## Pengantar Mengatur Alur Program

## [Ada 10 anak tangga yang diibaratkan modul kelas. Bima sedang menaiki anak tangga ke-6.](https://www.dicoding.com/academies/620/tutorials/34988?from=34978)

Dalam materi sebelumnya, Anda sudah memahami cara berpikir prosedural. Materi yang membawa kita untuk menganalisis dan memahami algoritma program secara sekuensial, mengamati aksi, serta mendalami maksud dari blok program.

Pada materi ini, kita akan mengenal control flow secara mendalam dengan perspektif pemrograman prosedural. Untuk itu, Anda perlu belajar beberapa materi mengenai konsep berikut:

* pengondisian if-else dalam melakukan percabangan alur program;
* pengulangan untuk mengulang sebuah blok program;
* skema pemrosesan sekuensial;
* hubungan berulang pada setiap iterasi pengulangan; dan
* penerapan tingkat lanjut dalam melakukan pengondisian dan pengulangan.

Sebagai permulaan, kita akan senam otak terlebih dahulu. Mengapa kita harus senam otak sebelum memulai pembelajaran? Salah satu alasan kita melakukan ini adalah untuk merangsang fungsi otak dalam meningkatkan kemampuan berpikir dan belajar. Tidak hanya menambah ilmu yang akan dipelajari, tetapi melatih memori dan memecahkan suatu masalah dalam aktivitas sehari-hari. Sudah siap? Let's read the story down below!

Bima sedang mengikuti pelajaran di sekolah. Dia mendengarkan materi yang diajarkan oleh guru dengan saksama. Suatu ketika guru sedang mengajukan pertanyaan pada seluruh murid kelas. Beliau memberikan instruksi sebelum pertanyaan dilontarkan.

1. Jika ada siswa yang bisa menjawab pertanyaan dengan benar, guru akan memberi nilai tambah dan melanjutkan materi berikutnya.
2. Jika siswa tersebut menjawab salah, guru akan memberikan penjelasan tambahan terkait pertanyaan tersebut.

Dua instruksi tersebut dipahami oleh seluruh murid. Kemudian, guru melayangkan satu pertanyaan saja kepada seluruh murid kelas. Bima mengacungkan tangan dan berhasil menjawab pertanyaan dengan benar karena ia rajin belajar. Alhasil, guru memberikan nilai tambahan dan melanjutkan materi.

[](https://www.dicoding.com/academies/620/tutorials/34988?from=34978)

Melihat cerita di atas, bisakah Anda menarik kesimpulan apa yang terjadi jika ada siswa yang menjawab salah? Yup, seluruh siswa akan mendapatkan pelajaran tambahan terkait pertanyaan yang diajukan oleh guru.

Saat ini, Anda melihat ada dua keluaran yang akan terjadi berdasarkan cerita tersebut. Tidak ada efek neto negatif yang dihasilkan dari interaksi siswa dan guru. Kedua keluaran tersebut adalah ekspektasi guru saat menerima jawaban dari murid. Guru berekspektasi bahwa seluruh jawaban akan membawa siswa lebih mengerti dan memahami materi yang telah disampaikan.

Hal yang sama dapat terjadi dalam sebuah pemrograman. Kita dapat memberikan keluaran  pada suatu program berjumlah lebih dari satu dengan memanfaatkan pengondisian atau rule yang telah ditetapkan. Dengan begitu, kita dapat memberikan banyak opsi keluaran yang kita inginkan tanpa menghasilkan efek neto negatif.

Nah, pengondisian adalah salah satu cara untuk mengendalikan aliran program. Dengan mengatur jalannya algoritma program, kita dapat mempertimbangkan keluaran yang dihasilkan berdasarkan input yang diberikan. Program akan mengalir mengikuti kondisi atau ketetapan yang sudah Anda buat demi menghasilkan efek neto positif.

Apakah sudah mulai terbayang hal yang akan Anda pelajari di materi ini? Saat ini hingga mendatang, Anda akan belajar cara mengatur, mengendalikan, dan mempertimbangkan aliran program berdasarkan algoritma yang dibuat. Apakah Anda siap untuk memulai materi ini? Tanpa perlu menunggu lebih lama, mari kita langsung beralih ke materi berikutnya.

## Pengondisian

Dalam kehidupan sehari-hari, kita dihadapkan dengan pilihan dalam menyelesaikan suatu kegiatan. Anda beranjak dari tempat tidur hingga belajar materi ini adalah rangkaian tindakan yang dipilih dari sekian opsi yang tersedia. Setiap tindakan yang dikerjakan adalah hasil dari pemilihan opsi meskipun terkadang tidak kita sadari.

Dengan adanya opsi, artinya, kita menerima suatu pernyataan yang perlu dievaluasi. Proses ini memungkinkan kita untuk membuat keputusan terbaik berdasarkan keadaan dan tujuan yang ingin dicapai. Memperkirakan output menjadi hal yang dapat membantu kita mempersiapkan strategi dalam mencapai tujuan.

[](https://www.dicoding.com/academies/620/tutorials/34993?from=34988)

Dalam pemrograman, ada konsep serupa untuk memilih suatu opsi berdasarkan pernyataan tertentu, yaitu pengondisian. Yup, suatu pernyataan ini dapat diartikan sebagai syarat atau kondisi yang harus terpenuhi agar suatu blok program bisa dieksekusi. Dengan pengondisian, program dapat mengevaluasi suatu kondisi dan mengambil tindakan berdasarkan evaluasi tersebut.

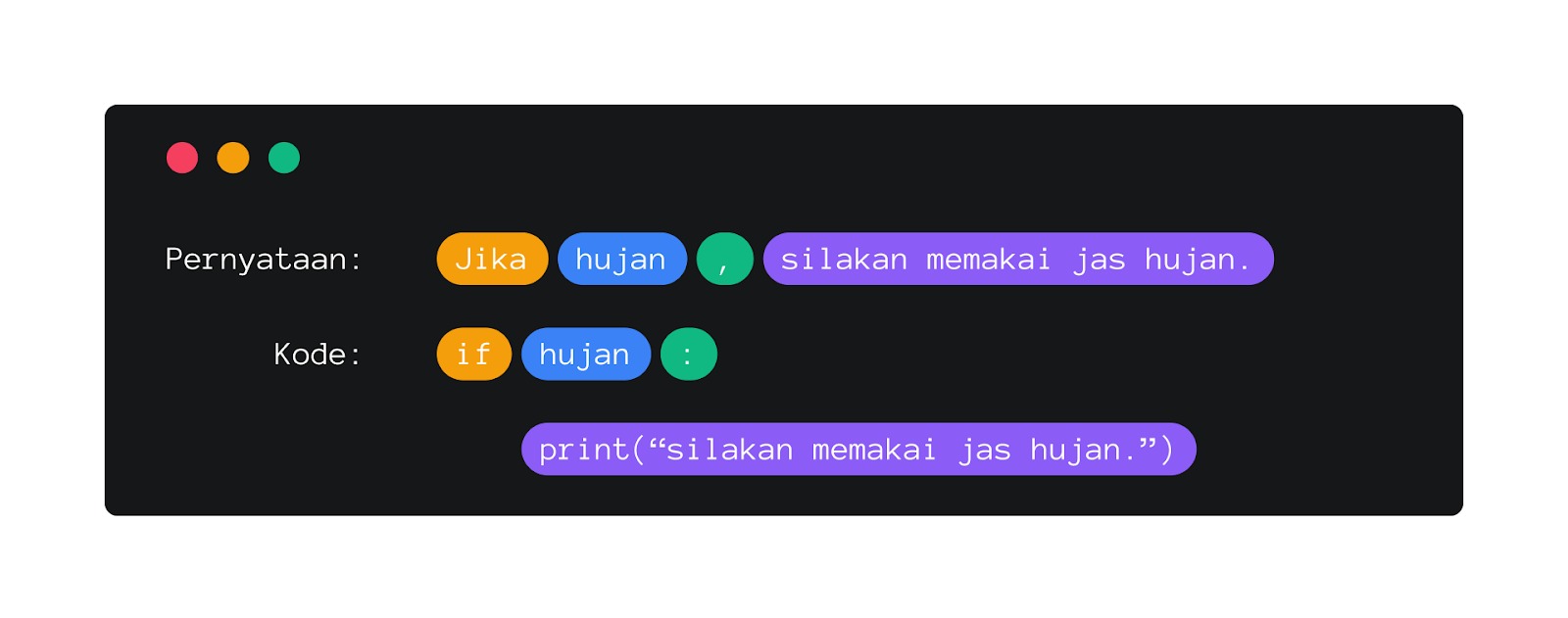
|  |
| --- |
| Saat Bima pergi ke sekolah, dia mempersiapkan buku, alat tulis, dan perlengkapan pendukung lainnya. Dia tak lupa membawa jas hujan untuk jaga-jaga. Jika terjadi hujan, Bima akan pakai jas hujan tersebut. Nyatanya, Bima mendapati cuaca hujan yang cukup deras ketika pulang sekolah. Dengan sigap, Bima mengeluarkan jas hujan dan memakainya hingga tiba di rumah.  [Bima sedang memakai jas hujan dan kehujanan saat pulang ke rumah.](https://www.dicoding.com/academies/620/tutorials/34993?from=34988)  Berdasarkan analogi di atas, ada kondisi bahwa “**Jika** terjadi hujan, Bima memakai jas hujan.” Pernyataan tersebut menggambarkan pengondisian yang didefinisikan oleh Bima. Saat kondisi tersebut terpenuhi, alur yang Bima lakukan pun berubah. Jika awalnya Bima tidak memakai jas hujan, ia jadi memakainya. |

Dalam proses mengevaluasi suatu kondisi, program membutuhkan ekspresi yang menghasilkan nilai boolean. Ada dua ekspresi yang mendukung hal tersebut, yaitu operator logika dan operator relasional. Operator aritmetika tidak menghasilkan nilai boolean sehingga tidak termasuk dalam proses evaluasi.

Mengapa hanya dua operator itu saja? Proses evaluasi ini semacam “yes/no question”, yakni program akan menjalankan blok program tertentu ketika kondisi bernilai **True**. Lalu, bagaimana jika ada kondisi yang bernilai False? Blok program yang sudah ditentukan sebelumnya akan diabaikan atau tidak dijalankan.

### Pernyataan if

Bahasa pemrograman Python memiliki pernyataan **if** untuk melakukan proses pengondisian. Cara penulisannya layaknya menulis pernyataan pengondisian pada kehidupan sehari-hari. Berikut adalah cara penulisan pengondisian if pada Python.

[](https://www.dicoding.com/academies/620/tutorials/34993?from=34988)

Di bawah ini adalah deskripsi mengenai blok ilustrasi di atas.

* Blok kuning adalah kata kunci if.
* Blok biru adalah kondisi yang harus terpenuhi.
* Blok hijau adalah karakter colon “:” sebagai pemisah pernyataan if dengan blok program.
* Blok ungu adalah blok program yang akan dijalankan apabila kondisi bernilai True.

Dalam menulis kode Python, terdapat blok program berwarna ungu yang ditulis menjorok ke dalam. Ingat, penulisan blok program memiliki indentasi yang sama. Pada pernyataan if, blok program warna ungu harus lebih menjorok ke dalam dan berbeda dengan blok program warna kuning. Ini harus dilakukan. Apabila dilanggar, program akan mengalami error.

Blok berwarna biru merupakan kondisi yang harus terpenuhi. Blok tersebut dapat Anda isi dengan operator logika atau relasional. Pastikan ekspresi yang digunakan menghasilkan nilai boolean.

Selain itu, terdapat karakter colon atau titik dua di akhir pernyataan if. Anda dapat menjumpainya pada blok berwarna hijau. Karakter tersebut harus ada di akhir pernyataan if agar compiler atau interpreter mampu memahami perintah tersebut. Ingat, bahasa pemrograman dapat diibaratkan seperti KBBI (kamus besar bahasa indonesia) yang harus Anda taati dalam menulis kode.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/34993?from=34988#run1-editor1)

1

hujan = True

2

​

3

if hujan:

4

print("silakan memakai jas hujan.")

5

​

6

'''

7

Output:

8

silakan memakai jas hujan.

9

'''

 Input  Reset

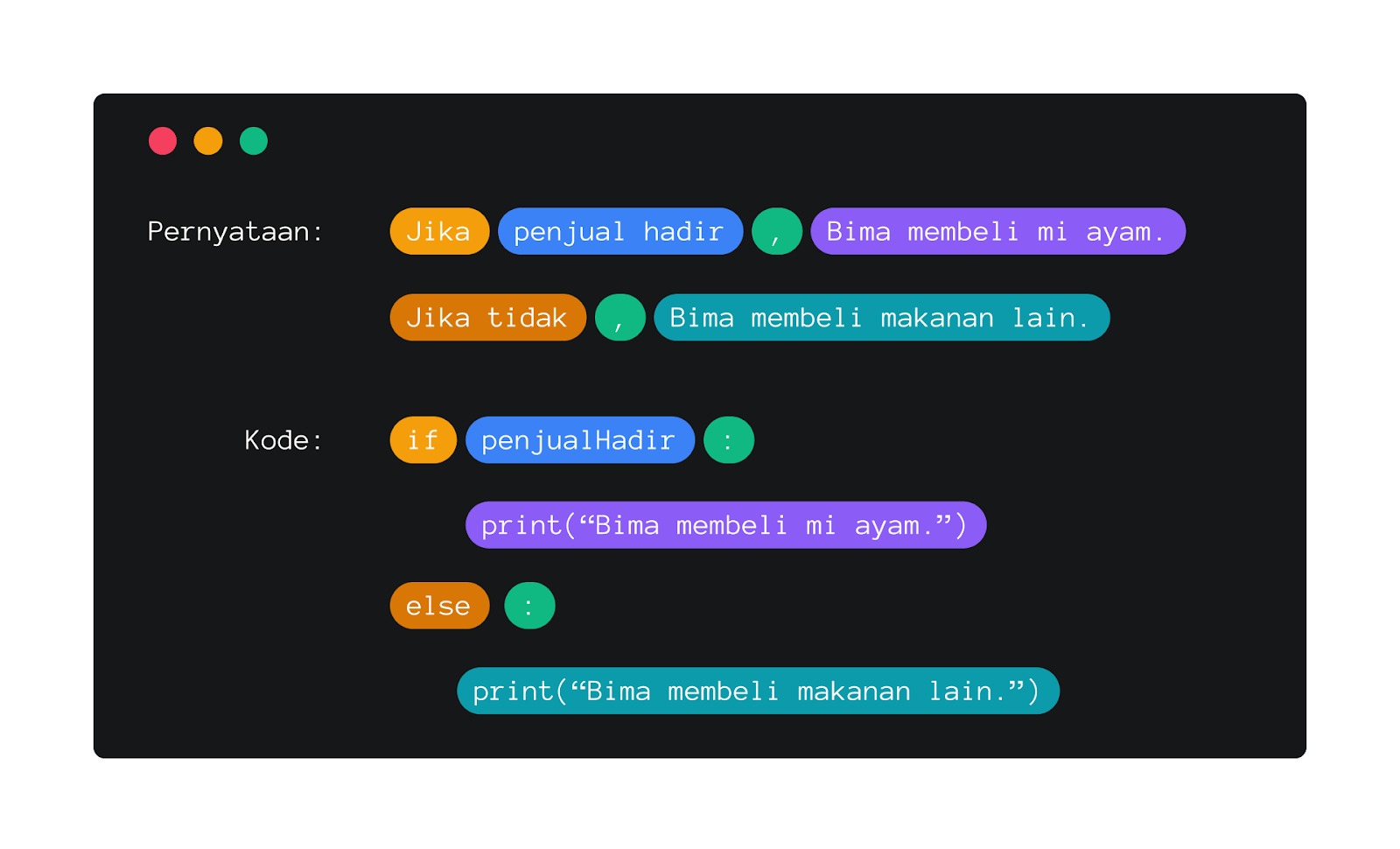
 Jalankan

Di awal sudah kita singgung bahwa pernyataan if harus bernilai True apabila blok program ingin dijalankan. Padahal, dalam kehidupan sehari-hari terdapat pilihan yang harus tetap berjalan ketika opsi tersebut tidak berjalan dengan semestinya. Saat pernyataan if bernilai False, seharusnya kita memiliki langkah lain supaya program tetap berjalan sesuai rencana.

Dalam konsep pengondisian, kita dapat menggunakan pernyataan **else** untuk menangani kondisi alternatif. Saat pernyataan if tidak terpenuhi atau bernilai False, blok program dalam pernyataan else akan dieksekusi sebagai jalur alternatif.

|  |
| --- |
| Ketika jam istirahat, Bima pergi ke kantin sekolah untuk membeli makanan. Karena sudah dalam kondisi lapar, Bima ingin sekali memakan mi ayam kesukaannya. Ia berniat untuk membelinya jika penjual mi ayam sedang berjualan di kantin. Namun, jika penjual mi ayam sedang absen, dengan terpaksa Bima membeli makanan ala kadarnya.  [Bima membeli nasi goreng karena kedai mi ayam sedang tutup.](https://www.dicoding.com/academies/620/tutorials/34993?from=34988)  Dari cerita di atas, kita bisa menarik kesimpulan bahwa Bima akan tetap makan pada jam istirahat. Akan tetapi, pilihan makanan yang Bima pilih berbeda bergantung dengan kondisi dari pernyataan “penjual mi ayam sedang berjualan di kantin.” Jika kondisi tersebut terpenuhi, Bima akan membeli mi ayam. Apabila yang terjadi sebaliknya, Bima akan membeli makanan ala kadarnya yang tersedia di kantin tersebut. |

Cara menuliskan pernyataan else mirip dengan penerapan if. Berikut adalah cara penulisan pengondisian else pada Python.

[](https://www.dicoding.com/academies/620/tutorials/34993?from=34988)

Berikut adalah deskripsi mengenai blok ilustrasi di atas.

* Blok oranye adalah pernyataan else.
* Blok hijau adalah karakter colon “:” sebagai pemisah pernyataan if/else dengan blok program.
* Blok cyan adalah blok program yang akan dijalankan apabila kondisi bernilai False.

Konsep if-else ini dapat Anda lakukan ketika memiliki satu kondisi yang diharapkan. Jika kondisi bernilai True, blok program A akan berjalan, sedangkan blok program B akan berjalan ketika kondisi bernilai False.

1. if kondisiTrue:
2. # blok program A
3. else:
4. # blok program B

Dengan kata lain, pernyataan if-else hanya memiliki dua opsi pilihan dengan satu kondisi yang dibandingkan.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/34993?from=34988#run2-editor1)

1

penjualHadir = False

2

​

3

if penjualHadir:

4

print("Bima membeli mi ayam.")

5

else:

6

print("Bima membeli makanan lain.")

7

​

8

​

9

'''

10

Output:

11

Bima membeli makanan lain.

12

'''

 Input  Reset

 Jalankan

Namun, apa jadinya bila kita memiliki kondisi lebih dari satu? Python memiliki pernyataan **elif** yang mampu menyediakan beberapa kondisi lebih dari satu. Kata kunci elif ini seperti else-if yang akan dievaluasi setelah pernyataan if dan sebelum else.

Berikut adalah struktur penulisan pengondisian if-elif-else dalam Python.

1. if kondisi1:
2. # Blok program dieksekusi jika kondisi1 benar
3. elif kondisi2:
4. # Blok program dieksekusi jika kondisi1 salah dan kondisi2 benar
5. elif kondisi3:
6. # Blok program dieksekusi jika kondisi1 dan kondisi2 salah, dan kondisi3 benar
7. else:
8. # Blok program dieksekusi jika semua kondisi di atas salah

Ketika pernyataan if dievaluasi dan bernilai False, kondisi dalam pernyataan elif berikutnya akan dievaluasi secara berurutan. Blok program akan dijalankan berdasarkan kondisi yang memenuhi syarat atau bernilai True. Jika tidak ada satu pun kondisi yang terpenuhi, blok program dalam pernyataan else akan dieksekusi.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/34993?from=34988#run3-editor1)

1

penjualMiAyamHadir = False

2

penjualNasiGorengHadir = True

3

​

4

if penjualMiAyamHadir:

5

print("Bima membeli mi ayam.")

6

elif penjualNasiGorengHadir:

7

print("Bima membeli nasi goreng.")

8

else:

9

print("Bima membeli makanan lain.")

10

​

11

​

12

'''

13

Output:

14

Bima membeli nasi goreng.

15

'''

 Input  Reset

 Jalankan

Jika melihat kode di atas, kondisi penjualMiAyamHadir tidak menghasilkan nilai True. Dengan kata lain, program akan mengevaluasi kondisi berikutnya, yaitu penjualNasiGorengHadir. Saat kondisi tersebut bernilai True (benar), program akan mengeksekusi blok program elif dengan mencetak teks “Bima membeli nasi goreng.”

Ingatlah bahwa hanya satu blok program yang akan dieksekusi pada pengondisian if, if-else, atau if-elif-else. Setelah satu kondisi terpenuhi, blok program di bawahnya akan dijalankan dan program akan keluar dari pengondisian tersebut. Ia akan mengabaikan blok program lainnya.

Perhatikan kode di bawah ini.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/34993?from=34988#run4-editor1)

1

penjualMiAyamHadir = False

2

penjualNasiGorengHadir = True

3

​

4

if penjualMiAyamHadir:

5

print("Bima membeli mi ayam.")

6

elif penjualNasiGorengHadir:

7

print("Bima membeli nasi goreng.")

8

else:

9

print("Bima membeli makanan lain.")

10

​

11

print("Bima memakan makanan di kantin.")

12

​

13

'''

14

Output:

15

Bima membeli nasi goreng.

16

Bima memakan makanan di kantin.

17

'''

 Input  Reset

 Jalankan

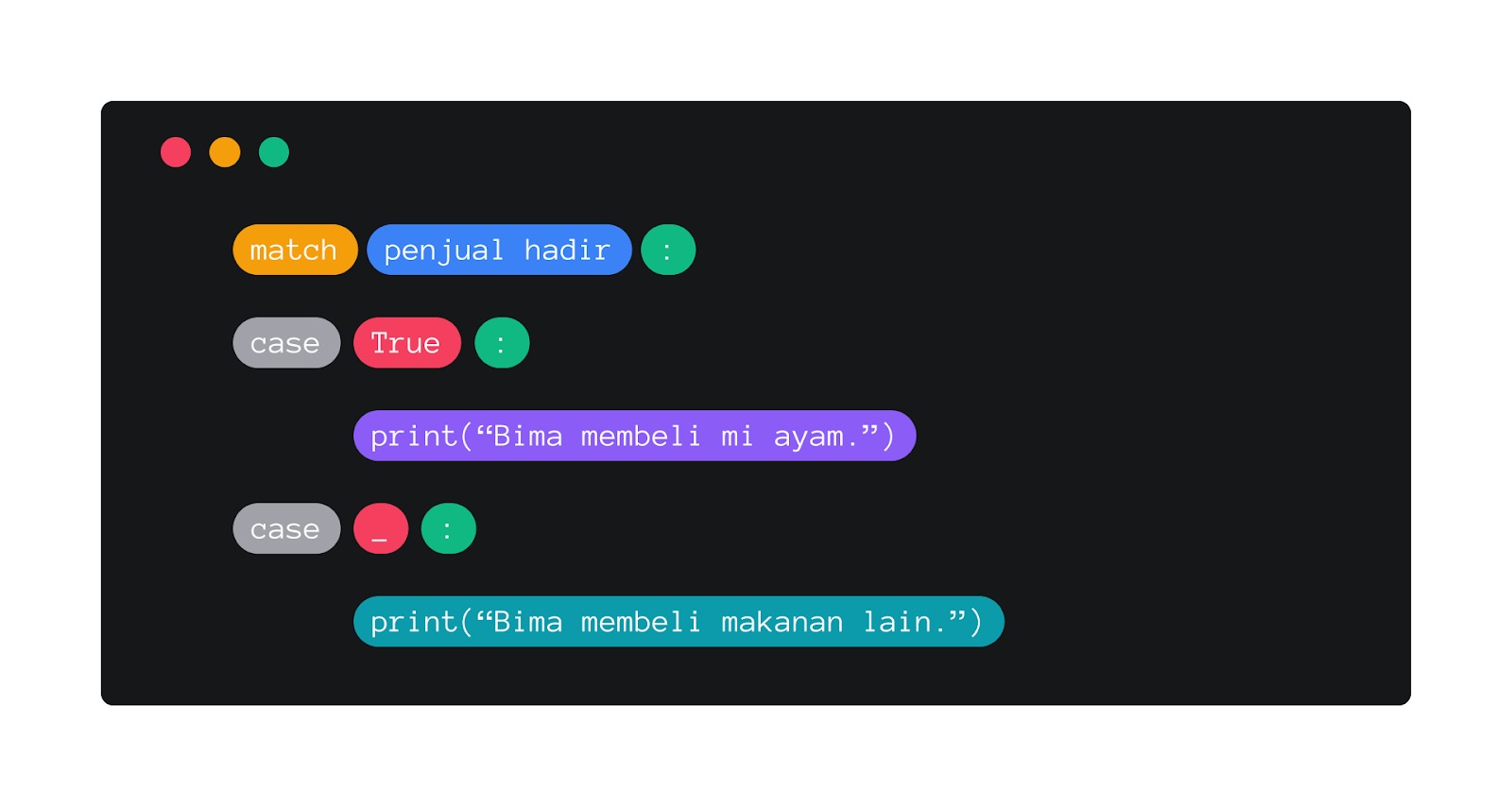
Saat kondisi penjualNasiGorengHadir terpenuhi, program akan menampilkan teks yang ada dalam blok program elif. Setelah itu, program akan mengabaikan blok program yang ada dalam pengondisian lainnya. Ia akan menjalankan perintah cetak di akhir algoritma program, yaitu teks “Bima memakan makanan di kantin.”

### Pernyataan match

Bayangkan Anda memiliki banyak pengondisian dengan membandingkan nilai yang sama seperti beberapa konstanta. Apakah Anda akan menggunakan pernyataan if-elif-else? Mungkin saja masih diperlukan. Namun, ada pernyataan lain serupa dengan if untuk membandingkan nilai yang sama berkali-kali, namanya pernyataan **match**.

Pernyataan match ini diperkenalkan dalam Python 3.10 yang mampu melakukan pemilihan berdasarkan pola atau pattern matching terhadap sebuah nilai. Konsep ini serupa dengan pernyataan switch pada bahasa pemrograman C, Java, Javascript, dll.

Berikut adalah tatanan penulisan pernyataan match pada Python.

[](https://www.dicoding.com/academies/620/tutorials/34993?from=34988)

Di bawah ini adalah deskripsi mengenai blok ilustrasi di atas.

* Blok kuning adalah kata kunci match.
* Blok abu adalah kata kunci case.
* Blok biru adalah variabel yang ingin dibandingkan.
* Blok hijau adalah karakter colon “:” sebagai pemisah pernyataan match/case dengan blok program.
* Blok merah adalah pola atau nilai pembanding.
* Blok ungu adalah blok program yang akan dijalankan apabila perbandingan antara variabel dan nilai pembanding bernilai True.
* Blok cyan adalah blok program yang akan dijalankan apabila hasil perbandingan tidak ada yang cocok atau sama.

Apakah Anda sudah bisa membayangkan cara kerja pernyataan match? Mari kita terjun ke suatu permasalahan. Apabila kita mempunyai algoritma untuk membandingkan warna dan menerapkan pernyataan if, program akan menjadi seperti berikut.

1. myFood = "mi ayam"
3. if myFood == "nasi goreng":
4. print("Makanan Anda adalah nasi goreng.")
5. elif myFood == "mi ayam":
6. print("Makanan Anda adalah mi ayam.")
7. elif myFood == "soto":
8. print("Makanan Anda adalah soto.")
9. elif myFood == "sate":
10. print("Makanan Anda adalah sate.")
11. else:
12. print("Kita tidak tahu makanan Anda.")

Melihat kode di atas, variabel myFood selalu dibandingkan dengan warna lainnya dan tidak ada pembanding lainnya. Selain itu, pernyataan “myFood ==” selalu berulang di setiap pernyataan. Dengan pernyataan match, kita dapat meringkasnya menjadi seperti berikut.

1. myFood = "mi ayam"
3. match myFood:
4. case "nasi goreng":
5. print("Makanan Anda adalah nasi goreng.")
6. case "mi ayam":
7. print("Makanan Anda adalah mi ayam.")
8. case "soto":
9. print("Makanan Anda adalah soto.")
10. case "sate":
11. print("Makanan Anda adalah sate.")
12. case \_:
13. print("Kita tidak tahu makanan Anda.")

Cara membandingkan dua nilai ini memiliki konsep yang serupa dengan pernyataan if. Namun, ada yang berbeda, yaitu karakter underscore ‘\_’ pada akhir case. Python menggunakan karakter tersebut untuk menunjukkan kondisi yang tidak cocok layaknya else pada pernyataan if.

Kata kunci yang dipakai adalah “match-variable-case-value” yang serupa dengan “if-variable-equal-value”. Penggunaan pernyataan match memungkinkan kode yang terlihat lebih ringkas dan efisien daripada menggunakan pernyataan if-elif-else. Dengan begitu, kita mampu menangani kasus-kasus tertentu dengan pola yang lebih efisien.

Pada saat menulis algoritma program atau kode dengan penambahan pengondisian di dalamnya, pastikan Anda memperhatikan indentasi dari setiap blok program yang ada. Penulisan blok program lebih menjorok ke dalam dibandingkan dengan pengondisian if-elif-else ataupun match-case. Dengan begitu, compiler atau interpreter mampu memahami maksud dari algoritma program yang Anda tulis.

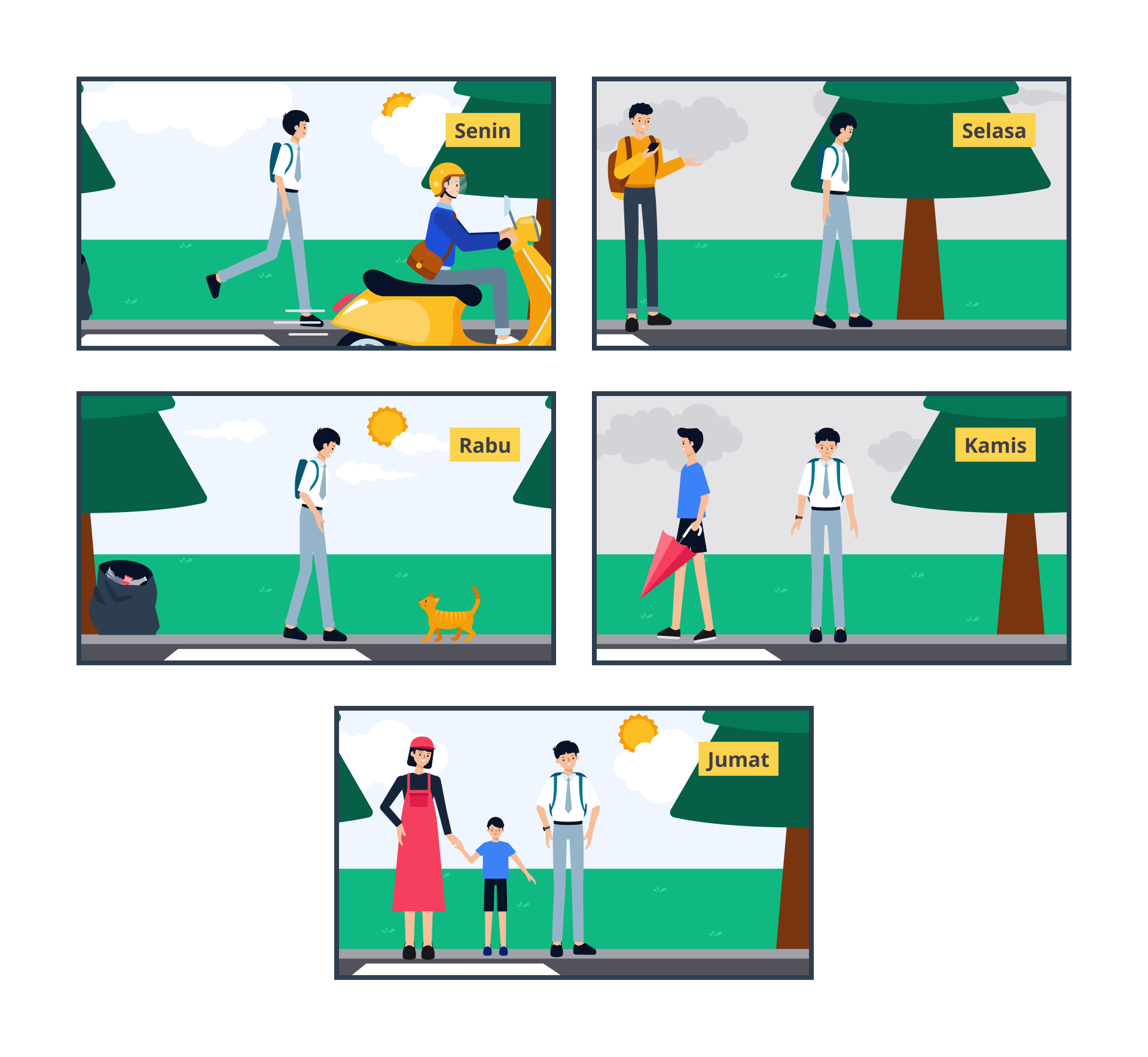
Apabila Anda ingin mengulik lebih dalam terkait pengondisian if-elif-else atau match-case, silakan kunjungi beberapa tautan berikut.

* [if Statements](https://docs.python.org/3.11/tutorial/controlflow.html#if-statements)
* [match Statements](https://docs.python.org/3.11/tutorial/controlflow.html#match-statements)

Yuk, segera kunjungi materi berikutnya supaya lebih memahami konsep pengondisian dalam bahasa pemrograman Python. Let’s go!

## Perulangan

Saat pertama kali masuk sekolah, Bima sangat bersemangat pergi ke sekolah. Sebelum pukul 7, dia sudah tiba di sekolah demi mengemban tugasnya sebagai pelajar. Mulai dari hari Senin hingga Jumat dan kembali lagi ke hari Senin, Bima lalui dengan antusias untuk menimba ilmu di sekolah. Namun, adakalanya dia mulai jenuh menerima materi yang diberikan oleh guru. Dia harus membulatkan tekad dan pantang menyerah demi meraih impian yang diidamkan.

[](https://www.dicoding.com/academies/620/tutorials/35008?from=35003)

### Kemampuan Komputer vs. Manusia

Kita bisa mendapatkan insight melalui pola perjalanan Bima meraih impian tersebut. Setiap hari, khususnya weekday atau hari kerja, Bima melakukan pola yang berulang, yaitu pergi ke sekolah. Suatu saat, Bima mulai merasakan jenuh sehingga terjadi penurunan performa karena ada faktor internal dan eksternal yang memengaruhinya. Walaupun dia akan bersemangat lagi, suatu saat Bima akan kembali mengalami hal yang serupa.

Berbeda dengan kemampuan komputer dalam menjalankan aksi yang berulang. Ia dapat mengulang suatu instruksi atau aksi secara berulang-ulang dengan performa yang sama. Komputer unggul pada beberapa aspek, terutama dalam melakukan aksi perulangan. Berikut adalah aspek yang membuat komputer lebih unggul dibandingkan manusia.

#### Kecepatan dan Efisiensi

Pemrosesan yang cepat membuat komputer lebih baik daripada manusia dalam melakukan aksi perulangan. Komputer memiliki prosesor yang mampu melakukan operasi aritmetika dan logika dalam hitungan detik bahkan milidetik. Manusia memiliki keterbatasan fisik dan kecepatan dalam melakukan tugas yang berulang.

#### Kapasitas

Dibekali penyimpanan yang lebih besar daripada manusia memungkinkan komputer mampu menyimpan dan memanipulasi data dalam jumlah besar. Proses perulangan dengan jumlah iterasi yang besar pun dapat dilakukan oleh komputer karena tersimpan pada memori yang besar.

#### Ketelitian

Tingkat ketelitian yang tinggi menjadi aspek yang diunggulkan oleh komputer dalam melakukan aksi. Manusia akan sering teledor dalam menjalankan aksi apabila dilakukan secara berulang-ulang. Lengah, lalai, dan bosan membuat konsentrasi manusia berkurang sehingga tingkat ketelitiannya menurun.

#### Konsistensi

Hal ini yang paling diunggulkan, yaitu konsistensi. Apabila komputer melakukan suatu aksi tertentu secara berulang, ia akan menghasilkan keluaran dengan durasi waktu yang seragam. Tidak adanya rasa jenuh yang membuat menurunnya performa komputer. Berbeda dengan manusia yang dapat merasa jenuh pada suatu waktu.

Berdasarkan aspek di atas, manusia memang memiliki kelemahan dalam melakukan aksi yang berulang. Namun, manusia memiliki keterampilan dalam mengasah kreativitas yang jauh lebih besar dibandingkan dengan komputer. Menganalisis permasalahan, memahami pola, berpikir kritis, dan mengidentifikasi konteks menjadi anugerah bagi manusia untuk membuat keputusan yang kompleks. Komputer hanyalah mengikuti instruksi yang telah didefinisikan dan tidak mempunyai aspek kreativitas tersebut.

[](https://www.dicoding.com/academies/620/tutorials/35008?from=35003)

Anyway, kita bisa memanfaatkan kemampuan komputer dalam menjalankan aksi perulangan. Kombinasi antara manusia dalam memahami konteks dan komputer dalam menjaga performa menjadi kolaborasi yang baik untuk menciptakan tugas yang terstruktur serta dapat menjalankan tugas sebanyak mungkin.

### Konsep Perulangan

Perulangan merupakan suatu aksi dalam pemrograman yang berfungsi untuk mengulang serangkaian aksi dalam waktu tertentu. Kita dapat menjalankan satu blok program secara berulang kali dengan kondisi atau jumlah iterasi tertentu.

Mengapa kita perlu mempelajari perulangan? Terkadang, kita perlu menjalankan suatu aksi secara terus menerus. Hal ini menjadikan proses automasi lebih mudah dalam mengulang serangkaian blok program tanpa menuliskannya secara berkali-kali. Dengan begitu, penulisan kode menjadi semakin fleksibel dan adaptif dalam menyelesaikan masalah.

Tahukah Anda bahwa looping (perulangan) dalam suatu program memiliki empat kunci utama? Secara tidak langsung, Anda telah memahami keempat bagian ini. Berikut adalah kunci utama dalam aksi perulangan.

#### State Awal

Kita perlu mendefinisikan state awal dalam menjalankan suatu perulangan. State ini dapat berupa sebuah variabel atau nilai tertentu bergantung pada tipe perulangan yang dipakai. Mendefinisikan state ini merupakan hal penting supaya mulainya proses perulangan dapat dipastikan.

Saat berangkat ke sekolah di awal pekan, Bima harus mengetahui hari ia memulai belajar. Apakah hari Senin, Selasa, atau hari lainnya? Dia tidak akan berangkat ke sekolah tanpa mengetahui waktu pembelajaran dimulai di awal pekan. Untuk itulah, Anda perlu mendefinisikan state awal supaya perulangan dapat terjadi dengan benar.

#### Kondisi Berhenti

Dalam suatu perulangan, ada pernyataan pengondisian yang harus terpenuhi. Maksudnya bagaimana? Kita harus mendefinisikan kondisi yang membuat proses perulangan tetap berjalan. Pada setiap iterasi, program akan mengevaluasi kondisi tersebut untuk menentukan perulangan harus dilanjutkan atau dihentikan.

Bayangkan penentuan kondisi lanjut atau berhenti ini seperti saat Bima memulai hari sekolah pada hari Senin, apakah dia akan berangkat tiap hari hingga Minggu? Umumnya, pembelajaran sekolah berakhir pada hari Jumat. Dengan kata lain, Bima harus mengetahui hari terakhir pembelajaran di sekolah.

Lalu, apa yang terjadi ketika dia tidak mengetahuinya? Bima akan setiap hari berangkat ke sekolah mulai hari Senin hingga Minggu dan kembali mengulanginya sampai tiada akhir. Jika hal itu terjadi di kehidupan nyata, dia akan merasa lelah, jenuh, dan capai untuk memulai pembelajaran di sekolah.

Maka dari itu, penentuan kondisi ini perlu diperhatikan dengan baik. Ia memberikan kontrol dan keamanan dalam mengeksekusi aksi perulangan. Kondisi berhenti memungkinkan kita untuk menghentikan perulangan ketika memenuhi syarat dan mencegah terjadinya infinite loop.

#### Badan Perulangan

Dalam melakukan aksi perulangan, ada iterasi yang berulang saat kondisi berhenti belum memenuhi syarat. Pada saat itulah serangkaian aksi atau satu blok program berulang dan menjalankan instruksinya. Satu blok program yang berulang itu disebut sebagai badan perulangan.

Ibaratnya, Bima mengetahui state awal dan kondisi berhenti, tetapi tidak ada aksi yang dijalankan. Perulangan tersebut sama saja tidak akan terjadi karena tidak ada aksi yang berjalan. Dia perlu mendefinisikan aksi yang harus berulang supaya setiap iterasi melakukan aksi yang sama.

#### Proses Iterasi

Kunci terakhir adalah proses iterasi. Lo, apakah iterasi ini sama dengan badan perulangan? Tentu tidak. Proses iterasi ini merupakan mekanisme perulangan dengan mengombinasikan ketiga kunci sebelumnya, yaitu state awal, kondisi akhir, dan badan perulangan. Kombinasi kunci ini menjadi satu kesatuan yang akan menjalankan aksi berikut.

* Mendefinisikan state awal.
* Melakukan evaluasi kondisi.
* Menjalankan badan perulangan.
* Memperbarui state.
* Mengulangi proses evaluasi kondisi.

Umumnya, saat proses iterasi berlangsung terjadi perubahan state yang diperlukan untuk iterasi berikutnya.

Bima memulai berangkat ke sekolah di awal pekan pada hari Senin. Dia mencoba berangkat lebih awal supaya tidak terlambat atau melebihi pukul 7 pagi. Besoknya, dia akan memulai mengulangi proses yang sama seperti hari sebelumnya. Di sanalah perubahan state yang awalnya hari Senin menjadi Selasa. Dengan begitu, Bima mengulangi serangkaian aksi yang sama.

Perulangan yang tepat pastinya memiliki keempat kunci utama. Apabila tidak terpenuhi dengan benar, ia tidak dapat dikatakan sebagai perulangan. Ia hanyalah serangkaian aksi yang hanya berjalan sekali tanpa adanya iterasi yang berulang. Materi selanjutnya, kita akan membahas beragam skema yang terjadi pada perulangan. Tunggu apa lagi, let’s go to the next journey!

## Skema Perulangan

Pada materi sebelumnya, kita telah membahas kemampuan komputer dalam melakukan aksi dan konsep perulangan secara umum. Saat ini kita akan belajar beragam skema yang terjadi pada perulangan.

Mengapa kita perlu belajar berbagai skema perulangan? Apakah satu konsep perulangan tidak cukup? Belajar berbagai pandangan dalam melakukan perulangan memungkinkan kita untuk memilih skema yang tepat berdasarkan masalah yang ada.

Ada empat macam skema perulangan yang perlu Anda ketahui. Berikut adalah daftar skema perulangan [1].

### Berdasarkan Jumlah Iterasi

Anda bisa menerapkan perulangan yang sederhana dengan menentukan jumlah iterasi yang diinginkan. Ketika ingin menjalankan serangkaian aksi secara berulang, kita hanya menentukan aksi tersebut beriterasi sebanyak n kali. Dengan cara ini, pengelolaan iterasi menjadi lebih mudah karena kita hanya menentukan jumlah iterasi yang diinginkan.

Perulangan yang menerapkan skema ini memiliki manfaat sebagai berikut.

* **Ketepatan jumlah iterasi**: program akan memastikan serangkaian aksi berulang sebanyak yang diminta karena hanya berlandaskan jumlah iterasi.
* **Kemudahan implementasi**: skema ini memudahkan kita sebagai pengembang aplikasi untuk menerapkannya.
* **Kejelasan logika program**: kemudahan dalam menerapkan skema ini menjadikan orang yang membacanya mudah memahaminya.

Sejatinya, kita dapat mengulang serangkaian aksi dengan menduplikasinya sebanyak yang diinginkan. Misalkan, kita ingin mencetak teks sebanyak lima kali sehingga kode Python akan seperti berikut.

1. print("Perintah ini berjalan sebanyak 5 kali.")
2. print("Perintah ini berjalan sebanyak 5 kali.")
3. print("Perintah ini berjalan sebanyak 5 kali.")
4. print("Perintah ini berjalan sebanyak 5 kali.")
5. print("Perintah ini berjalan sebanyak 5 kali.")

Namun, bagaimana ketika kita ingin mengulang perintah tertentu sebanyak ratusan bahkan ribuan? Masihkah kita menggunakan cara yang sama? Hal itu akan membuang waktu, tidak efisien, dan tidak praktis.

Dalam bahasa pemrograman Python, kita dapat memanfaatkan kata kunci **for-in**untuk menerapkan perulangan berdasarkan jumlah iterasi. Berikut adalah contoh program perulangannya.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35013?from=35008#run1-editor1)

1

for \_ in range(5):

2

print("Perintah ini berjalan sebanyak 5 kali.")

3

​

4

'''

5

Output:

6

Perintah ini berjalan sebanyak 5 kali.

7

Perintah ini berjalan sebanyak 5 kali.

8

Perintah ini berjalan sebanyak 5 kali.

9

Perintah ini berjalan sebanyak 5 kali.

10

Perintah ini berjalan sebanyak 5 kali.

11

'''

 Input  Reset

 Jalankan

Kode di atas akan menjalankan blok program berisikan teks yang dicetak sebanyak lima kali. Untuk menerapkannya, perlu ada kata kunci **range** saat melakukan perulangan for-in. Selain itu, terdapat karakter underscore “\_” yang merujuk pada variabel temporer. Gunanya untuk mengabaikan nilai yang tidak akan digunakan dalam iterasi.

### Berdasarkan Pencacah

Selanjutnya adalah perulangan berlandasan pencacah. Maksudnya bagaimana? Pencacah ini memiliki arti seperti “mencincang” yang merujuk pada membagi list menjadi bagian kecil-kecil dan melakukan perulangan berdasarkan potongan elemen tersebut. Potongan tersebut menjadi iterator atau state yang akan berubah pada setiap iterasinya.

Bayangkan kita ingin mencetak angka 0, 1, 2, hingga 4. Dengan kata lain, program akan beriterasi sebanyak 5 kali karena diawali angka 0 dan berakhir di angka 4. Susunan angka 0 hingga 4 tersebut dapat ditulis menggunakan perulangan **for-in** dengan range(5).

Lo, mengapa sama seperti sebelumnya? Iya, secara tidak langsung perulangan tersebut mencacah angka mulai 0 sampai 4 sehingga beriterasi sebanyak 5 kali. Akan tetapi, pada perulangan sebelumnya tidak memanfaatkan variabel temporer. Saat ini kita bisa memanfaatkannya dengan mengganti karakter underscore menjadi variabel tertentu. Berikut adalah salah satu contoh perulangan berdasarkan pencacah.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35013?from=35008#run2-editor1)

1

for i in range(5):

2

print(f"Perintah ini mencetak angka {i}")

3

​

4

'''

5

Output:

6

Perintah ini mencetak angka 0.

7

Perintah ini mencetak angka 1.

8

Perintah ini mencetak angka 2.

9

Perintah ini mencetak angka 3.

10

Perintah ini mencetak angka 4.

11

'''

 Input  Reset

 Jalankan

Dengan memanfaatkan variabel temporer, kita dapat menyimpan nilai sementara dan digunakan dalam badan perulangan.

Selain menggunakan for-in dengan range, ada cara lainnya sehingga kita bisa menerapkan perulangan berdasarkan pencacah. Kita perlu memanfaatkan variabel bertipe data **list**, **tuple**, atau **string**. Ketiga tipe data tersebut merupakan [iterable](https://wiki.python.org/moin/Iterator) yang memungkinkan kita untuk mengulangi serangkaian aksi berdasarkan potongan objek dari variabel. Berikut contohnya.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35013?from=35008#run3-editor1)

1

print("Dengan list/tuple")

2

for i in [0,"Bima", True, 97.5]:

3

print(f"Perintah ini mencetak nilai {i}.")

4

​

5

print("\nDengan string")

6

for i in "Bima":

7

print(f"Perintah ini mencetak karakter {i}.")

8

​

9

​

10

'''

11

Output:

12

Dengan list/tuple

13

Perintah ini mencetak nilai 0.

14

Perintah ini mencetak nilai Bima.

15

Perintah ini mencetak nilai True.

16

Perintah ini mencetak nilai 97.5.

17

​

18

Dengan string

19

Perintah ini mencetak karakter B.

20

Perintah ini mencetak karakter i.

 Input  Reset

 Jalankan

Perulangan dengan skema ini memudahkan kita untuk melakukan perulangan berdasarkan potongan objek yang dihasilkan melalui proses pencacahan. Berbekal list/tuple memungkinkan program untuk mengulang setiap elemen dengan tipe data yang variatif. Fleksibilitas tersebut membuat algoritma tidak terikat dengan pengindeksan secara manual sehingga kode terlihat lebih bersih dan mudah dibaca.

### Berdasarkan Evaluasi Kondisi Berhenti

Skema sebelumnya hanya berfokus pada proses iterasi dengan jumlah pencacahan suatu objek. Berbeda dengan skema yang satu ini. Skema yang akan dibahas kali ini adalah perulangan berdasarkan kondisi berhenti. Maksudnya bagaimana? Ingatkah Anda dengan kondisi berhenti dalam perulangan? Pada skema ini proses evaluasi pada kondisi berhenti lebih ditekankan.

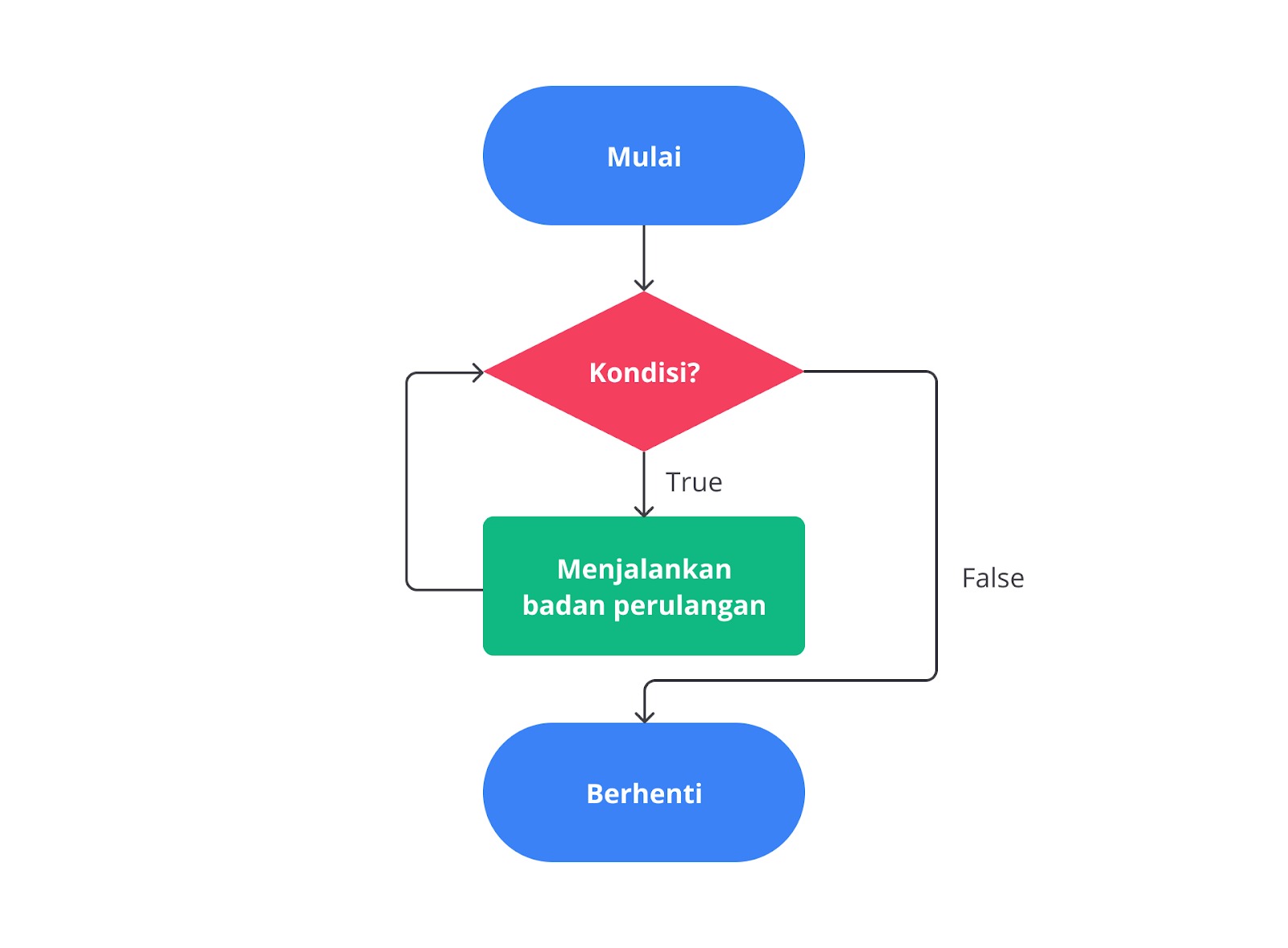
Adanya skema ini memiliki keunggulan tersendiri dibandingkan dengan yang lain. Berikut adalah manfaat ketika Anda menerapkan skema perulangan berdasarkan evaluasi kondisi berhenti.

* **Fleksibilitas**: kemudahan dalam mengulang suatu blok program membuat skema ini fleksibel dalam mengatur berhentinya proses iterasi.
* **Menghemat sumber daya**: melalui kondisi yang tepat memungkinkan program untuk terhindar dari pemborosan sumber daya komputasi.
* **Menyediakan kontrol alur program**: Anda dapat mengatur jalannya program dengan memberikan kontrol melalui kondisi berhenti yang terdefinisi dengan baik.
* **Mampu menangani kasus khusus**: adanya pengontrol membuat program dapat mengatur dan menjalankan perulangan dalam kondisi yang khusus atau berbeda dengan skema lainnya.

Skema ini memiliki dua jenis penanganan. Perbedaan antara kedua jenis ini terletak pada waktu penanganan evaluasi serta awal dan akhir iterasi. Berikut penjelasan kedua jenis penanganannya.

#### Evaluasi pada Awal Iterasi

Sebagaimana namanya, proses evaluasi kondisi berhenti terjadi pada awal iterasi. Dengan kata lain, sebelum menjalankan blok program atau badan perulangan, program mengevaluasi kondisi berhenti terlebih dahulu. Apabila kondisi terpenuhi atau bernilai True, program melanjutkan proses berikutnya, yaitu menjalankan badan perulangan. Jika digambarkan dalam sebuah flowchart, alur program akan seperti berikut.

[](https://www.dicoding.com/academies/620/tutorials/35013?from=35008)

Dalam bahasa pemrograman Python, kita dapat memanfaatkan kunci **while** dalam menerapkan skema pada jenis ini. Berikut contoh program perulangannya.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35013?from=35008#run4-editor1)

1

iterateNumber = 0

2

while iterateNumber < 5:

3

print(f"Perintah ini mencetak angka {iterateNumber}")

4

iterateNumber = iterateNumber + 1

5

​

6

'''

7

Output:

8

Perintah ini mencetak angka 0.

9

Perintah ini mencetak angka 1.

10

Perintah ini mencetak angka 2.

11

Perintah ini mencetak angka 3.

12

Perintah ini mencetak angka 4.

13

'''

 Input  Reset

 Jalankan

Lo, apa bedanya dengan skema yang sebelumnya? Padahal, keluaran yang dihasilkan oleh program sama seperti skema lain dengan melakukan iterasi sebanyak 5 kali. Benar, hasilnya tidak ada yang berbeda.

Akan tetapi, ada perbedaan pada pola pikir atau algoritma yang diusung dalam proses iterasi. Perulangan **while** memerlukan suatu kondisi yang harus terpenuhi sebelum menjalankan badan perulangan. Apabila tidak terpenuhi, proses iterasi tidak akan berjalan dan keluar dari perulangan.

Perlu diingat bahwa dalam proses iterasi ini, Anda harus memperhatikan variabel yang dipakai dalam kondisi berhenti. Ketika kondisi berhenti selalu bernilai True, program akan berjalan secara terus menerus tanpa henti. Bayangkan saja kode di atas tidak menjalankan penambahan angka satu. Kondisi tersebut akan selalu terpenuhi dan mengulangi perintah yang sama hingga program mengalami infinite loop.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35013?from=35008#run5-editor1)

1

iterateNumber = 0

2

while iterateNumber < 5:

3

print(f"Perintah ini mencetak angka {iterateNumber}")

4

# baris ini dihapus

5

# iterateNumber = iterateNumber + 1

6

​

7

'''

8

Output:

9

Perintah ini mencetak angka 0.

10

Perintah ini mencetak angka 0.

11

Perintah ini mencetak angka 0.

12

Perintah ini mencetak angka 0.

13

…

14

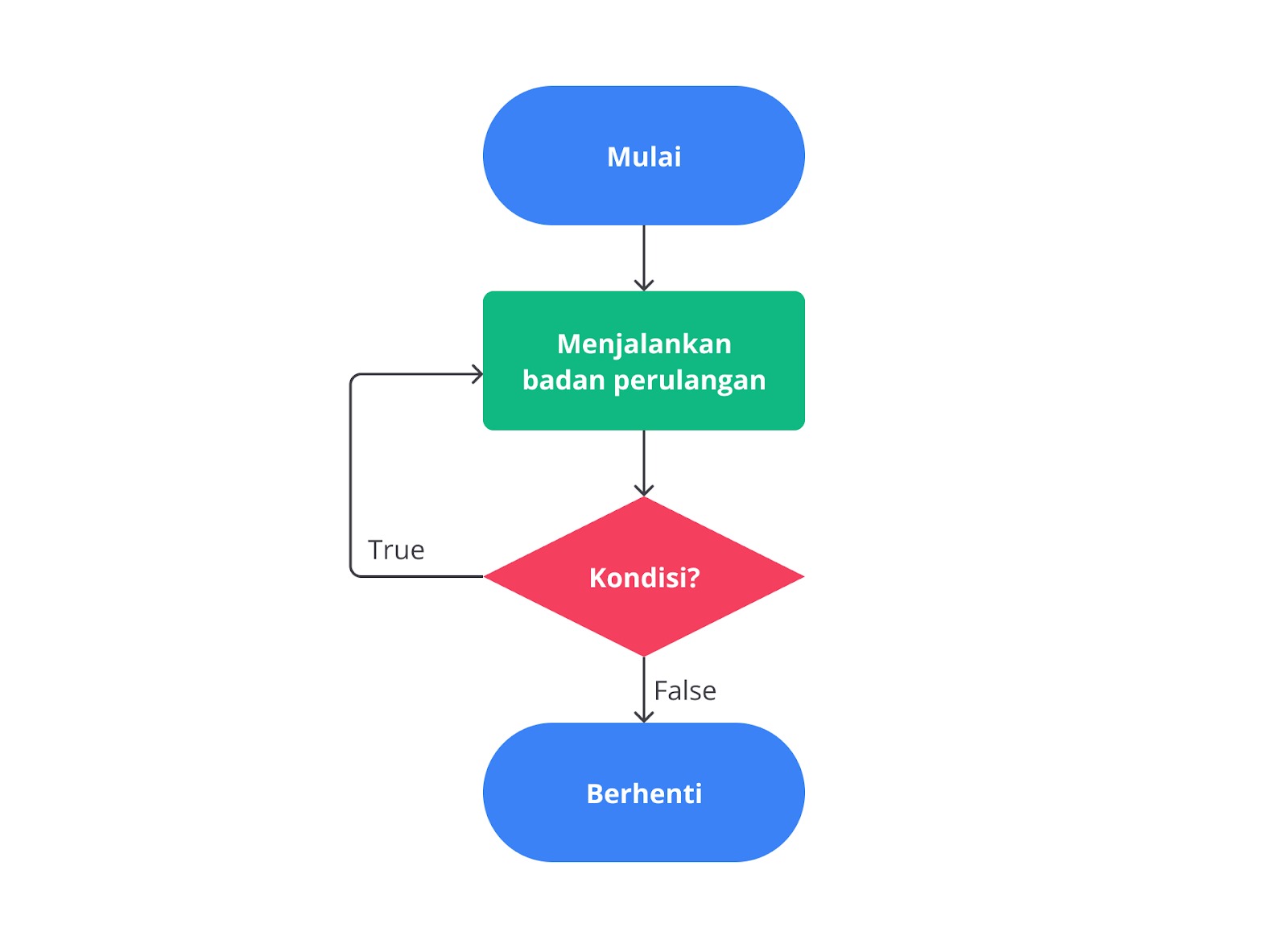
'''

 Input  Reset

 Jalankan

#### Evaluasi pada Akhir Iterasi

Berbeda dengan di awal, proses evaluasi pada perulangan ini terjadi di akhir iterasi. Jadi, badan perulangan akan dijalankan dahulu, kemudian dilanjutkan dengan mengevaluasi kondisi perulangan. Apabila kondisi terpenuhi atau menghasilkan nilai True, perulangan akan berlanjut ke iterasi berikutnya. Jika digambarkan dalam sebuah flowchart, alur program akan seperti berikut.

[](https://www.dicoding.com/academies/620/tutorials/35013?from=35008)

Berdasarkan flowchart tersebut, badan perulangan akan berjalan setidaknya sekali. Saat mengevaluasi kondisi menghasilkan False, perulangan tidak akan berlanjut.

Pada umumnya, perulangan yang berjenis seperti ini dapat dilakukan dengan kata kunci do-while. Anda dapat menjumpai perulangan ini pada bahasa pemrograman C.

* [**main.c**](https://www.dicoding.com/academies/620/tutorials/35013?from=35008#run6-editor1)

1

#include <stdio.h>

2

​

3

int main(void) {

4

int iterateNumber = 0;

5

do {

6

printf("Perintah ini mencetak angka %i\n", iterateNumber);

7

iterateNumber = iterateNumber + 1;

8

} while( iterateNumber < 5 );

9

}

10

​

11

/\*

12

Output:

13

Perintah ini mencetak angka 0.

14

Perintah ini mencetak angka 1.

15

Perintah ini mencetak angka 2.

16

Perintah ini mencetak angka 3.

17

Perintah ini mencetak angka 4.

18

\*/

 Input  Reset

 Jalankan

Perintah atau konsep do-while tidak tersedia dalam bahasa pemrograman Python. Perintah yang tersedia pada Python hanyalah konsep while yang diawali dengan mengevaluasi kondisi, kemudian menjalankan badan perulangan. Namun, Anda bisa memanipulasi kode supaya memiliki konsep yang serupa dengan do-while.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35013?from=35008#run7-editor1)

1

iterateNumber = 0

2

iterateValue = True

3

while iterateValue:

4

print(f"Perintah ini mencetak angka {iterateNumber}")

5

iterateNumber = iterateNumber + 1

6

if iterateNumber >= 5:

7

iterateValue = False

8

​

9

'''

10

Output:

11

Perintah ini mencetak angka 0.

12

Perintah ini mencetak angka 1.

13

Perintah ini mencetak angka 2.

14

Perintah ini mencetak angka 3.

15

Perintah ini mencetak angka 4.

16

'''

 Input  Reset

 Jalankan

Secara alur program, program menerapkan hal yang serupa dengan konsep do-while. Akan tetapi, pengecekan dalam perulangan ini sedikit berbeda. Pada perulangan jenis ini, ada dua pengondisian, yaitu memeriksa variabel iterateValue pada while dan iterateNumber menggunakan pernyataan if.

Dari penjelasan kedua jenis penanganan kondisi tersebut, Python hanya memberikan fitur **while** yang diawali dengan mengevaluasi kondisi dan dilanjutkan dengan menjalankan badan perulangan. Menilik beberapa contoh di atas, secara tidak langsung kita hanya menjalankan badan perulangan berdasarkan jumlah iterasi. Apa bedanya dengan skema lain?

Dengan perulangan while, kita mampu memanipulasinya supaya tidak terikat dengan jumlah iterasi. Inilah yang biasa dipakai ketika menangani permasalahan atau kasus tertentu. Caranya bagaimana? Biasanya, kondisi berhenti ini bergantung pada input yang terima melalui pengguna atau lainnya. Berikut contoh programnya.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35013?from=35008#run8-editor1)

1

iterateValue = True

2

while iterateValue:

3

char = input("Silakan kirim karakter Y/y untuk menghentikan iterasi! ")

4

if char == 'y' or char == 'Y':

5

iterateValue = False

6

​

7

'''

8

Output:

9

Silakan kirim karakter Y/y untuk menghentikan iterasi! n

10

Silakan kirim karakter Y/y untuk menghentikan iterasi! N

11

Silakan kirim karakter Y/y untuk menghentikan iterasi! Y

12

'''

 Input  Reset

 Jalankan

Apabila pengguna tidak mengirimkan karakter ‘y’ atau ‘Y’, program akan menjalankan badan perulangan secara terus menerus. Kapan proses iterasi berhenti? Ya, saat pengguna menekan karakter yang diminta, yaitu  ‘y’ atau ‘Y’. Di situlah pengondisian if menghasilkan True yang menyebabkan variabel iterateValue bernilai False. Program akan menghentikan proses iterasi karena hasil evaluasinya bernilai False.

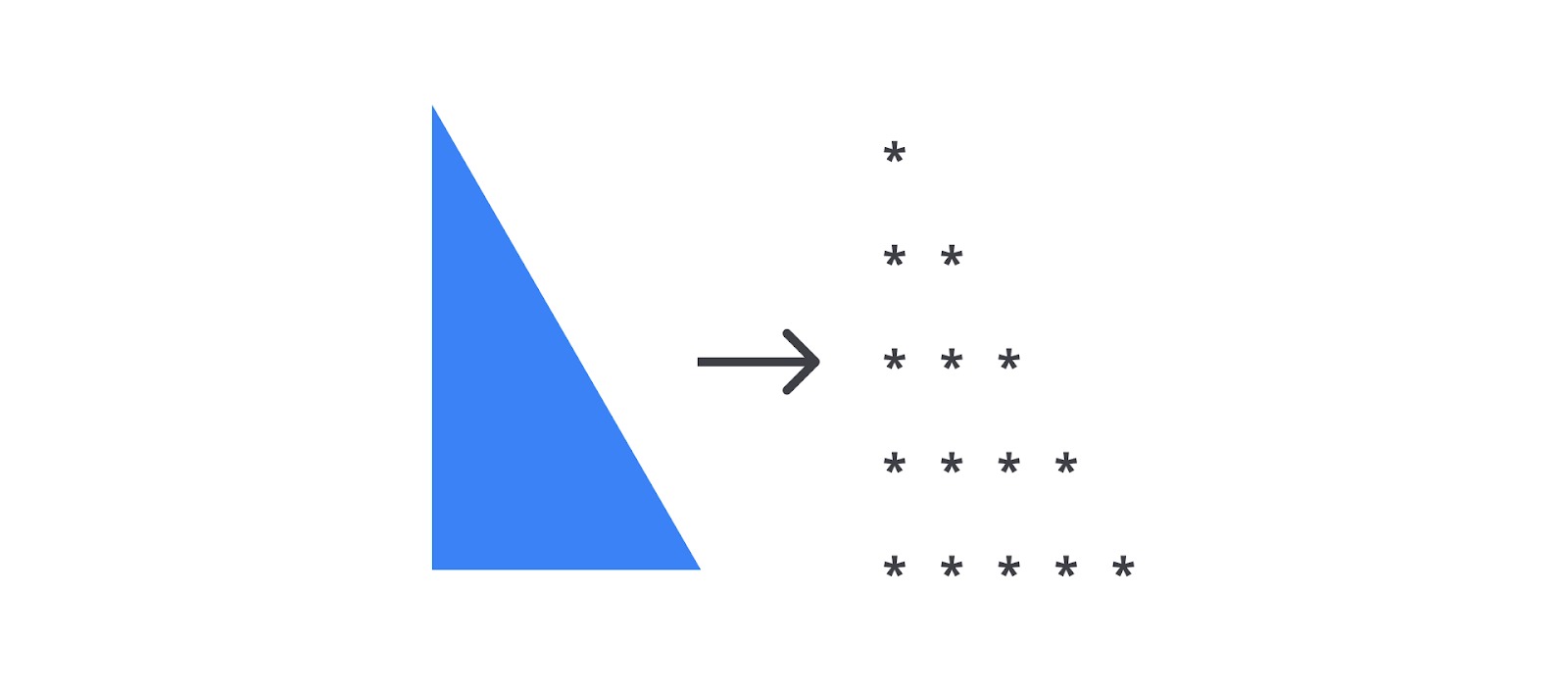
### Berdasarkan Kompleksitas Struktur

Terkadang, ada permasalahan yang memiliki tingkat kompleksitas tinggi. Butuh adanya aksi yang berulang dalam suatu perulangan lainnya. Hal ini menyebabkan struktur perulangan yang bersarang atau dikenal dengan istilah nested loop. Dari asal katanya, nested artinya bersarang dan loop artinya perulangan yang memungkinkan program memiliki perulangan dalam perulangan.

|  |
| --- |
| Saat pelajaran sedang berlangsung, Bima mendapatkan tugas dari guru dengan soal yang lumayan menantang. Dia mencoba untuk menyelesaikannya dengan mencari buku-buku terkait tugasnya di perpustakaan. Bima mengambil beberapa buku dan ditumpuk pada satu tempat. Dia membaca dan memeriksa halaman satu per satu dan lanjut ke buku berikutnya hingga mendapatkan jawaban yang dicari.  [Bima sedang mencari materi matematika dari tiga buku perpustakaan yang sudah ditumpuk di meja.](https://www.dicoding.com/academies/620/tutorials/35013?from=35008)  Kasus tersebut dapat dianalogikan seperti **nested loop** ketika tumpukan buku mewakili perulangan luar dan memeriksa setiap halaman buku mewakili perulangan dalam. Setiap berpindah ke tumpukan berikutnya, Bima mengulangi kembali proses memeriksa tiap halaman di dalam bukunya.  [Ilustrasi perulangan dalam dan luar.](https://www.dicoding.com/academies/620/tutorials/35013?from=35008) |

Contohnya, Anda ingin membuat program mencetak teks yang berbentuk seperti segitiga siku-siku. Secara bentuk, segitiga mempunyai susunan unik dengan setiap barisnya memiliki panjang berbeda.

Kita bisa memanipulasi bentuk segitiga dengan karakter yang membentuk segitiga. Apabila diilustrasikan, program akan menghasilkan gambar seperti berikut.

[](https://www.dicoding.com/academies/620/tutorials/35013?from=35008)

Bagaimana cara membuatnya? Bayangkan kita mempunyai baris dan kolom. Baris dapat diumpamakan sebagai perulangan luar dan kolom diibaratkan sebagai perulangan dalam. Baris pertama memiliki satu karakter saja. Kemudian, baris kedua memiliki dua karakter. Hal ini berlanjut pada baris dan kolom berikutnya. Berdasarkan pola tersebut, kita bisa beranggapan bahwa perulangan dalam (kolom) mengikuti baris yang sedang dicetak. Jadi, pola program akan seperti berikut.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35013?from=35008#run9-editor1)

1

for i in range (1,6):

2

for j in range (i):

3

print("\*", end = ' ')

4

print("")

5

6

'''

7

Output:

8

\*

9

\* \*

10

\* \* \*

11

\* \* \* \*

12

\* \* \* \* \*

13

'''

 Input  Reset

 Jalankan

Perulangan luar dengan variabel temporer i akan beriterasi sebanyak jumlah baris, sedangkan perulangan dalam dengan variabel temporer j akan beriterasi sebanyak variabel i. Apakah jumlah iterasi dapat mengikuti variabel tertentu seperti perulangan dalam? Tentu bisa. Inilah salah satu keunggulan dari penerapan nested loop dalam menjalankan penugasan yang fleksibel mengikuti alur program.

Berikut adalah manfaat dari penerapan nested loop.

* **Dapat mengakses setiap kombinasi elemen**: kita dapat mengakses dan mengombinasikan elemen dari struktur data yang berbeda karena ia memiliki lebih dari satu variabel temporer.
* **Memiliki struktur perulangan yang terorganisasi**: memungkinkan kita untuk mengatur dan mengontrol perulangan dengan terstruktur.
* **Mampu menangani tugas kompleks**: Anda dapat menyelesaikan penugasan kompleks dengan menciptakan pola perulangan yang sesuai kebutuhan dengan nested loop.
* **Fleksibel dalam memanipulasi data**: mengombinasikan seluruh variabel temporer memungkinkan kita untuk memanipulasi data secara terstruktur.

Perlu diingat bahwa perulangan memiliki waktu eksekusi tersendiri. Pasti durasi waktunya akan berbeda apabila serangkaian aksi berjalan sebanyak satu kali. Dengan adanya nested loop, struktur perulangan semakin dalam dan kompleks. Apabila semakin bersarang, durasi eksekusi perulangan akan semakin lama. Oleh karena itu, penting bagi kita untuk mempertimbangkan efisiensi dan kompleksitas algoritma program saat menggunakan nested loop.

Nah, itulah beberapa skema perulangan yang perlu Anda ketahui. Dengan begitu, kita bisa menjadi lebih selektif dalam memilih skema yang tepat berdasarkan permasalahan yang ada. Tidak serta-merta memilih perulangan for-in atau while di segala penyelesaian, tetapi mempertimbangkan masalah yang sedang dihadapi.

Apabila Anda ingin menambah wawasan terkait perulangan for-in dan while, silakan kunjungi tautan berikut.

* [for Statements](https://docs.python.org/3/tutorial/controlflow.html?highlight=loop#for-statements)
* [The range() Function](https://docs.python.org/3/tutorial/controlflow.html?highlight=loop#the-range-function)
* [break and continue Statements, and else Clauses on Loops](https://docs.python.org/3/tutorial/controlflow.html?highlight=loop#break-and-continue-statements-and-else-clauses-on-loops)
* [For loops](https://wiki.python.org/moin/ForLoop)
* [While loops](https://wiki.python.org/moin/WhileLoop)

## Dinamika Perulangan

Pada perulangan suatu algoritma, ada hal yang terjadi di sana, seperti perubahan state, menjalankan aksi dalam badan perulangan, proses evaluasi kondisi, dan proses iterasi untuk mengulang serangkaian aksi. Semua hal tersebut saling terikat dan berperan penting untuk menjalankan logika perulangan dengan benar.

Nyatanya, ada hal lain yang dapat terjadi ketika melakukan perulangan. Maksudnya bagaimana? Saat menjalankan perulangan, algoritma yang kita susun pada setiap iterasi harus menghasilkan efek neto positif atau sesuatu yang diharapkan. Apabila ada hambatan, kondisi, atau permasalahan tertentu, program harus menghasilkan efek neto positif juga.

Oleh sebab itu, kita perlu mengetahui hambatan, kondisi, atau masalah yang mampu ditangani dan tetap menghasilkan sesuatu yang kita harapkan. Yuk, kita bahas satu per satu.

### Putus atau Terus

Sekolah Bima memiliki empat lantai. Ketika ingin ke ruangan guru di lantai dua, Bima perlu menaiki tangga dari lantai satu menuju tujuan. Saat menaiki tangga dan mencapai lantai dua, Bima berhenti dan tidak melanjutkan naik ke lantai berikutnya.

Berdasarkan cerita tersebut, Bima sedang melakukan perulangan dengan menaiki anak tangga menuju lantai tujuan, yaitu lantai 2. Saat berhenti di lantai 2, Bima tidak melanjutkan ke lantai berikutnya karena sudah mencapai tujuan. Proses berhenti atau “putus”-nya iterasi ini dapat terjadi pada perulangan dengan menerapkan pernyataan **break**. Kondisi itu membuat perulangan terhenti dan tidak melanjutkan perulangan lagi.

Berikut adalah contoh pernyataan break dalam bahasa pemrograman Python.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35018?from=35013#run1-editor1)

1

for i in range(1, 5):

2

print(f"Ini adalah lantai {i}")

3

if i == 2:

4

print(f"Berhenti di lantai {i}")

5

break

6

​

7

print(f"Melewati lantai {i}")

8

​

9

'''

10

Output:

11

Ini adalah lantai 1

12

Melewati lantai 1

13

Ini adalah lantai 2

14

Berhenti di lantai 2

15

'''

 Input  Reset

 Jalankan

Saat berada pada iterasi kedua, program akan mengevaluasi kondisi menggunakan pernyataan if dan masuk ke blok program baru. Di dalamnya terdapat pernyataan break yang bertujuan untuk keluar dari iterasi yang ada. Dengan begitu, iterasi tidak dilanjutkan hanya sampai pada kondisi i bernilai 2.

Coba perhatikan hasil dari kode di atas, perintah setelah break, yaitu mencetak teks “melewati lantai”, tidak dieksekusi. Mengapa demikian? Proses break membuat program keluar dari perulangan tanpa memedulikan instruksi di bawahnya. Dengan begitu, perintah mencetak teks tidak dijalankan.

Di hari berikutnya, Bima mendapatkan mata pelajaran praktikum biologi di ruang laboratorium lantai tiga. Ketika Bima menaiki tangga, mungkin ada level bangunan yang tidak relevan dengan tujuannya. Saat berada di lantai dua, Bima tetap melanjutkan menaiki tangga karena tujuannya adalah lantai tiga. Saat berada di lantai tiga, Bima berhenti dan memasuki ruangan laboratorium.

Berdasarkan alur Bima untuk mencapai lantai 3, dia perlu melewati lantai satu dan dua. Saat berada di lantai 3, Bima berhenti dan tidak melanjutkan menaiki tangga. Hal tersebut serupa dengan perulangan yang melewati atau meneruskan iterasi perulangan. Dalam Python, terdapat pernyataan **continue** untuk melanjutkan atau meneruskan perulangan tanpa memedulikan perintah di bawahnya.

Berikut adalah contoh pernyataan break dalam bahasa pemrograman Python.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35018?from=35013#run2-editor1)

1

for i in range(1, 5):

2

print(f"Ini adalah lantai {i}")

3

4

if i != 3:

5

print(f"Melewati lantai {i}")

6

continue

7

​

8

print("Berhenti di lantai 3")

9

break

10

​

11

​

12

'''

13

Output:

14

Ini adalah lantai 1

15

Melewati lantai 1

16

Ini adalah lantai 2

17

Melewati lantai 2

18

Ini adalah lantai 3

19

Berhenti di lantai 3

20

'''

 Input  Reset

 Jalankan

Saat berada pada iterasi pertama atau kedua, program akan mengevaluasi kondisi dan mengeksekusi perintah continue. Secara tidak langsung, perintah di bawahnya tidak dilaksanakan dan program akan mengulang ke iterasi berikutnya. Saat pengondisian tidak terpenuhi, program akan meloloskan perintah continue dan menjalankan aksi break.

### Terikat dengan Hubungan Perulangan

Sejatinya, program akan mengevaluasi kondisi berdasarkan perubahan state yang sebelumnya sudah ditentukan pada setiap iterasi. Artinya, hanya ada variabel state yang selalu berubah dalam satu perulangan. Berikut adalah perulangan dengan hanya variabel state yang berubah pada setiap iterasinya.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35018?from=35013#run3-editor1)

1

for i in range(5):

2

print(f"Perintah ini mencetak angka {i}")

3

​

4

'''

5

Output:

6

Perintah ini mencetak angka 0.

7

Perintah ini mencetak angka 1.

8

Perintah ini mencetak angka 2.

9

Perintah ini mencetak angka 3.

10

Perintah ini mencetak angka 4.

11

'''

 Input  Reset

 Jalankan

Namun, apakah Anda berpikir bahwa sebuah perulangan berjalan dengan mengandalkan variabel state saja? Tentu tidak. Variabel state memiliki peran penting dalam perulangan untuk menyimpan informasi terkait status saat iterasi berlangsung. Dalam permasalahan tertentu, kita bisa menggunakan komponen lain untuk menjalankan perulangan sesuai algoritma.

Adakalanya, kita perlu mengombinasikan variabel selain state ke dalam perulangan untuk memperluas kemampuan dan fleksibilitas algoritma yang kita buat. Dengan menggunakan variabel tambahan, kita dapat melacak, menyimpan, dan memanipulasi informasi yang diperlukan selama proses perulangan. Hal ini memungkinkan kita untuk melakukan operasi khusus, membuat kondisi baru, atau menghitung hasil yang lebih kompleks.

Supaya terbayang, coba kita menyelesaikan suatu permasalahan. Ketika ingin menyelesaikan masalah penjumlahan dalam suatu daftar angka, kita memerlukan wadah untuk menyimpan jumlah secara sementara. Dengan begitu, pada iterasi terakhir, program akan mendapatkan nilai akhir dari jumlah kumpulan angka tersebut. Berikut adalah program pada permasalahan ini.

* [**main.py**](https://www.dicoding.com/academies/620/tutorials/35018?from=35013#run4-editor1)

1

total = 0

2

for value in [2, 12, 8, 5]:

3

total = total + value

4

print(f"Total sementara adalah {total}")

5

​

6

print(f"Total akhir adalah {total}")

7

​

8

'''

9

Output:

10

Total sementara adalah 2

11

Total sementara adalah 14

12

Total sementara adalah 22

13

Total sementara adalah 27

14

Total akhir adalah 27

15

'''

 Input  Reset

 Jalankan

Variabel total adalah variabel selain state yang dipakai dalam perulangan. Fungsinya untuk menyimpan jumlah sementara pada setiap iterasinya. Apabila perulangan sudah mencapai iterasi terakhir, program akan menyimpan jumlah akhir dari kumpulan angka tersebut.

Konsep ini dapat berkembang saat Anda ingin menyelesaikan permasalahan serupa, misalnya pengurangan, perkalian, pembagian, atau instruksi yang membutuhkan operasi yang sama pada setiap iterasinya. Dengan begitu, Anda dapat menyelesaikan operasi yang kompleks dalam banyaknya jumlah data.

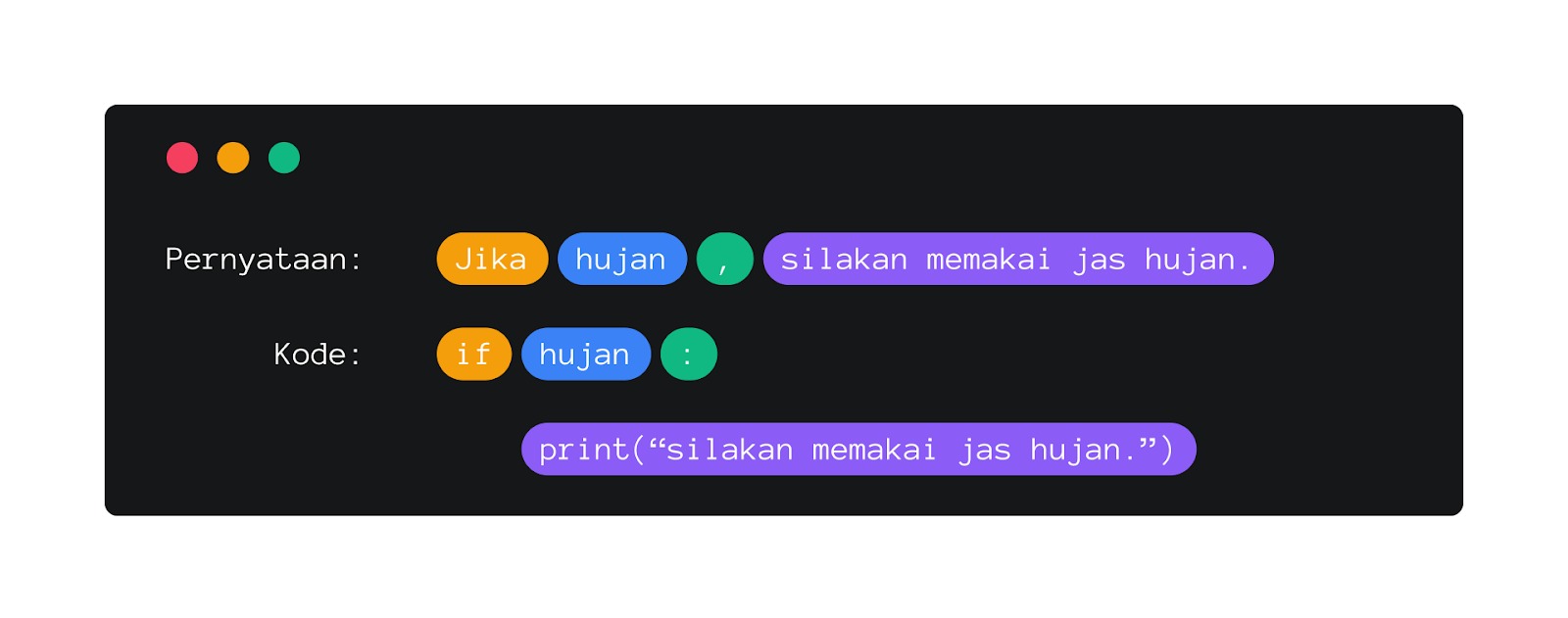
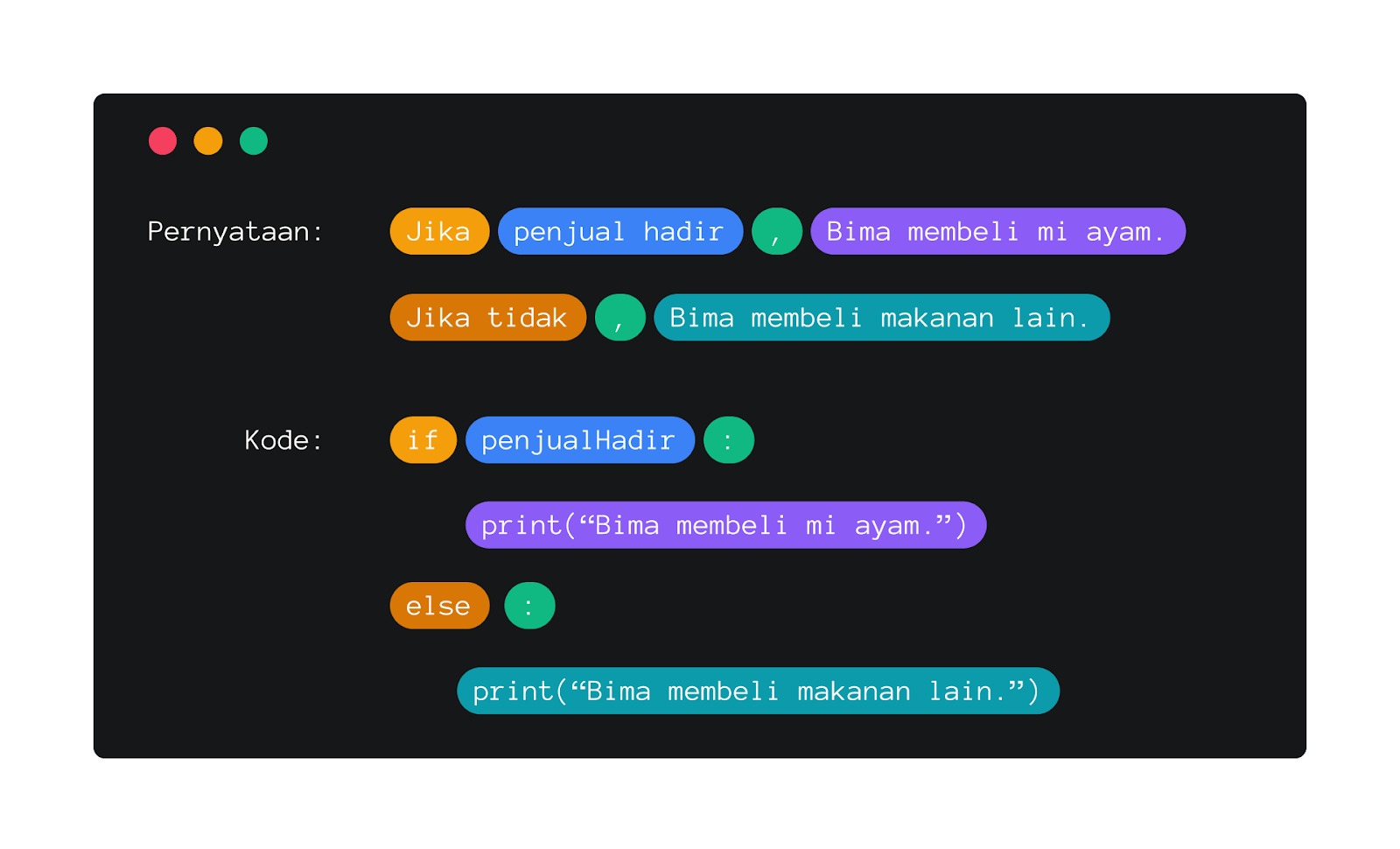
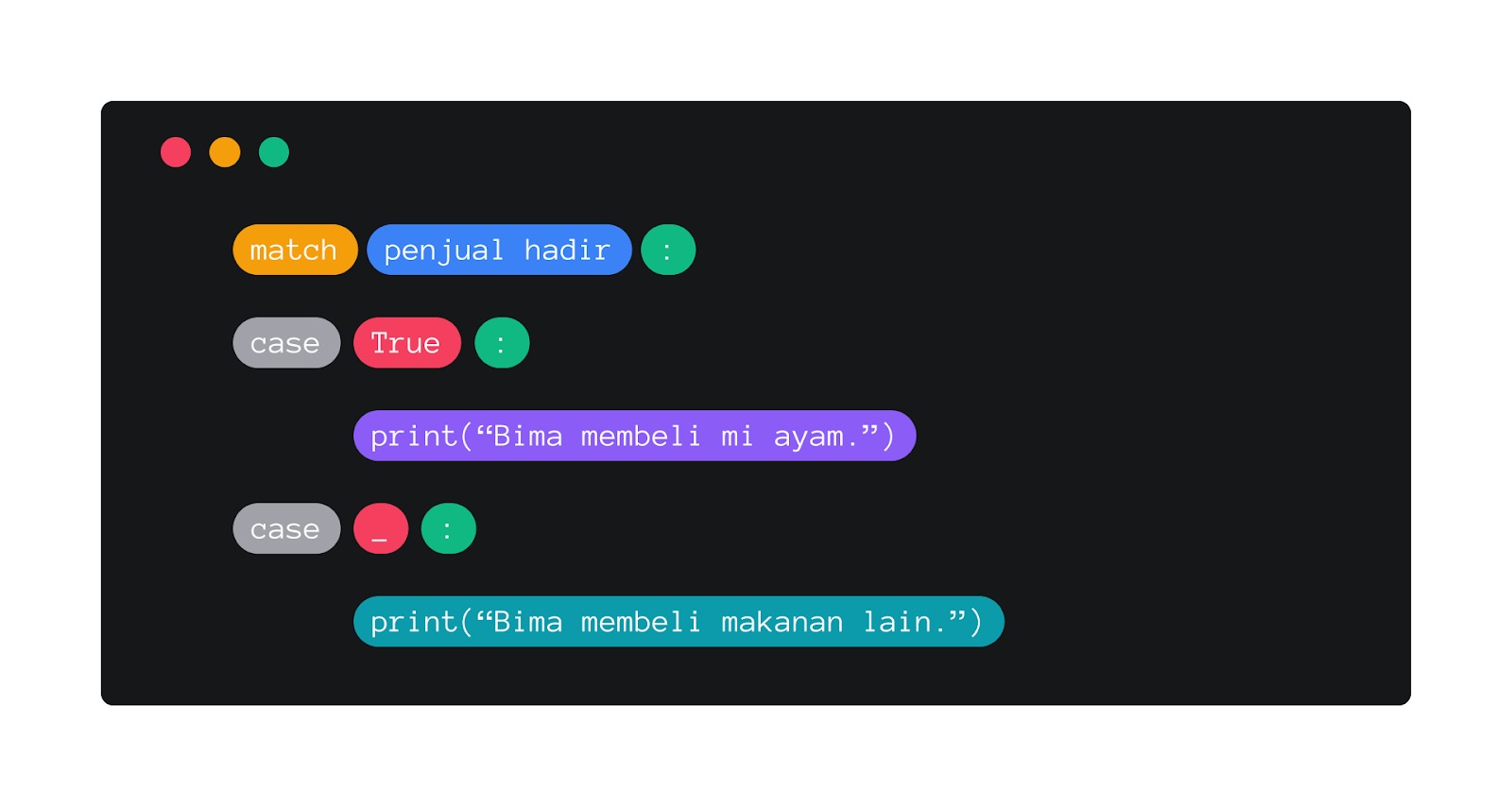
Sekian materi terkait dinamika perulangan atau permasalahan yang umum terjadi dalam proses perulangan. Apabila Anda ingin mengeksplorasi materi ini, kunjungilah beberapa tautan berikut.

* [break and continue Statements, and else Clauses on Loops](https://docs.python.org/3/tutorial/controlflow.html#break-and-continue-statements-and-else-clauses-on-loops)
* [pass Statements](https://docs.python.org/3/tutorial/controlflow.html#pass-statements)

## Rangkuman Mengatur Alur Program

Selamat! Anda sudah berada pada penghujung materi terkait mengatur alur program. Sejauh ini, Anda telah berlatih cara membuat dan memahami konsep pengondisian dan perulangan dalam suatu permasalahan. Berikut adalah ringkasan materi yang sudah Anda pelajari.

### Pengondisian

* Dalam pemrograman, ada konsep bahwa kita dapat memilih suatu opsi berdasarkan pernyataan tertentu, yaitu pengondisian.
* Suatu pernyataan dalam pengondisian dapat diartikan sebagai syarat atau kondisi yang harus terpenuhi agar suatu blok program dapat dieksekusi.
* Pengondisian memungkinkan program untuk mengevaluasi suatu kondisi dan mengambil tindakan berdasarkan evaluasi tersebut.
* Dalam proses mengevaluasi suatu kondisi, program membutuhkan ekspresi yang menghasilkan nilai boolean. Ada dua ekspresi yang mendukung hal tersebut, yaitu operator logika dan operator relasional.
* Proses evaluasi ini semacam “yes/no question”, yakni program akan menjalankan blok program tertentu ketika kondisi bernilai **True**. Apabila hasil evaluasi kondisi bernilai False, blok program akan diabaikan atau tidak dijalankan.
* Bahasa pemrograman Python memiliki pernyataan **if** untuk melakukan proses pengondisian. Cara penulisannya layaknya menulis pernyataan pengondisian pada kehidupan sehari-hari. Berikut adalah cara penulisan pengondisian if pada Python.  
  [](https://www.dicoding.com/academies/620/tutorials/35038)
  + Blok kuning adalah kata kunci if.
  + Blok biru adalah kondisi yang harus terpenuhi.
  + Blok hijau adalah karakter colon “:” sebagai pemisah pernyataan if dengan blok program.
  + Blok ungu adalah blok program yang akan dijalankan apabila kondisi bernilai True.
* Dalam konsep pengondisian, kita dapat menggunakan pernyataan **else** untuk menangani kondisi alternatif. Saat pernyataan if tidak terpenuhi atau bernilai False, blok program dalam pernyataan else akan dieksekusi sebagai jalur alternatif.
* Cara menuliskan pernyataan else mirip dengan penerapan if. Berikut adalah cara penulisan pengondisian else pada Python.  
  [](https://www.dicoding.com/academies/620/tutorials/35038)
  + Blok oranye adalah pernyataan else.
  + Blok hijau adalah karakter colon “:” sebagai pemisah pernyataan if/else dengan blok program.
  + Blok cyan adalah blok program yang akan dijalankan apabila kondisi bernilai False.
* Python memiliki pernyataan **elif** yang mampu menyediakan beberapa kondisi lebih dari satu. Kata kunci elif ini seperti else-if yang akan dievaluasi setelah pernyataan if dan sebelum else.
* Ketika pernyataan if dievaluasi dan bernilai False, kondisi dalam pernyataan elif berikutnya akan dievaluasi secara berurutan. Blok program yang akan dijalankan berdasarkan kondisi yang memenuhi syarat atau bernilai True. Jika tidak ada satu pun kondisi yang terpenuhi, blok program dalam pernyataan else akan dieksekusi.
* Ingatlah bahwa hanya satu blok program yang akan dieksekusi pada pengondisian if, if-else, atau if-elif-else. Setelah satu kondisi terpenuhi, blok program di bawahnya akan dijalankan dan program akan keluar dari pengondisian tersebut. Ia akan menghiraukan blok program lainnya.
* Pernyataan **match** merupakan pernyataan serupa dengan if untuk membandingkan nilai yang sama berkali-kali. Pernyataan ini diperkenalkan dalam Python versi 3.10 yang mampu melakukan pemilihan berdasarkan pola atau pattern matching terhadap sebuah nilai.
* Berikut adalah tatanan penulisan pernyataan match pada Python.  
  [](https://www.dicoding.com/academies/620/tutorials/35038)
  + Blok kuning adalah kata kunci match.
  + Blok abu adalah kata kunci case.
  + Blok biru adalah variabel yang ingin dibandingkan.
  + Blok hijau adalah karakter colon “:” sebagai pemisah pernyataan match/case dengan blok program.
  + Blok merah adalah pola atau nilai pembanding.
  + Blok ungu adalah blok program yang akan dijalankan apabila perbandingan antara variabel dan nilai pembanding bernilai True.
  + Blok cyan adalah blok program yang akan dijalankan apabila hasil perbandingan tidak ada yang cocok atau sama.
* Cara membandingkan dua nilai ini memiliki konsep yang serupa dengan pernyataan if. Namun, ada yang berbeda, yaitu karakter underscore ‘\_’ pada akhir case. Python menggunakan karakter tersebut untuk menunjukkan kondisi yang tidak cocok layaknya else pada pernyataan if.

### Perulangan

* Kemampuan komputer dan manusia dalam menjalankan instruksi atau aksi jelas berbeda. Komputer dapat mengulang suatu instruksi dengan performa yang sama.
* Komputer unggul pada beberapa aspek dalam melakukan aksi perulangan. Berikut adalah aspek yang membuat komputer lebih unggul dibandingkan manusia.
  + **Kecepatan dan efisiensi**: pemrosesan yang cepat dan menjalankan operasi dalam hitungan detik membuat komputer lebih baik daripada manusia dalam menjalankan aksi perulangan.
  + **Kapasitas**: dibekali penyimpanan yang lebih besar daripada manusia memungkinkan komputer mampu menyimpan dan memanipulasi data dalam jumlah besar.
  + **Ketelitian**: tingkat ketelitian yang tinggi menjadi aspek yang diunggulkan dibandingkan dengan manusia yang mengalami penurunan performa saat merasa jenuh atau lelah.
  + **Konsistensi**: saat komputer menjalankan aksi secara berulang, ia akan menghasilkan keluaran dengan durasi waktu yang seragam.
* Namun, manusia memiliki keterampilan yang tidak dimiliki komputer, yaitu mengasah kreativitas. Proses menganalisis permasalahan, memahami pola, berpikir kritis, mengidentifikasi konteks menjadi anugerah manusia untuk membuat keputusan yang kompleks.
* Dengan menggabungkan kedua kemampuan komputer dan manusia, kita dapat menciptakan tugas yang terstruktur dan dapat menjalankan tugas sebanyak mungkin.
* Perulangan adalah suatu aksi untuk mengulang serangkaian aksi dalam waktu tertentu. Kita dapat menjalankan satu blok program secara berulang kali dengan kondisi atau jumlah iterasi tertentu.
* Kita perlu menjalankan suatu aksi secara terus menerus. Hal ini menjadikan proses automasi lebih mudah dalam mengulang serangkaian blok program tanpa menuliskannya secara berkali-kali. Dengan begitu, penulisan kode menjadi semakin fleksibel dan adaptif dalam menyelesaikan masalah.
* Ada empat kunci utama dalam menjalankan perulangan.
  + **State awal**: state ini berupa variabel atau nilai tertentu yang berfungsi untuk memberi tanda memulai dan berhentinya suatu perulangan.
  + **Kondisi berhenti**: ada pernyataan pengondisian yang harus terpenuhi apabila ingin mengulangi serangkaian aksi.
  + **Badan perulangan**: serangkaian aksi yang berjalan secara berulang saat iterasi terjadi.
  + **Proses iterasi**: mekanisme perulangan yang mengombinasikan state awal, kondisi berhenti, dan badan perulangan.

### Skema Perulangan

* Ada empat macam skema perulangan yang perlu Anda ketahui. Berikut adalah daftar skema perulangan.
  + Perulangan berdasarkan jumlah iterasi.
  + Perulangan berdasarkan pencacahan.
  + Perulangan berdasarkan evaluasi kondisi berhenti.
  + Perulangan berdasarkan kompleksitas struktur.
* Dengan menerapkan skema perulangan berdasarkan**jumlah iterasi**memungkinkan program untuk menjalankan serangkaian aksi secara berulang berlandasan banyaknya iterasi.
* Perulangan yang menerapkan skema berdasarkan jumlah iterasi memiliki manfaat sebagai berikut.
  + **Ketepatan jumlah iterasi**: program akan memastikan serangkaian aksi berulang sebanyak yang diminta karena hanya berlandasan jumlah iterasi.
  + **Kemudahan implementasi**: skema ini memudahkan kita sebagai pengembang aplikasi untuk menerapkannya.
  + **Kejelasan logika program**: kemudahan dalam menerapkan skema ini menjadikan orang yang membacanya mudah memahaminya.
* Dengan menerapkan skema perulangan berlandasan **pencacahan** memungkinkan program untuk mengulang serangkaian aksi berdasarkan potongan elemen dari list, tuple, atau string.
* Perulangan yang menerapkan skema berdasarkan pencacahan memiliki manfaat sebagai berikut.
  + Mengulang setiap elemen melalui proses pencacahan kumpulan data.
  + Mengulang setiap elemen dengan tipe data yang variatif.
  + Fleksibel tanpa mementingkan pengindeksan secara manual.
* Dengan menerapkan skema perulangan berlandasan **evaluasi kondisi berhenti** memungkinkan program untuk mengevaluasi pengondisian sebelum atau sesudah menjalankan iterasi perulangan.
* Berikut adalah manfaat ketika Anda menerapkan skema perulangan berdasarkan evaluasi kondisi berhenti.
  + **Fleksibilitas**: kemudahan dalam mengulang suatu blok program membuat skema ini fleksibel dalam mengatur berhentinya proses iterasi.
  + **Menghemat sumber daya**: melalui kondisi yang tepat memungkinkan program untuk terhindar dari pemborosan sumber daya komputasi.
  + **Menyediakan kontrol alur program**: Anda dapat mengatur jalannya program dengan memberikan kontrol melalui kondisi berhenti yang terdefinisi dengan baik.
  + **Mampu menangani kasus khusus**: dengan adanya pengontrol, program dapat mengatur dan menjalankan perulangan dalam kondisi yang khusus atau berbeda dengan skema lainnya.
* Dengan menerapkan skema perulangan berlandasan kompleksitas struktur memungkinkan program memiliki perulangan dalam perulangan atau dikenal dengan istilah nested loop.

### Dinamika Perulangan

* Ada hal yang dapat terjadi ketika melakukan perulangan. Apabila ada hambatan, kondisi, atau permasalahan tertentu, program harus tetap menghasilkan efek neto positif. Oleh sebab itu, kita perlu mengetahui cara menanganinya dan tetap menghasilkan sesuatu yang kita harapkan.
* Penggunaan pernyataan **break** dalam Python mampu menghentikan perulangan supaya tidak melanjutkan ke iterasi berikutnya. Saat pernyataan break dipanggil, instruksi di bawahnya tidak akan dieksekusi.
* Penggunaan pernyataan **continue** dalam Python memungkinkan program untuk melanjutkan iterasi perulangan tanpa memedulikan perintah/instruksi di bawahnya.
* Adakalanya, kita perlu mengombinasikan variabel selain state ke dalam perulangan untuk memperluas kemampuan dan fleksibilitas algoritma yang kita buat. Dengan menggunakan variabel tambahan, kita dapat melacak, menyimpan, dan memanipulasi informasi yang diperlukan selama proses perulangan.