan.

Berbeda jika misal kota yang saya cari adalah kotak “Depok”:

Masukkan nama kota yang dicari:

depok   
Bukan Jakarta

Bukan Surabaya Ketemu di index 2

Di sini perintah yang ada di blok kode else tidak dieksekusi oleh sistem. Kenapa? Karena perulangannya diberhentikan secara paksa dengan perintah break, bukan karena berhenti secara natural.

Kapan harus menggunakan for, dan kapan harus menggunakan while?

Sekarang, mungkin masih ada satu pertanyaan lagi yang belum terjawab, yaitu:Kapan kita seharusnya menggunakan for? Dan kapan seharusnya menggunakan while?

Sebenarnya tidak ada acuan yang sangat baku, karena banyak sekali kasus-kasus yang bisa diselesaikan dengan menggunakan keduanya.

Tapi, kalau memang ingin sebuah jawaban:

* Kalian bisa menggunakan for untuk kasus-kasus yang berkaitan dengan data *sequence*

pada python, atau untuk kasus yang sudah jelas jumlah perulangannya berapa.

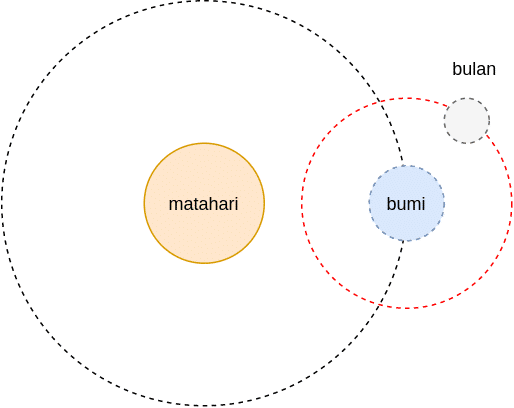
* Dan kalian bisa menggunakan while jika memang perulangannya tidak jelas akan dilakukan berapa banyak

## Pengulangan bertingkat

Apa itu Perulangan Bersarang/Bertingkat?

Perulangan bersarang atau perulangan bertingkat adalah sebuah perulangan yang berada atau terletak di dalam perulangan yang lain.

Contoh yang simpel adalah: hubungan antara bumi, bulan, dan matahari.



1. Bulan mengelilingi bumi
2. Bumi mengelilingi matahari
3. Akhirnya bulan juga ikutan mengelilingi matahari karena bumi pun mengelilinginya.

Atau, contoh yang lebih simpel adalah Putaran jarum jam. Ada tiga jarum:

* jarum detik
* jarum panjang / menit
* jarum pendek / jam Alurnya:

1. Dalam sehari, jarum jam akan berputar 360 derajat sebanyak 1 kali.
2. Setiap jam, jarum menit akan berputar 360 derajat sebanyak 1 kali. Sehingga jarum panjang akan berputar sebanyak 60 kali dalam sehari.
3. dan setiap menit, jarum detik akan berputar 360 derajat sebanyak 1 kali, sehingga dalam satu jam, ia akan berputar sebanyak 60 kali.

Intinya di dalam setiap satu putaran, di dalamnya masih ada putaran yang lainnya, dan ada putaran yang lainnya lagi.

Inilah yang dinamakan perulangan bersarang.

Oke, biar tidak makin bingung, mari kita seduh kopi dulu kemudian kita mulai contoh pengaplikasiannya dalam kode program python.

Alur Dasar

Silakan buka tools kalian untuk ngoding python (just info, di [seri tutorial](https://jagongoding.com/python/dasar/overview) ini saya menggunakan vscode –lebih tepatnya [vscodium](https://jagongoding.com/others/vscodium-vscode-tapi-100-open-source/)). Langkah selanjutnya silakan buat file python baru. Kemudian tulis dan perhatikan kode program berikut:

**for** i **in** range**(3):**

print**(**'Perulangan luar [i] = '**,** i**)**

**for** j **in** range**(5):**

print**(**' Perulangan dalam [i, j] = '**,** str**(**i**) +** ', ' **+** str**(**j**))**

Jika dijalankan, kita akan mendapatkan output:

| Perulangan luar[i] | = | 0 |  | | | |
| --- | --- | --- | --- | --- | --- | --- |
| Perulangan dalam | [i, |  | j] | = | 0, | 0 |
| Perulangan dalam | [i, |  | j] | = | 0, | 1 |
| Perulangan dalam | [i, |  | j] | = | 0, | 2 |
| Perulangan dalam | [i, |  | j] | = | 0, | 3 |
| Perulangan dalam | [i, |  | j] | = | 0, | 4 |
| Perulangan luar[i] | = | 1 |  |  |  |  |
| Perulangan dalam | [i, |  | j] | = | 1, | 0 |
| Perulangan dalam | [i, |  | j] | = | 1, | 1 |
| Perulangan dalam | [i, |  | j] | = | 1, | 2 |
| Perulangan dalam | [i, |  | j] | = | 1, | 3 |
| Perulangan dalam | [i, |  | j] | = | 1, | 4 |

| Perulangan luar[i] | = | 2 |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Perulangan dalam | [i, |  | j] | = | 2, | 0 |
| Perulangan dalam | [i, |  | j] | = | 2, | 1 |
| Perulangan dalam | [i, |  | j] | = | 2, | 2 |
| Perulangan dalam | [i, |  | j] | = | 2, | 3 |
| Perulangan dalam | [i, |  | j] | = | 2, | 4 |

Di dalam kode program kita di atas, kita telah membuat 2 buah perulangan:

1. Perulangan pertama sebanyak 3 kali.
2. Dan perulangan kedua sebanyak 5 kali setiap satu kali perulangan pertama.

Sehingga, total iterasi seluruhnya adalah 3x5 yaitu 15. Contoh lain:

**for** baris **in** range**(5):**

**for** kolom **in** range**(7):**

print**(**'o'**,** end **=** ' '**)**

**else:**

print**(**''**)**

Output:

o o o o o o o

o o o o o o o

o o o o o o o

o o o o o o o

o o o o o o o

Menggunakan while

Kita juga bisa menggunakan while untuk membangun sebuah perulangan bersarang, perhatikan contoh berikut:

listKota **= [**

'Jakarta'**,** 'Surabaya'**,** 'Makassar'

]

**for** kota **in** listKota**:** print**(**kota**)**

listTempat **= [**

'Taman'**,** 'Lapangan'**,** 'Mall'

]

w**hile** listTempat**:**

print**(**'-->'**,** listTempat**.**pop**(0))**

Jika dijalankan, program di atas akan menghasilkan output sebagai berikut:

Jakarta

--> Taman

--> Lapangan

--> Mall Surabaya

--> Taman

--> Lapangan

--> Mall Makassar

--> Taman

--> Lapangan

--> Mall

Kita juga bisa menambahkan perulangan lebih dari 2 tingkat, 3 tingkat, atau lebih sesuai kebutuhan.

Contoh tambahan perulangan bersarang menggunakan while:

max\_baris **= 7**

max\_kolom **= 7**

baris **= 0**

**while** baris **<** max\_baris**:** kolom **= 0**

**while** kolom **<** max\_kolom**:**

print**(**'o' **if** kolom **<=** baris **else** '+'**,** end **=** ' '**)** kolom **+= 1**

else:

print**(**''**)** baris **+= 1**

Output:

o + + + + + +

o o + + + + +

o o o + + + +

o o o o + + +

o o o o o + +

o o o o o o +

o o o o o o o

Bonus Contoh Program

Program di bawah ini mengilustrasikan bagaimana cara mengubah huruf vokal pada suatu kata dengan memanfaatkan perulangan bersarang sebagai struktur kontrolnya.

Silakan coba dan perhatikan dengan seksama meskipun ada beberapa kode program yang mungkin belum kalian pahami.

**import** re

listKota **= [**

'Solo'**,** 'Surabaya'**,** 'Bekasi'**,** 'Jakarta'

]

listHurufVokal **= [**

'a'**,** 'i'**,** 'u'**,** 'e'**,** 'o'

]

**for** kota **in** listKota**:** print**(**'[' **+** kota **+** ']'**)**

**for** vokal **in** listHurufVokal**:**

print**(**'-->'**,** re**.**sub**(**'[aiueo]'**,** vokal**,** kota**))**

Jika dijalankan, output program di atas terlihat seperti ini:

[Solo]

--> Sala

--> Sili

--> Sulu

--> Sele

--> Solo [Surabaya]

--> Sarabaya

--> Siribiyi

--> Surubuyu

--> Serebeye

--> Soroboyo [Bekasi]

--> Bakasa

-—> Bikisi

--> Bukusu

--> Bekese

--> Bokoso [Jakarta]

--> Jakarta

--> Jikirti

--> Jukurtu

--> Jekerte

--> Jokorto

# 

# Fungsi (*def*)

## Pengenalan Fungsi

Fungsi pada python adalah kumpulan perintah atau baris kode yang dikelompokkan menjadi satu kesatuan untuk kemudian bisa dipanggil atau digunakan berkali-kali. Sebuah fungsi bisa menerima parameter, bisa mengembalikan suatu nilai, dan bisa dipanggil berkali-kali secara independen. Dengan fungsi kita bisa memecah program besar yang kita tulis, menjadi bagian-bagian kecil dengan tugasnya masing-masing. Juga, fungsi akan membuat kode program kita menjadi lebih “*reusable*” dan lebih terstruktur.

**Sintaks Fungsi**

Di dalam python, sintaks pembuatan fungsi terlihat seperti berikut:

**def <**nama\_fungsi**>(**parameters**):**

statements

Sintaks di atas secara umum terbagi menjadi 4 bagian:

1. Kata kunci def yang menjadi pertanda bahwa blok kode program adalah sebuah fungsi
2. Nama fungsi yang kita buat
3. Parameters yang akan diterima oleh fungsi yang kita buat (tidak wajib)
4. Dan blok kode fungsi yang di sana akan kita tulis perintah-perintah yang harus dilakukan oleh sebuah fungsi

Oiya: jangan lupa bahwa blok kode program di dalam python didefinisikan dengan indentasi. Silakan baca [aturan sintaks python](https://jagongoding.com/python/dasar/aturan-sintaks-python) untuk lebih lengkapnya.

Contoh sebuah fungsi sederhana dengan nama halo\_dunia():

**def** halo\_dunia**():**

print**(**'Halo python! Halo dunia!'**)**

Fungsi di atas, jika dipanggil, akan mengeksekusi perintah print() yang ada di dalamnya.

**Memanggil Fungsi**

Bagaimana cara memanggil fungsi yang telah kita definisikan? Sederhana.

Cukup ketik nama fungsinya, ditambah dengan tanda kurung () seperti berikut:

halo\_dunia**()**

Output:

Halo python! Halo dunia

Bahkan kita bisa memanggil fungsi halo\_dunia() berkali-kali:

halo\_dunia**()**

halo\_dunia**()**

halo\_dunia**()**

Output:

Halo python! Halo dunia

Halo python! Halo dunia

Halo python! Halo dunia

## Fungsi dengan *arguments* dan *parameters*

Sebuah fungsi juga bisa menerima parameter atau pun argumen. Ia merupakan suatu nilai/variabel yang dilemparkan ke dalam fungsi untuk diproses lebih lanjut.

Sebagai contoh, perhatikan output berikut:

Halo Nurul, selamat datang! Halo Lendis, selamat datang! Halo Fabri, selamat datang! Halo Isa, selamat datang!

Lalu, bagaimana cara kita memproduksi output seperti itu dengan python? Ada banyak cara. Bisa dengan list, perulangan, dan lain sebagainya.

Akan tetapi, mungkin yang langsung terbesit dalam benak kita adalah dengan melakukan 4x print() seperti ini:

print**(**'Halo Nurul, selamat datang!'**)**

print**(**'Halo Lendis, selamat datang!'**)**

print**(**'Halo Fabri, selamat datang!'**)**

print**(**'Halo Isa, selamat datang!'**)**

Itu adalah cara yang sangat simpel, dan juga tidak salah. Akan tetapi, dari pada kita melakukan 4x print seperti di atas, kita bisa memanfaatkan fungsi dan parameter pada python.

Sehingga kode programnya akan terlihat seperti ini:

**def** selamat\_datang **(**nama**):**

print**(**f'Halo {nama}, selamat datang!'**)**

selamat\_datang**(**'Nurul'**)** selamat\_datang**(**'Lendis'**)** selamat\_datang**(**'Fabri'**)** selamat\_datang**(**'isa'**)**

Dan kita tetap akan mendapatkan output yang sama. Lebih elegan bukan 😎

**Parameter Wajib**

Parameter di dalam python bisa lebih dari satu, bisa wajib semua (harus diisi), dan bisa juga bersifat opsional.

Perhatikan contoh fungsi berikut:

**def** perkenalan **(**nama**,** asal**):**

print**(**f"Perkenalkan saya {nama} dari {asli}"**)**

Jika dipanggil:

perkenalan**(**"Renza Ilhami"**,** "Jawa Timur"**)**

Kita akan mendapatkan output:

Perkenalkan saya Renza Ilhami dari Jawa Timur

Tapi jika kita memanggilnya dengan parameter tidak lengkap, justru kita akan mendapatkan error:

perkenalan**(**"Renza Ilhami"**)**

Error:

**Exception** has occurred**: TypeError**

perkenalan**()** missing **1** required positional argument**:** 'asal'

Kenapa? Karena kita hanya memasukkan satu parameter saja padahal parameter yang diminta ada 2.

**Parameter Opsional (atau Default)**

Tidak semua parameter fungsi pada python itu bersifat wajib. Ada yang opsional.

Parameter opsional adalah parameter yang seandainya tidak diisi, dia sudah memiliki nilai default.

Perhatikan contoh berikut:

**def** suhu\_udara**(**daerah**,** derajat**,** satuan **=** 'celcius'**):**

print**(**f"Suhu di {daerah} adalah {derajat} {satuan}"**)**

Pada fungsi suhu\_udara() di atas, kita mendefinisikan 3 buah parameter:

* daerah
* derajat
* suhu = 'celcius'

Dua parameter pertama adalah bersifat wajib dan harus diisi, sedangkan parameter ketiga tidak wajib. Jika tidak kita isi, maka nilai default-nya adalah “celcius”.

Sekarang, kita coba panggil fungsi tersebut dengan 2 cara:

suhu\_udara**(**"Surabaya"**, 30)**

suhu\_udara**(**"Surabaya"**, 86,** 'Fahrenheit'**)**

Jika dijalankan, outputnya akan terlihat seperti ini:

Suhu di Surabaya adalah 30 celcius Suhu di Surabaya adalah 86 Fahrenheit

**Fungsi Dengan Parameter Tidak Berurut**

Jika kita perhatikan lagi fungsi suhu\_udara(), kita akan dapati kalau parameter yang bersifat opsional hanya ada 1, dan hanya ada di belakang.

Tapi, bagaimana jika ternyata parameter opsionalnya ada lebih dari 1? Coba perhatikan:

**def** suhu\_udara **(**daerah**,** derajat **= 30,** satuan **=** 'celcius'**):** print**(**f"Suhu di {daerah} adalah {derajat} {satuan}"**)**

Pada fungsi tersebut, kita telah mengatur nilai default untuk parameter derajat. Sehingga sekarang kita memiliki dua buah parameter.

Kita coba panggil dengan 2 parameter seperti ini:

suhu\_udara**(**'Jakarta'**,** 'fahrenheit'**)**

Apa outputnya?

Suhu di Jakarta adalah fahrenheit celcius

Hmmm. Kok gitu?

Padahal kita inginnya hanya mengisi 2 parameter saja:

1. Satu untuk parameter daerah
2. Dan yang kedua untuk parameter satuan

Tapi di sini malah terisi adalah parameter derajat.

Untuk mengatasi hal ini, kita bisa mendefinisikan nama argumen/parameter yang akan kita isi. Perhatikan contoh ini:

suhu\_udara**(**'Jakarta'**,** 'fahrenheit'**)** suhu\_udara**(**'Jakarta'**,** satuan**=**'fahrenheit'**)**

Pemanggilan fungsi di atas akan menghasilkan output:

Suhu di Jakarta adalah fahrenheit celcius Suhu di Jakarta adalah 30 fahrenheit

Dengan mendefinisikan nama argumen yang kita *passing*, kita sekarang bisa memanggilnya bahkan secara tidak berurutan sekalipun:

suhu\_udara**(**satuan**=**'kelvin'**,** daerah**=**'Makasar'**,** derajat**=100)**

Output:

Suhu di Makasar adalah 100 kelvin

## Fungsi dengan *return types*

Jenis fungsi yang berikutnya adalah berkaitan dengan nilai kembalian. Ditinjau dari segi pengembalian nilai, fungsi terbagi menjadi 2:

1. Fungsi yang tidak mengembalikan nilai
2. Fungsi yang mengembalikan nilai

Pada contoh-contoh di atas, kita telah membuat dan memanggil fungsi-fungsi yang tidak memiliki nilai. Sekarang, kita akan coba membuat fungsi yang mempunyai atau mengembalikan sebuah nilai.

**def** luas\_persegi **(**sisi**):**

**return** sisi **\*** sisi

Penjelasan

* Kata kunci return berfungsi untuk mengembalikan nilai.
* Nilai yang dikembalikan suatu fungsi, bisa kita olah kembali untuk berbagai kebutuhan.

Contoh:

*# tidak menghasilkan output apa pun*

luas\_persegi**(10)**

*# menghasilkan output*

print**(**'Luas persegi dengan sisi 4 adalah:'**,** luas\_persegi**(4))**

*# kita juga bisa simpan di dalam variabel*

persegi\_besar **=** luas\_persegi**(100)**

persegi\_kecil **=** luas\_persegi**(50)**

print**(**'Toal luas persegi besar dan kecil adalah:'**,** persegi\_besar **+** persegi\_kecil**)**

Jika dijalankan, kita akan mendapatkan output:

Luas persegi dengan sisi **4** adalah: **16**

Toal luas persegi besar dan kecil adalah: **12500**

Jadi intinya: fungsi yang mengembalikan nilai adalah sebuah fungsi yang jika kita panggil, dia akan memberikan kita sebuah nilai yang bisa kita olah lebih lanjut, seperti misalkan kita simpan dalam sebuah variabel atau kita lakukan operasi tertentu.

Lebih dari 1 return

Jika statement return telah dieksekusi pada sebuah fungsi, maka semua proses yang ada di dalam blok kode fungsi tersebut akan berhenti.

Sehingga, misalkan kita memiliki lebih dari 1 buah return, maka hanya ada satu return saja yang dieksekusi. Dan ketika sebuah return telah dieksekusi, semua perintah yang ada di bawahnya akan di-skip –*ini mirip dengan perintah break pada* [*perulangan for*](https://jagongoding.com/python/dasar/perulangan-for) *mau pun* [*while*](https://jagongoding.com/python/dasar/perulangan-while).

Perhatikan contoh berikut:

**def** persentase **(**total**,** jumlah**):**

**if (**total **>= 0 and** total **<=** jumlah**):**

**return** total **/** jumlah **\* 100**

return False

*# output 50*

print**(**persentase**(30, 60))**

*# output False*

print**(**persentase**(100, 60))**

Output:

50.0

False

Ruang Lingkup (dan Siklus Hidup) Variabel Pada Fungsi

Variabel memiliki ruang lingkup dan siklus hidup.

Secara umum, terdapat dua ruang lingkup variabel pada python:

1. Variabel global
2. Dan variabel lokal

Variable global adalah variabel yang bisa dipanggil dari manapun dari satu file python.

Sedangkan variable lokal adalah variabel yang hanya hidup di dalam satu blok kode tertentu (seperti di dalam fungsi, seperti kasus kita pada pertemuan ini).

Perhatikan contoh berikut:

kota **=** 'Lamongan'

**def** halo**() :**

print**(**kota**)**

print**(**'[print secara langsung]'**,** kota**)** print**(**'[panggil fungsi halo]'**,** end**=**' '**)**

halo**()**

Output:

[print secara langsung] Lamongan [panggil fungsi halo] Lamongan

Pada kode di atas, variabel kota yang ada di dalam fungsi, adalah variabel kota yang sama dengan yang ada di luar fungsi.

Tapi, coba kita ubah kode programnya:

kota**,** provinsi **=** 'Lamongan'**,** 'Jawa Timur'

**def** hello **():**

provinsi **=** 'Jawa Barat' print**(**kota**,** provinsi**)**

print**(**'[PANGGIL FUNGSI hello()]'**)** hello**()**

print**(**'\n[SECARA LANGSUNG]'**)**print**(**kota**,** provinsi**)**

Output:

[PANGGIL FUNGSI hello()]

Lamongan Jawa Barat

[SECARA LANGSUNG]

Lamongan Jawa Timur

Pencerahan

1. Kita membuat 2 buah variabel dengan nama kota dan provinsi
2. Kita *mengubah* variabel provinsi pada fungsi hello()
3. Saat fungsi hello() dieksekusi, benar saja variabel provinsi berubah.
4. Tapi, ketika kita tampilkan lagi (secara langsung) variabel provinsi, nilainya kembali ke nilai semula.

Kenapa?

Karena ketika kita mengubah variabel provinsi pada fungsi hello(), itu sebenarnya kita tidak merubah variabel provinsi yang ada di luar, melainkan kita membuat variabel baru dengan nama yang sama, akan tetapi dengan ruang lingkup lokal yang hanya bisa diakses pada fungsi hello() saja.

Docstring

Yang terakhir, kita bisa memberikan sebuah deskripsi terhadap fungsi yang kita buat.

Deskripsi ini biasanya akan ditampilkan oleh Text Editor mau pun IDE sebagai bantuan tentang apa yang sebenarnya dilakukan oleh sebuah fungsi.

Caranya gampang.

Untuk mendefinisikan deskripsi program, kita hanya perlu menuliskan [komentar multi baris](https://jagongoding.com/python/dasar/aturan-sintaks-python/#penulisan-komentar) tepat setelah mendefinsikan nama fungsi.

Perhatikan contoh berikut:

**def** suhu\_udara **(**daerah**,** derajat **= 30,** satuan **=** 'celcius'**):**

"""

Fungsi ini bertugas untuk menampilkan teks yang memberikan informasi

tentang suhu udara di suatu daerah. """

print**(**"Suhu di {} adalah {} {}"**.**format**(**daerah**,** derajat**,** satuan**))**

Berikut ini contoh tampilannya jika kita menggunakan Visual Studio Code.

## 

## Fungsi *recursive*

Dari segi tempat pemanggilan, fungsi pada python bisa dipanggil dari berbagai tempat. Bisa dari console. Bisa dari fungsi yang lainnya. Bisa juga dari dirinya sendiri.

Fungsi yang memanggil dirinya sendiri, akan menciptakan sebuah perulangan. Dan perulangan ini biasa disebut sebagai [perulangan rekursif](https://jagongoding.com/python/dasar/perulangan-rekursif).

Di dalam dunia pemrograman, fungsi rekursif merupakan sebuah metode perulangan yang bersifat non-iterasi.

Sebenarnya fungsi rekursif hanyalah sebuah fungsi biasa seperti [fungsi def](https://jagongoding.com/python/dasar/fungsi) pada umumnya. Dia bisa dipanggil, bisa menerima parameter, bisa mengembalikan nilai, dan lain sebagainya.

Hanya saja, sesuai namanya, fungsi rekursif itu bersifat rekursi. Kenapa?

Karena ia memanggil dirinya sendiri sehingga menimbulkan 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.

Untuk membuat fungsi rekursif, caranya sama saja dengan [fungsi biasa](https://jagongoding.com/python/dasar/fungsi).

Yang membuat sebuah fungsi menjadi rekursif adalah karena ia memanggil dirinya sendiri. Itu saja.

Jadi contoh paling sederhanya bisa terlihat seperti berikut:

**def** halo\_dunia**():**

print**(**'Halo dunia!'**)**

*# panggil dirinya sendiri*

halo\_dunia**()** *# <-- rekursifitas*

*# memanggil fungsi helo\_dunia untuk pertama kali*

halo\_dunia**()**

Error yang saya dapat:

...

Halo dunia! Halo dunia! Halo dunia!

Fatal Python error**:** \_Py\_CheckRecursiveCall**:** Cannot recover

**from** stack overflow**.**

Python runtime state**:** initialized

Kenapa Error?

Karena perulangan rekursif sudah dipanggil terlalu banyak dan tidak ada tanda-tanda akan berhenti. Oleh karena itu sistem langsung memberhentikannya secara paksa.



Menampilkan Angka 1 Sampai 10

Untuk lebih memahami bagaimana cara fungsi rekursif bekerja, mari kita buat contoh sederhana yaitu dengan menampilkan angka 1 sampai 10.

Kalau kita membuatnya dengan [perulangan for](https://jagongoding.com/python/dasar/perulangan-for), kode programnya akan terlihat sesimpel ini:

**for** i **in** range**(10):**

print**(**i**)**

Nah, pertanyaannya adalah: bagaimana cara mengubah perulangan di atas menjadi perulangan rekursif?

Ikuti langkah-langkah berikut:

Step 1: Bikin dasarnya dulu

Untuk ancang-ancang dan mempermudah pemahaman, kita bikin dulu seperti ini:

**def** tampilkanAngka **(**i**):**

print**(**f'Perulangan ke {i}'**)**

*panggil beberapa kali* tampilkanAngka**(1)** tampilkanAngka**(2)** tampilkanAngka**(3)**

Output:

Perulangan ke 1 Perulangan ke 2 Perulangan ke 3

Sampai sini sangat jelas. Fungsi tampilanAngka() kita panggil 3 kali dengan parameter i

yang berbeda-beda.

Step 2: Tentukan batasnya

Sekarang, ubah parameter fungsi tampilkanAngka menjadi 2 parameter:

* batas - Sebagai batas dari perulangan. Ini bersifat wajib diisi.
* i - Ini sebagai penanda iterasi keberapa. Kita jadikan ini parameter ke-2 dan bersifat opsional.

**def** tampilkanAngka **(**batas**,** i **= 1):**

print**(**f'Perulangan ke {i}'**)**

*# Panggil beberapa kali untuk mensimulasikan # cara kerja*

tampilkanAngka**(3)** tampilkanAngka**(3, 2)**

tampilkanAngka**(3, 3)**

Jika dijalankan, kita akan mendapatkan output:

Perulangan ke **1** Perulangan ke **2** Perulangan ke **3**

Step 3: Rekursifitas! Panggil diri sendiri.

Nah, dari kode program terakhir, kita jadi tahu bagaimana harusnya fungsi tampilanAngka()

dipanggil.

Jika kita ingin menampilkan 10x perulangan, itu artinya:

1. Parameter batas akan selalu bernilai 10
2. Sedangkan parameter i akan bernilai 1 sampai dengan 10.
3. Lalu ketika variabel i sudah sama dengan nilai variabel batas, maka perulangan dihentikan alias rekursifitas tidak perlu dilakukan lagi.

Kita terapkan dalam kode program:

**def** tampilkanAngka **(**batas**,** i **= 1):**

print**(**f'Perulangan ke {i}'**)**

**if (**i **<** batas**):**

*# di sini lah rekursifitas itu terjadi*

tampilkanAngka**(**batas**,** i **+ 1)**

*# memanggil fungsi tampilkanAngka # untuk pertama kali* tampilkanAngka**(10)**

Output: Perulangan ke 1 Perulangan ke 2 Perulangan ke 3

...

Perulangan ke 8

perulangan ke 9

Perulangan ke 10

Perhatikan Alur Perjalanan Program

Oke. Sekarang kita akan bermain-main dengan alur program rekursif.

Kita akan coba balik perintah print() dan perintah rekursif. Yang awalnya print() terlebih dulu kemudian proses rekursif, sekarang kita ubah menjadi proses rekursif dahulu baru setelah itu proses print().

Perhatikan kode program di bawah:

**def** tampilkanAngka **(**batas**,** i **= 1):**

**if (**i **<** batas**):**

*# di sini lah rekursifitas itu terjadi*

tampilkanAngka**(**batas**,** i **+ 1)**

print**(**f'Perulangan ke {i}'**)**

*# memanggil fungsi tampilanAngka # untuk pertama kali* tampilkanAngka**(10)**

Coba jalankan, maka kita akan mendapatkan output:

Perulangan ke 10 Perulangan ke 9 Perulangan ke 8

...

Perulangan ke 3 Perulangan ke 2 Perulangan ke 1

Loh, kok jadi kebalik? 😁 Jadi, alurnya gini:

1. Kita manggil fungsi tampilanAngka(10).
2. Lalu fungsi tersebut memanggil dirinya sendiri sebelum melakukan print()
3. Dia hanya akan melakukan print() ketika proses rekursif selesai, alias setelah pemanggilan fungsi tampilanAngka() yang kedua selesai.
4. Tapi, ternyata ketika memanggil yang kedua kalinya, dia langsung memanggil dirinya sendiri untuk yang ketiga kalinya. Dia hanya akan melakukan perintah print() ketika proses pemanggilan yang ketiga selesai.
5. Begitu lah seterusnya hingga proses rekursif berakhir, sehingga perintah print() yang pertama kali dilakukan adalah yang ketiga.

Masih Agak Bingung?

Jadi, alur dari kode di atas itu kalau kita bedah, kira-kira seperti ini:

i **= 1**

batas **= 3**

**if** i **<** batas**:**

iDua **=** i **+ 1**

**if** iDua **<** batas**:**

iTiga **=** iDua **+ 1**

**if** iTiga **<** batas**:**

iEmpat **=** iTiga **+ 1**

**if** iEmpat **<** batas**:**

iLima **=** iEmpat **+ 1**

*# dan seterusnya*

print**(**iLima**)**

print**(**iEmpat**)**

print**(**iTiga**)**

print**(**iDua**)**

print**(**i**)**

Coba teman-teman edit variabel batas menjadi 4, 5, atau 2. Lalu jalankan kodenya.

Masih Tetep Bingung?

Oke. Untuk lebih memperjelas, mari kita ubah kodingannya menjadi lebih ruwet ya 😁

Kita akan melakukan print() sebanyak dua kali: yaitu sebelum dan sesudah pemanggilan fungsi rekursif.

Perhatikan kode berikut:

**def** tampilkanAngka **(**batas**,** i **= 1):**

prefix **=** '--' **\* (**i **- 1)**

print**(**f'{prefix} Sebelum rekursif ({i})'**)**

**if (**i **<** batas**):**

*# di sini lah rekursifitas itu terjadi*

tampilkanAngka**(**batas**,** i **+ 1)**

print**(**f'{prefix} Setelah rekursif ({i})'**)**

*# memanggil fungsi tampilkanAngka # untuk pertama kali* tampilkanAngka**(5)**

Jika dijalankan, hasilnya seperti ini:

Sebelum rekursif (1)

--Sebelum rekursif (2)

----Sebelum rekursif (3)

------Sebelum rekursif (4)

--------Sebelum rekursif (5)

--------Setelah rekursif (5)

------Setelah rekursif (4)

----Setelah rekursif (3)

--Setelah rekursif (2)

Setelah rekursif (1)

Penjelasan

Kita bisa simpulkan bahwa:

Fungsi yang pertama kali dipanggil, adalah fungsi yang terakhir kali selesai. Dan fungsi yang terakhir kali dipanggil, ia adalah fungsi yang paling pertama selesai.

Tidak terasa, ternyata hanya untuk menjelaskan perulangan i sampai x saja lumayan panjang.

4 Contoh Program Rekursif Python:

Untuk lebih memperdalam lagi bagaimana tentang penerapan fungsi rekursif pada python, saya ada beberapa [contoh kasus](https://jagongoding.com/python/latihan-logika/overview) yang menggunakan fungsi reskursif sebagai solusinya:

* [Membuat Deret Fibonacci Dengan Perulangan Rekursif](https://jagongoding.com/python/latihan-logika/fibonacci-dengan-rekursif/)
* [4 Cara Menghitung Pangkat di Python (Salah Satunya Rekursif)](https://jagongoding.com/python/latihan-logika/perpangkatan/)
* [3 Cara Menghitung Faktorial di Python (Salah Satunya Rekursif)](https://jagongoding.com/python/latihan-logika/faktorial/)
* [2 Cara Manual Menghitung Nilai Maksimum dan Minimum di Python (Salah Satunya](https://jagongoding.com/python/latihan-logika/mencari-nilai-maksimum-dan-minimum/) [Rekursif)](https://jagongoding.com/python/latihan-logika/mencari-nilai-maksimum-dan-minimum/)