## Pengantar Berpikir Secara Prosedural

[](https://www.dicoding.com/academies/620/tutorials/34953?from=34948)

Tak terasa kini kita sudah memasuki pertengahan materi pada kelas ini. Pada materi sebelumnya, Anda sudah memahami konsep dasar penerapan nilai, input, output, dan ekspresi. Materi tersebut menambah wawasan kita dalam melakukan assignment, menerima dan menampilkan data, serta penerapan operator dalam suatu pengambilan keputusan atau perubahan alur program.

Pada materi ini ada yang menarik. Menyusun algoritma program, menulis dan menjalankan kode, dan menghasilkan keluaran yang diinginkan, mungkin adalah hal yang wajar dan harus dilakukan bagi seorang pengembang aplikasi. Akan tetapi, ada peraturan dan regulasi yang harus ditaati dalam menulis kode. Wah, apa itu?

Nah, saat ini kita akan belajar cara berpikir secara prosedural untuk melatih pandangan dalam menyelesaikan permasalahan secara sekuensial. Untuk itu, Anda perlu belajar beberapa materi:

* menganalisis kasus aksi sekuensial dalam kehidupan sehari-hari;
* mengamati aksi dalam suatu algoritma program; dan
* pengenalan konsep blok program dalam menulis kode.

Apakah Anda tahu cara menulis program dengan cara yang mudah? Anggaplah kita ingin mengerjakan sesuatu, seperti memakan nasi goreng. Apa yang ada dalam benak Anda ketika ingin memakan makanan tersebut? Makanan tersaji di depan Anda, ambil sendok, mulai mengambil sesendok nasi goreng dari piring, masukkan sendok ke mulut secara perlahan dan kunyah makanan tersebut.

[](https://www.dicoding.com/academies/620/tutorials/34953?from=34948)

Apakah Anda sepemikiran dengan ilustrasi di atas? Langkah demi langkah Anda lalui untuk mencapai keluaran yang diharapkan, yaitu menghilangkan rasa lapar atau menikmati nasi goreng. Lalu, sadarkah kita sudah menjalani langkah-langkah tersebut secara sekuensial? Benar sekali. Secara tidak sadar, apa pun yang ingin kita tuju selalu melewati tahapan demi tahapan.

Sama seperti pemrograman prosedural yang harus tersusun secara sekuensial. Kita harus merancang program dengan serangkaian instruksi secara berurutan dalam mencapai hasil yang diinginkan.

Nah, di sinilah tantangan bagi kita untuk mengubah cara berpikir dan menuangkannya dalam sebuah program. Langkah demi langkah yang kita bayangkan dapat ditransformasikan menjadi kode-kode. Dengan begitu, kita dapat mencapai hasil yang diharapkan dengan cara yang terstruktur dan logis.

Dari “mukadimah” atau pembukaan di atas, kita bisa menjawab pertanyaan di awal terkait cara menulis program dengan mudah. Caranya, cobalah berpikir secara terstruktur dan logis untuk menggapai keluaran yang diinginkan. Setelah bisa mengaitkan satu aksi dengan aksi berikutnya, kita bisa mentransformasikannya menjadi kode program yang dapat dipahami oleh komputer.

Sudah terbayang apa yang akan dibahas pada materi kali ini? Kita akan belajar aksi sekuensial lebih mendalam terkait mengubah cara berpikir dan menuangkannya ke dalam kode program. Penasaran? Yuk, silakan lanjut ke materi berikutnya.

## Kasus Aksi Sekuensial

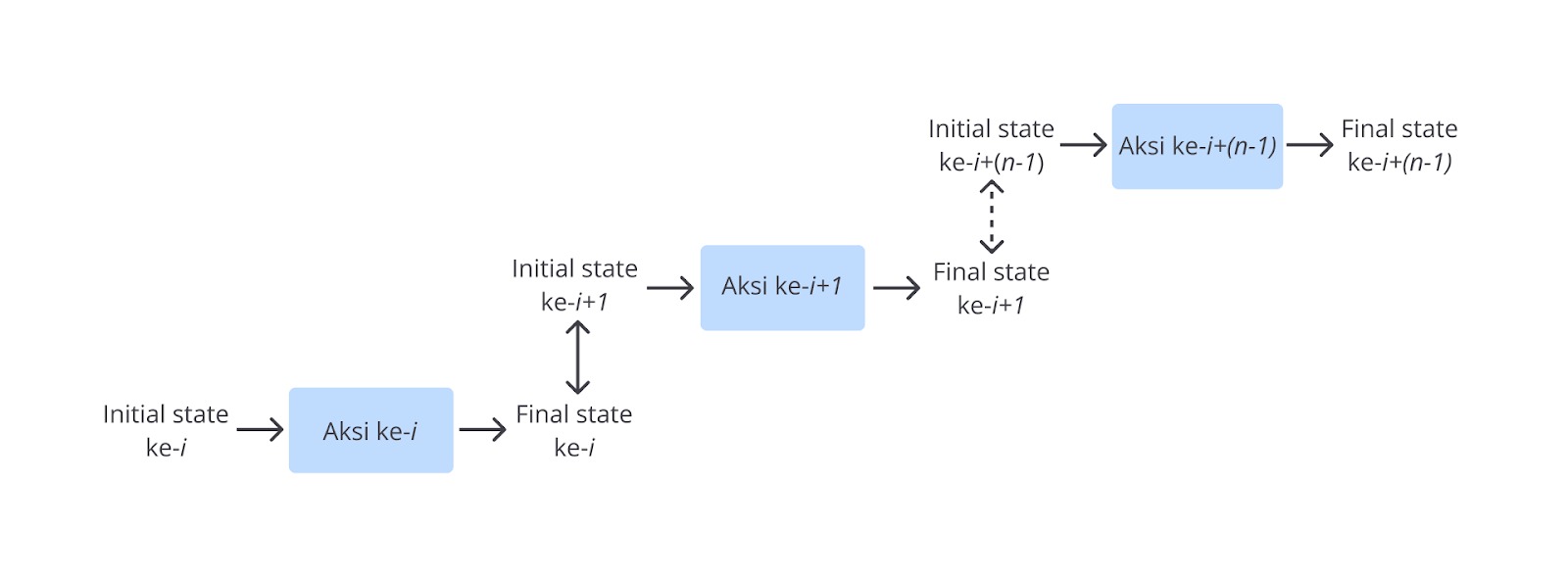
Tahukah Anda maksud dari aksi sekuensial? Aksi sekuensial atau sequential statement adalah deretan instruksi primitif atau tindakan yang dilakukan berdasarkan urutan penulisannya. Misalnya, kita mempunyai lima tahapan yang harus diselesaikan. Tahapan tersebut harus diselesaikan satu demi satu dan berurutan.

Dengan kata lain, aksi sekuensial dapat diartikan sebagai deretan instruksi yang terdiri dari tahapan 1, 2, 3, 4, hingga n. Nilai n ini dapat didefinisikan berapa pun asalkan selalu bernilai bilangan bulat positif lebih dari 0. Apakah satu tahapan bisa dikatakan sebagai aksi sekuensial? Yup, aksi sekuensial sederhana juga dapat mengandung satu instruksi saja.

Ibarat Bima ingin naik ke lantai 4, pasti memiliki anak tangga yang harus dilalui secara bertahap. Dimulai dari tangga dari lantai ke-1, kemudian ke-2, dan seterusnya hingga lanjut ke-4. Bima tidak mungkin tiba-tiba lompat dari lantai 1 ke 4, bukan? Tujuannya apa? Tentu saja untuk tiba di lantai 4 tanpa hambatan.

Ingat, setiap tahapan aksi atau instruksi memiliki keterkaitan satu sama lain. Bagaimana maksudnya? Sebagaimana penjelasan sebelumnya, ketika menaiki tangga ke lantai 2, kita harus menaiki tangga dari lantai 1 untuk mencapainya. Artinya, setiap tahapan memiliki state atau status yang harus dipenuhi sebelum lanjut ke tahapan berikutnya.

Dalam aksi sekuensial, ada dua state yang saling terhubung antara aksi satu dan lainnya, yaitu **initial state** dan **final state**. Initial state adalah status aksi yang harus dipenuhi, sedangkan final state adalah status setelah instruksi atau aksi terlaksana. Pada instruksi atau aksi ke-i, final state akan menjadi initial state untuk i+1. Dengan kata lain, setiap aksi akan saling terhubung dan tersusun dengan baik [1].

[](https://www.dicoding.com/academies/620/tutorials/34958?from=34953)

Supaya lebih familier dengan aksi sekuensial, mari kita simak beberapa sketsa berikut.

### Perspektif Initial dan Final State

Saat Ibu Fika memasak sup untuk seluruh anggota keluarganya, beliau melakukan beberapa runtutan aksi.

|  |
| --- |
| 1. Mencuci sayuran. 2. Memotong sayur menjadi bagian kecil-kecil. 3. Memasak kaldu dalam panci. 4. Memasukkan sayur yang telah terpotong ke dalam panci. 5. Menyajikan sup dalam mangkuk. |

Pada aksi tersebut, kejadian dijelaskan dalam lima sub-aksi yang berurutan.

Jika flashback pada materi sebelumnya, setiap aksi memiliki batasan yang mengikat. Masih ingatkah Anda dengan hal itu? Ya, setiap aksi memiliki kondisi yang mengikat setidaknya berupa durasi waktu. Selain itu, ada kondisi atau batasan lain yang dapat menimbulkan pertanyaan. Hal ini yang menyebabkan aksi tidak bisa berjalan. Salah satu contohnya adalah pertanyaan “dari mana Ibu mendapatkan sayuran, bahan masakan, dan peralatan dapur lainnya?”

Itulah adanya **initial state** yang harus terpenuhi pada tahapan aksi pertama. Setiap menjalankan aksi atau sub-aksi tertentu, pastikan seluruh initial state terpenuhi dengan baik. Apabila tidak terpenuhi, aksi tersebut tidak akan bisa berjalan dengan semestinya.

Setelah sub-aksi pertama (mencuci sayuran) terlaksana, muncullah **final state** dengan status bahwa sayuran telah bersih. Saat melangkah ke sub-aksi kedua, **final state** tadi berubah menjadi **initial state**. State inilah yang perlu dicek pada tahapan kedua. Hal ini terus berlangsung hingga tahapan terakhir terlaksana dengan baik sehingga menghasilkan **final state** akhir yang bisa kita sebut dengan hasil akhir.

Lalu, apa yang terjadi ketika initial state tidak memenuhi kriteria pada suatu tahapan tertentu? Bayangkan ketika Ibu Fika melakukan tahapan “mencuci sayuran”, tetapi tidak ada sayur yang tersedia. Mulai dari tahapan pertama pun tidak bisa dijalankan. Alhasil, aksi “memasak sup” gagal dilakukan dengan baik. Hal ini berlaku ketika initial state didapatkan dari final state pada tahapan sebelumnya.

### Pengamatan Aksi

Ketika initial state pada sub-aksi tidak terpenuhi, tahapan tersebut tidak bisa terselesaikan dengan baik. Namun, kita bisa memanipulasi kondisi tersebut supaya sub-aksi tetap bisa berjalan. Bagaimana caranya? Alih-alih memaksa sub-aksi berjalan tanpa initial state, kita dapat menambahkan sub-aksi baru. Cara ini memungkinkan kita untuk melanjutkan keseluruhan aksi tanpa adanya kegagalan.

Misalnya, sayuran dan talenan belum tersedia pada area dapur. Untuk melangkah ke sub-aksi pertama (mencuci sayuran), kita bisa menambahkan sub-aksi baru sehingga susunan aksi menjadi seperti berikut.

|  |
| --- |
| 1. Mengambil sayuran di rak. 2. Mengambil talenan di rak. 3. Mencuci sayuran. 4. Memotong sayur menjadi bagian kecil-kecil. 5. Memasak kaldu dalam panci. 6. Memasukkan sayur yang telah terpotong ke dalam panci. 7. Menyajikan sup dalam mangkuk. |

Jadi, initial state saat tahapan “mencuci sayuran” akan terpenuhi dengan baik.

Adakalanya, suatu tahapan sub-aksi tidak terikat satu sama lain. Bagaimana maksudnya? Ketika sub-aksi tersebut berjalan dan menghasilkan final state, state tersebut tidak dibutuhkan oleh sub-aksi berikutnya.

Contohnya, tahapan pertama dan kedua yang sama-sama mengambil benda dari tempat penyimpanan. Ketika sub-aksi pertama dijalankan, tidak ada kaitannya dengan sub-aksi kedua.

Jika diamati secara saksama, kedua sub-aksi tersebut dibutuhkan, tetapi susunan letaknya tidak berpengaruh. Perubahan susunan tahapan satu dan dua tidak akan memengaruhi jalannya keseluruhan aksi. Dengan kata lain, kita bisa mengganti kedua tahapan tersebut menjadi seperti berikut.

|  |
| --- |
| 1. Mengambil talenan di rak. 2. Mengambil sayuran di rak. 3. Mencuci sayuran. 4. Memotong sayur menjadi bagian kecil-kecil. 5. Memasak kaldu dalam panci. 6. Memasukkan sayur yang telah terpotong ke dalam panci. 7. Menyajikan sup dalam mangkuk. |

Mengapa bisa demikian? Alasannya adalah kedua tahapan tersebut **tidak dipandang perlu berurutan** satu sama lain. Jika melihat sudut pandang Ibu Fika sebagai pelaku aksi dan seorang yang sedang multitasking, kedua tahapan tersebut dapat disatukan menjadi seperti berikut.

|  |
| --- |
| 1. Mengambil talenan di rak dan mengambil sayuran di rak. 2. Mencuci sayuran. 3. Memotong sayur menjadi bagian kecil-kecil. 4. Memasak kaldu dalam panci. 5. Memasukkan sayur yang telah terpotong ke dalam panci. 6. Menyajikan sup dalam mangkuk. |

Ya, saat ada sekumpulan tahapan yang sekiranya tidak perlu disusun secara sekuensial, kita bisa mengelompokkannya menjadi satu tahapan saja. Hal ini juga dilihat dari segi initial dan final state dari sekumpulan tahapan tersebut. Apabila state tidak berimbas pada tahapan lainnya, tahapan tersebut dapat dikumpulkan menjadi satu.

### Aksi Bersyarat

Terkadang, serangkaian aksi memasak sup tersebut tidak dilakukan oleh Ibu Fika. Serangkaian aksi tersebut dapat dikerjakan oleh orang lain ketika Ibu Fika sedang memiliki pesanan katering yang cukup banyak. Alhasil, Ibu Fika meminta asisten untuk membantu pekerjaannya.

Sebelum memulai, asisten diajarkan cara untuk melaksanakan serangkaian aksi dari awal hingga akhir oleh Ibu Fika. Namun, asisten belum cukup terampil dalam mengerjakan aksi tersebut. Banyak percikan atau kotoran bekas pekerjaannya menempel pada pakaiannya. Ibu Fika memberikan inisiatif kepada asisten untuk menggunakan celemek.

Dari penggalan cerita di atas, terdapat sebuah kondisi yang mengharuskan asisten memakai celemek saat melakukan serangkaian aksi memasak sup. Dengan begitu, serangkaian aksi berubah menjadi seperti berikut.

|  |
| --- |
| 1. Mengambil talenan di rak dan mengambil sayuran di rak. 2. Memakai celemek. 3. Mencuci sayuran. 4. Memotong sayur menjadi bagian kecil-kecil. 5. Memasak kaldu dalam panci. 6. Memasukkan sayur yang telah terpotong ke dalam panci. 7. Menyajikan sup dalam mangkuk. |

Kita tidak bisa menormalisasi serangkaian aksi di atas. Mengapa tidak? Ada beberapa orang yang sudah terbiasa dengan melaksanakan serangkaian aksi tanpa perlu menggunakan celemek. Menurut pengamatan orang tersebut, dia tidak akan meninggalkan bekas kotor pada pakaiannya sehingga celemek tidak diperlukan. Jadi, tahapan “memakai celemek” dianggap opsional.

|  |
| --- |
| 1. Mengambil talenan di rak dan mengambil sayuran di rak. 2. Jika sering meninggalkan bekas kotor saat memasak, pakailah celemek. 3. Mencuci sayuran. 4. Memotong sayur menjadi bagian kecil-kecil. 5. Memasak kaldu dalam panci. 6. Memasukkan sayur yang telah terpotong ke dalam panci. 7. Menyajikan sup dalam mangkuk. |

Tahapan opsional di atas juga dapat berkontribusi untuk menghasilkan keluaran yang diharapkan, yaitu sup. Namun, cara atau tahapan tersebut berbeda. Tahapan kedua memiliki aksi bersyarat dengan mengamati kondisi pelaku dan terdapat pernyataan yang menghasilkan suatu keadaan benar true) atau salah (false).

Apabila pernyataan “sering meninggalkan bekas kotor saat memasak” bernilai benar, pelaku akan menggunakan celemek untuk menghindari kotor. Jika yang terjadi sebaliknya, pelaku tidak memakai celemek.

### Pengulangan Aksi

Ketika Ibu Fika mendapatkan pesanan katering yang cukup banyak, yaitu 100 porsi, beliau pasti membeli bahan masakan yang banyak pula. Persediaan untuk satu keluarga saja tidak mungkin cukup untuk membuat sup dalam jumlah yang begitu banyak. Akhirnya, Ibu Fika menyediakan 100 porsi dengan jumlah dan takaran yang sudah diperhitungkan.

Saat memulai memasak sup, apakah Ibu Fika mengulangi serangkaian aksi yang sudah kita definisikan di awal? Bayangkan bila Ibu Fika memasak 1 porsi sup seperti cara sebelumnya, tetapi mengulangi aksi dari awal sampai akhir sebanyak 100 kali. Jelas itu akan membuat Ibu Fika kewalahan dan tentunya menghabiskan banyak waktu, biaya, dan tenaga.

Ada beberapa sub-aksi yang tidak perlu dilakukan secara berulang dan ada juga sub-aksi yang perlu diulang berkali-kali. Hal ini bergantung pada tujuan atau keluaran yang ingin kita capai. Kita perlu mengamati kembali susunan dalam aksi yang berulang dan tidak perlu berulang.

Mari kita cermati serangkaian aksi yang sudah didefinisikan. Menurut Anda, tahapan dalam aksi mana yang perlu diulang? Apakah tahapan “memakai celemek” perlu dilakukan pengulangan? Jelas tidak. Inilah salah satu pengamatan yang harus dikerjakan sebelum melakukan serangkaian aksi. Jika tahapan “mencuci sayuran” hingga akhir perlu dilakukan secara berulang, Anda menjawab dengan benar.

|  |
| --- |
| 1. Mengambil talenan di rak dan mengambil sayuran di rak. 2. Jika sering meninggalkan bekas kotor saat memasak, pakailah celemek. 3. Lakukan sub-aksi berikut hingga selesai sebanyak 100 kali.    1. Mencuci sayuran.    2. Memotong sayur menjadi bagian kecil-kecil.    3. Memasak kaldu dalam panci.    4. Memasukkan sayur yang telah terpotong ke dalam panci.    5. Menyajikan sup dalam mangkuk. |

Menurut Anda, apakah konsep mengulang seperti di atas sudah sesuai? Perlu diingat kembali bahwa yang mengerjakan serangkaian aksi ini hanyalah seorang saja. Apabila melakukan pengulangan sub-aksi seperti di atas, proses menghasilkan sup tidak akan efisien.

Mengapa demikian? Ada beberapa tahapan yang harus dikerjakan sebanyak 100 kali dahulu, kemudian lanjut ke tahapan berikutnya, supaya lebih efisien. Alhasil, serangkaian aksi menjadi seperti berikut.

**Catatan**:  
Serangkaian aksi ini dilakukan menggunakan panci dengan ukuran yang kecil dan cukup untuk satu porsi, ya. Jadi, Ibu Fika membutuhkan 100 kali pengulangan untuk membuat 100 porsi sup. Kondisi ini dibuat untuk memudahkan pembelajaran kita mengenai pengulangan aksi.

|  |
| --- |
| 1. Mengambil talenan di rak dan mengambil sayuran di rak. 2. Jika sering meninggalkan bekas kotor saat memasak, pakailah celemek. 3. Lakukan sub-aksi berikut hingga selesai sebanyak 100 kali.    1. Mencuci sayuran. 4. Lakukan sub-aksi berikut hingga selesai sebanyak 100 kali.    1. Memotong sayur menjadi bagian kecil-kecil. 5. Lakukan sub-aksi berikut hingga selesai sebanyak 100 kali.    1. Memasak kaldu dalam panci.    2. Memasukkan sayur yang telah terpotong ke dalam panci.    3. Menyajikan sup dalam mangkuk. |

Secara jumlah, sub-aksi memang menjadi lebih banyak karena ada tiga kali pengulangan. Akan tetapi, pelaku akan lebih terkoordinasi dalam melaksanakan serangkaian aksi. Tidak sembarangan menempatkan suatu pengulangan pada suatu sub-aksi. Perlu adanya pengamatan yang tepat untuk mendapatkan hasil yang optimal.

### Pengelompokan Aksi-Aksi

Setelah mencoba mengamati suatu sub-aksi yang akan dikerjakan, kita bisa menelaah bahwa ada kumpulan sub-aksi yang dapat dijadikan satu kelompok. Kelompok ini harus memiliki tujuan yang serupa. Mengapa demikian? Hal ini dapat dilakukan supaya meningkatkan struktur dan kejelasan suatu sub-aksi serta mempermudah pemahaman bagi pelakunya.

Lalu, bagaimana kita mengelompokkan kumpulan sub-aksi yang memiliki satu tujuan? Cara paling mudah adalah menjadikannya satu fungsi atau prosedur. Dengan mengelompokkan sub-aksi yang serupa, kita dapat memisahkan bagian-bagian aksi terkait secara terpisah.

Berdasarkan serangkaian aksi memasak sup, tahapan apa yang bisa dikelompokkan menjadi satu grup? Ingat, kumpulan sub-aksi yang memiliki satu tujuan dapat dijadikan satu kelompok.

Secara umum, serangkaian aksi memasak sup dapat dikelompokkan dan menghasilkan dua grup tahapan aksi, yaitu “persiapan memasak” serta “memasak dan menyajikan makanan”. Jadi, serangkaian aksi menjadi seperti berikut.

|  |
| --- |
| 1. Menyiapkan bahan dan peralatan memasak\* 2. Lakukan sub-aksi berikut hingga selesai sebanyak 100 kali.    1. Mencuci sayuran. 3. Lakukan sub-aksi berikut hingga selesai sebanyak 100 kali.    1. Memotong sayur menjadi bagian kecil-kecil. 4. Lakukan sub-aksi berikut hingga selesai sebanyak 100 kali.    1. Memasak dan menyajikan makanan\*\*   Di mana:  \*) Menyiapkan bahan dan peralatan memasak berisi:   * Mengambil talenan di rak dan mengambil sayuran di rak. * Jika sering meninggalkan bekas kotor saat memasak, pakailah celemek.   \*\*) Memasak dan menyajikan makanan berisi:   * Memasak kaldu dalam panci. * Memasukkan sayur yang telah terpotong ke dalam panci. * Menyajikan sup dalam mangkuk. |

Tahapan mengambil bahan masak dan memakai peralatan dapat kita satukan menjadi kelompok yang bernama “menyiapkan bahan dan peralatan memasak”. Kedua tahapan tersebut memiliki tujuan yang sama, yaitu mempersiapkan bahan dan alat memasak.

Berikutnya adalah prosesi memasak kaldu, memasukkan sayur, dan menyajikan makanan dapat dikelompokkan menjadi satu tahapan aksi bernama “memasak dan menyajikan makanan”. Kumpulan sub-aksi tersebut merepresentasikan tujuan yang sama, yaitu memasak dan menyajikan makanan.

Melihat serangkaian aksi di atas cukup panjang dengan penambahan fungsi baru. Akan tetapi, dari segi keterbacaan pelaku, aksi terlihat mudah karena menjadi lebih singkat dibandingkan sebelumnya. Ada bagian-bagian kecil yang terorganisasi dengan baik dan terpisah satu sama lain. Hal ini memudahkan kita memahami setiap aksi yang dilakukan serta memberikan kemudahan jika ingin melakukan perubahan atau mengerjakan ulang pada bagian tertentu.

Itulah beberapa konsep dasar yang perlu Anda pelajari dalam menjalankan aksi sekuensial. Dengan berbagai sketsa dan kasus yang berbeda-beda, membuat kita semakin memahami pendekatan yang harus dilalui untuk mencapai hasil yang diinginkan.

Hal ini berkaitan dengan cara kita mengamati suatu permasalahan. Perlu adanya proses menganalisis agar jalur yang kita pilih sesuai dengan harapan. Sudahkah kita memahami konsep initial dan final state dalam suatu program? Apakah kita menelaah penyelesaian ini dengan benar?

Seluruh jawaban itu dapat dilatih secara bertahap. Memang butuh waktu untuk mencapai hasil yang diinginkan. Tidak mudah, tetapi tidak mustahil untuk diraih. Mari belajar bersama-sama supaya mampu menciptakan konsep berpikir yang selaras layaknya pemrograman prosedural.

Apabila Anda masih belum paham terkait materi ini, silakan bertanya di Forum Diskusi, ya.

## Observasi Aksi

Dalam suatu algoritma program, kita perlu mengamati kelayakan setiap instruksi yang dibuat. Proses mengidentifikasi aksi menjadi cara kita untuk menganalisis serangkaian perintah yang terjadi saat program berjalan.

Misalnya, kita telah menuliskan program dengan algoritma yang runtut. Dengan memberikan pemisah berupa new line atau paragraf baru, kita dapat memahami dan memudahkan orang lain dalam mengamati setiap instruksi yang sudah dibuat. Namun, adakalanya kita dapat membuat instruksi tersebut lebih ringkas atau sederhana.

Lalu, bagaimana caranya kita meringkas instruksi atau perintah yang sudah kita buat? Apakah tujuan memberi “new line” akan bertolak belakang dengan meringkas suatu instruksi? Mari kita perhatikan perintah memasak sup berikut.

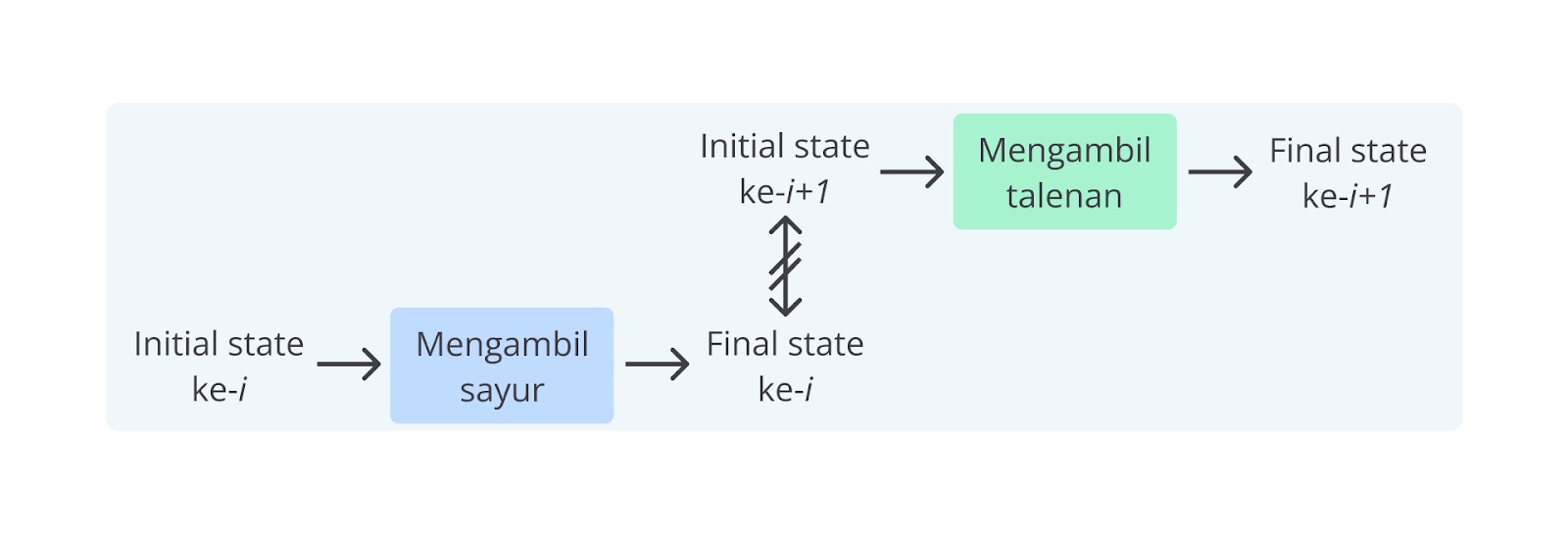
|  |
| --- |
| 1. Mengambil sayuran di rak. 2. Mengambil talenan di rak. 3. Mencuci sayuran. 4. Memotong sayur menjadi bagian kecil-kecil. 5. Memasak kaldu dalam panci. 6. Memasukkan sayur yang telah terpotong ke dalam panci. 7. Menyajikan sup dalam mangkuk. |

Kita sudah mendefinisikan suatu susunan perintah dengan terstruktur dan benar. Namun, coba perhatikan baik-baik, ada perintah yang tidak terikat dengan perintah sebelumnya atau setelahnya. Perintah “mengambil sayuran” dan “mengambil talenan” adalah contoh perintah yang tidak terikat satu sama lain.

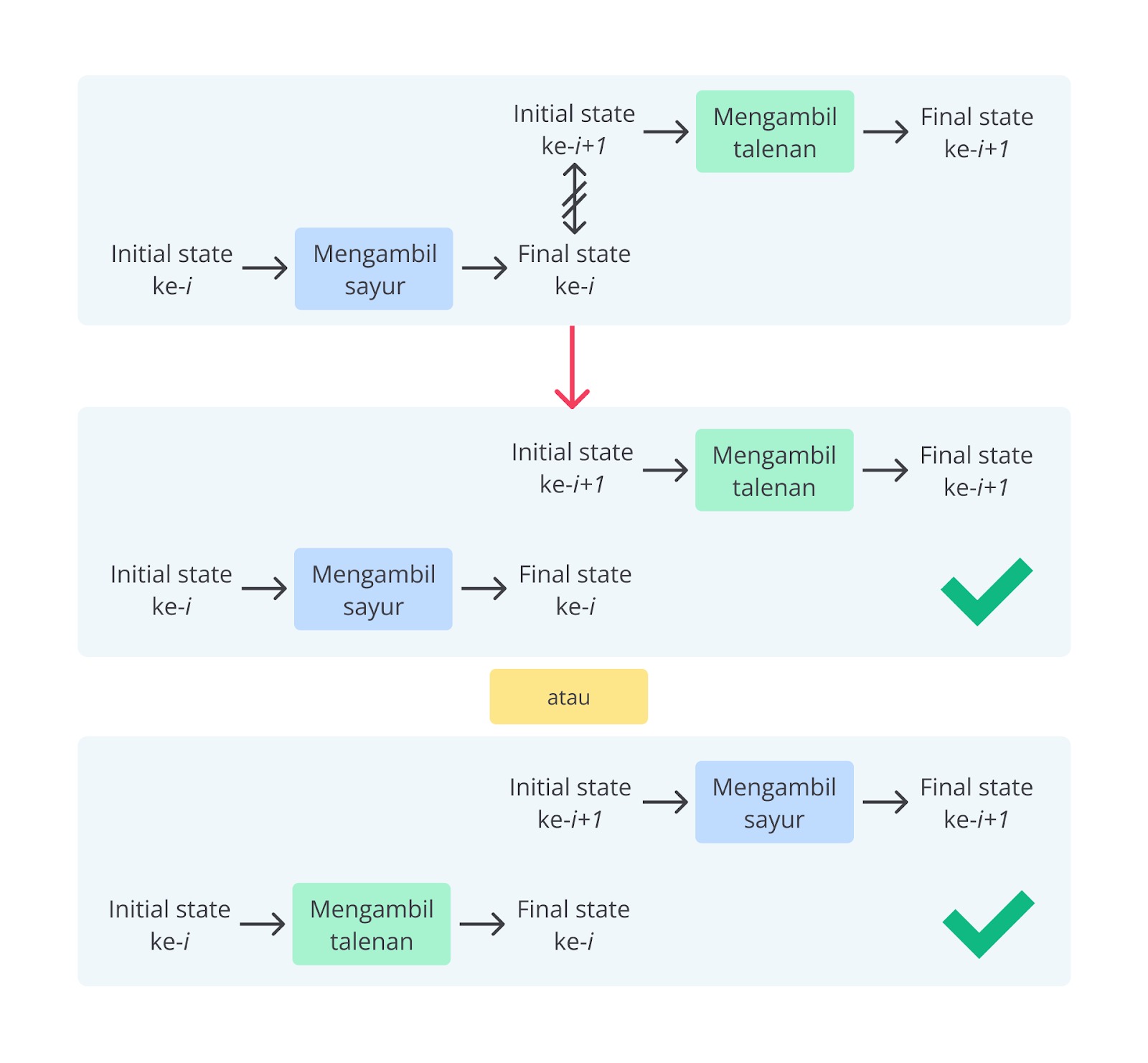
Apabila menemukan perintah dengan kondisi seperti tersebut, kita dapat melakukan dua hal.

1. Kita dapat mengubah posisi instruksi secara acak, perintah “mengambil sayuran” atau “mengambil talenan” terlebih dahulu.
2. Kita dapat menggabungkan kedua instruksi tersebut supaya berjalan secara bersamaan.

Kita akan bahas satu per satu terkait kondisi di atas. Pertama, kita bisa mengubah posisi instruksi secara acak. Mengapa demikian? Kedua instruksi tersebut memiliki initial dan final state yang tidak beririsan. Pada umumnya, final state akan menjadi initial state pada instruksi berikutnya (cek kembali pada materi “Kasus Aksi Sekuensial”). Akan tetapi, kondisi ini berbeda. Ia tidak terikat satu dengan lain.

[](https://www.dicoding.com/academies/620/tutorials/34963?from=34958)

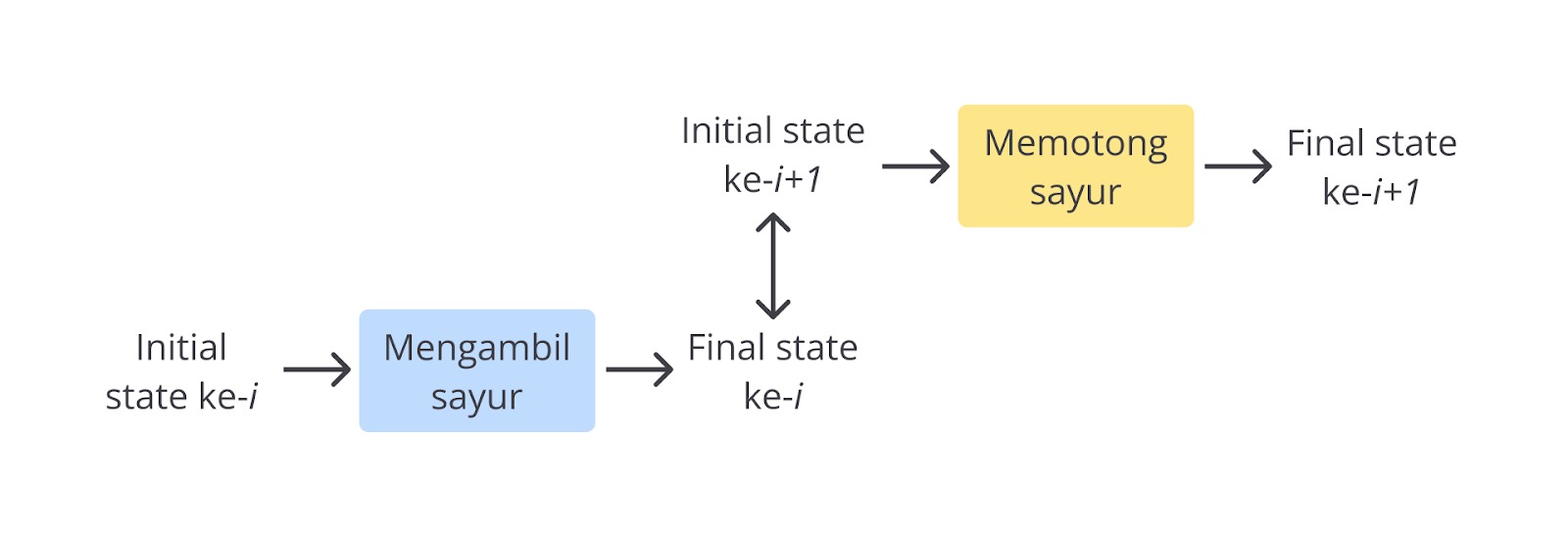
Alhasil, setiap instruksi atau aksi dapat berjalan secara terpisah. Dengan kata lain, kita bisa memanipulasi susunan aksi untuk meletakkannya menjadi posisi pertama atau kedua.

[](https://www.dicoding.com/academies/620/tutorials/34963?from=34958)

Pada bahasa pemrograman Python, kita bisa mengibaratkan kedua perintah ini adalah proses inisialisasi variabel. Kedua variabel tersebut pastinya tidak saling terikat satu sama lain. Dengan cara yang sama, kita bisa mengubah susunan instruksi menjadi urutan pertama atau kedua.

1. getVegetablesDone = True
2. getCuttingBoardDone = True
4. # atau
5. getCuttingBoardDone = True
6. getVegetablesDone = True

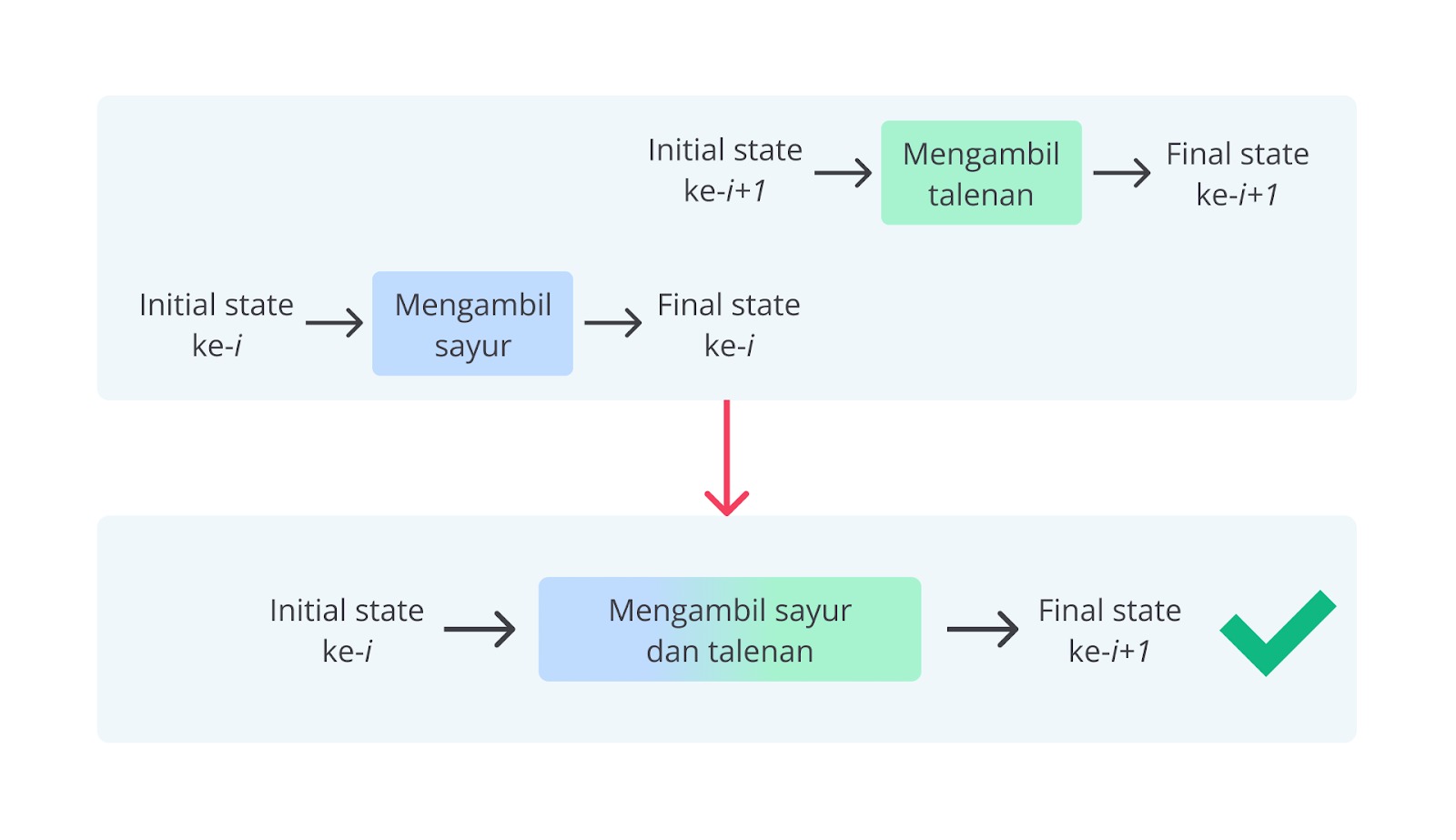
Berbeda cerita ketika melaksanakan suatu instruksi yang terikat dengan instruksi sebelumnya, misalnya perintah “memotong sayur”. Perintah tersebut harus memiliki initial state yang berupa “sayuran telah tersedia dan siap untuk dipotong”. Apabila sayur tidak tersedia, perintah “memotong sayur” tidak dapat terlaksana dengan baik. Maka dari itu, kedua perintah tersebut saling terikat dan haruslah berurutan.

[](https://www.dicoding.com/academies/620/tutorials/34963?from=34958)

Sama halnya dengan bahasa pemrograman Python. Jika ada perintah yang terikat dengan perintah sebelumnya, urutan perintah tersebut tidak dapat bertukar. **Final state** dari perintah ke-i harus didapatkan terlebih dahulu untuk menjadi **initial state** pada perintah berikutnya.

1. getVegetablesDone = True
2. cutVegetables = getVegetablesDone and True
4. # tidak boleh menjadi seperti berikut
5. cutVegetables = getVegetablesDone and True
6. getVegetablesDone = True

Selain itu, kita juga bisa memanfaatkan opsi kedua, yaitu menggabungkan dua instruksi menjadi satu dan dijalankan secara bersamaan. Dua perintah yang tidak saling terhubung dapat digabung menjadi satu perintah saja.

[](https://www.dicoding.com/academies/620/tutorials/34963?from=34958)

Hal yang sama dapat terjadi pada bahasa pemrograman Python. Ketika mempunyai dua perintah yang tidak terikat satu sama lain, kita bisa menggabungkannya menjadi satu baris perintah saja.

Konsepnya sedikit berbeda dari yang biasanya satu baris memiliki satu perintah saja. Dalam Python, kita bisa menjalankan dua perintah dalam satu baris yang dipisahkan dengan karakter semicolon “;”. Berikut contoh penerapannya.

1. getVegetablesDone = True
2. getCuttingBoardDone = True
4. # menjadi
5. getVegetablesDone = True; getCuttingBoardDone = True

Dari penampilan atau penulisan kode tidaklah berubah. Pembedanya adalah jumlah baris dan setiap instruksi dipisahkan oleh karakter.

Namun, ada konsep lain untuk Anda membuat kode dengan semakin mudah dan ringkas. Konsep tersebut dinamakan one-liner, yaitu menulis suatu kode atau fungsi dalam satu baris yang singkat dan padat. Konsep ini dapat diterapkan pada bahasa pemrograman Python, seperti mengetahui kata palindrom, menukar dua variabel, menjumlahkan seluruh nilai pada suatu list, dll.

1. getVegetablesDone = True
2. getCuttingBoardDone = True
4. # menjadi
5. getVegetablesDone, getCuttingBoardDone = True, True

Konsep one-liner ini memiliki kelebihan, yaitu membantu meningkatkan keterbacaan kode dan mempercepat penulisan kode supaya lebih ringkas. Di balik itu, Anda perlu memahami seluruh instruksi yang diberikan walaupun menerapkan one-liner.

Terkadang, perintah yang kompleks dalam suatu perintah one-liner memengaruhi keterbacaan dan pemahaman kode. Oleh karena itu, penting bagi Anda untuk mempertimbangkan kejelasan dan kegunaan kode dalam menerapkan one-liner atau menyusunnya menjadi beberapa baris yang terstruktur.

Materinya super sekali, bukan? Dalam menulis kode, kita perlu menganalisis kembali perintah yang sudah dibuat supaya kode memiliki keterbacaan dan pemahaman yang baik. Dengan begitu, penulis kode ataupun orang lain yang membaca perintah tersebut mengerti maksud dari perintah tersebut. Apabila ingin menambah wawasan terkait materi ini, silakan kunjungi tautan berikut.

* [Powerful Python One-Liners](https://wiki.python.org/moin/Powerful%20Python%20One-Liners)

## Blok Program

Halo, masih semangatkah untuk menerima materi kali ini? Materi ini masih berkaitan dengan materi aksi sekuensial dan pengamatan suatu aksi. Jadi, tidak perlu berpusing-pusing lagi, kita simak materi terkait blok program.

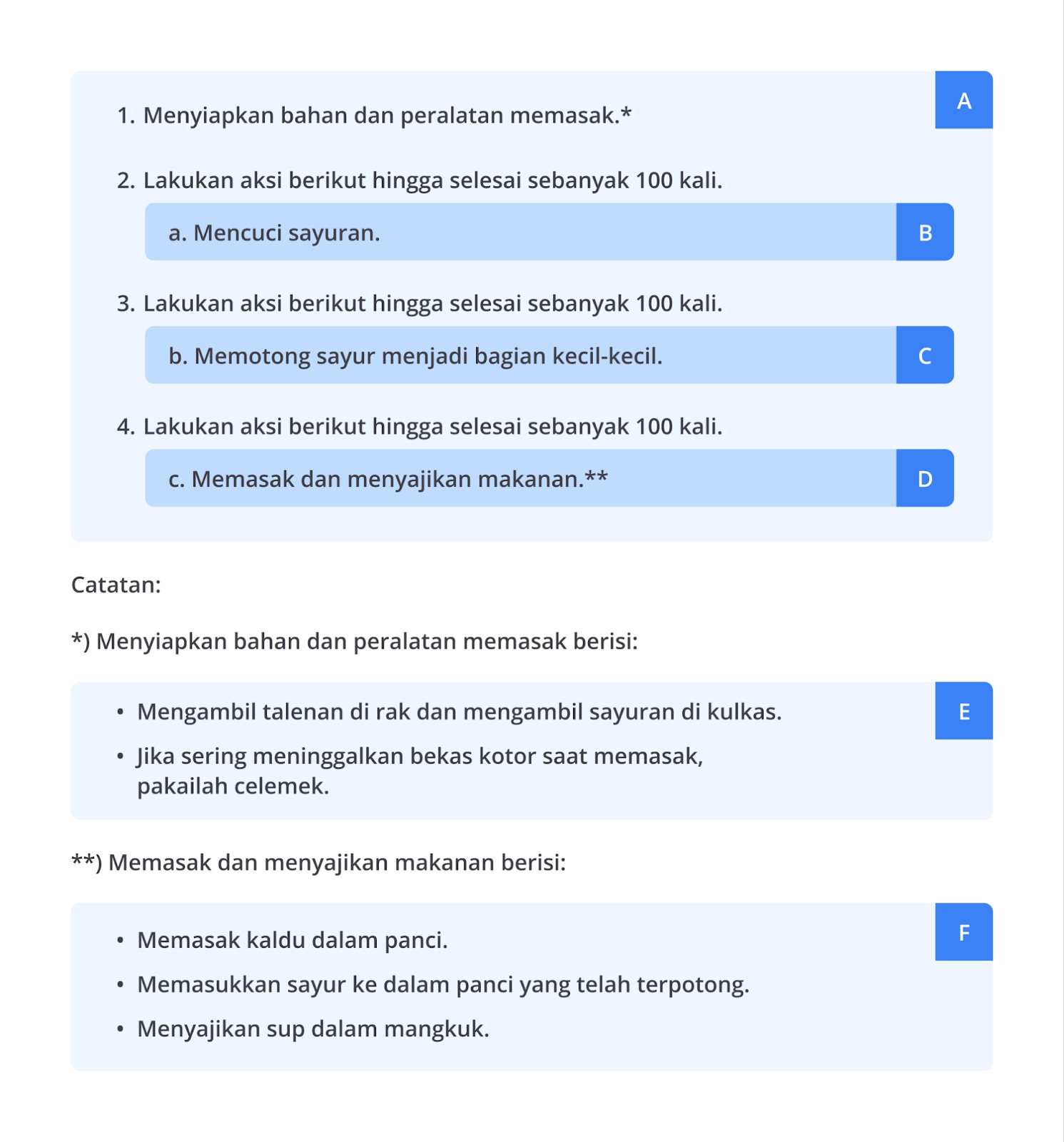
Tahukah Anda, arti dari blok program? Menurut Cambridge Dictionary, [block](https://dictionary.cambridge.org/dictionary/english/block) atau blok memiliki arti sekelompok hal yang dibeli, ditangani, atau dipertimbangkan bersama. Lain halnya dengan [program](https://dictionary.cambridge.org/dictionary/english/program), artinya adalah sekelompok kegiatan atau hal-hal yang ingin dicapai. Apabila disatukan, arti kata blok program adalah kumpulan instruksi yang berjalan pada satu unit yang sama untuk mencapai tujuan tertentu.

Seharusnya, Anda sudah bisa membayangkan arti dari blok program menilik dari materi sebelumnya. Coba kita mengulik kembali serangkaian aksi memasak sup.

|  |
| --- |
| 1. Menyiapkan bahan dan peralatan memasak\* 2. Lakukan sub-aksi berikut hingga selesai sebanyak 100 kali.    1. Mencuci sayuran. 3. Lakukan sub-aksi berikut hingga selesai sebanyak 100 kali.    1. Memotong sayur menjadi bagian kecil-kecil. 4. Lakukan sub-aksi berikut hingga selesai sebanyak 100 kali.    1. Memasak dan menyajikan makanan\*\*   Di mana:  \*) Menyiapkan bahan dan peralatan memasak berisi:   * Mengambil talenan di rak dan mengambil sayuran di rak. * Jika sering meninggalkan bekas kotor saat memasak, pakailah celemek.   \*\*) Memasak dan menyajikan makanan berisi:   * Memasak kaldu dalam panci. * Memasukkan sayur yang telah terpotong ke dalam panci. * Menyajikan sup dalam mangkuk. |

Berdasarkan serangkaian aksi di atas, manakah yang merupakan satu kesatuan blok program? Salah satu contohnya adalah tahapan memasak kaldu hingga menyajikan sup. Mengapa demikian? Kumpulan tahapan tersebut berada dalam satu blok atau unit yang memiliki tujuan tertentu, yaitu memasak dan menyajikan makanan.

Lalu, apakah hanya itu saja? Jelas tidak. Jika kita mengamati secara saksama, sub-aksi dalam indentasi yang sama merupakan satu kesatuan blok program. Dengan kata lain, indentasi yang berbeda bukan disebut sebagai blok program. Berikut adalah blok program pada aksi memasak sup.

[](https://www.dicoding.com/academies/620/tutorials/34968?from=34963)

Melihat ilustrasi di atas, apakah satu tahapan aksi “mencuci sayuran” merupakan sebuah blok program? Betul sekali. Walaupun hanya satu instruksi, ia dapat dikatakan sebagai satu blok program karena berada pada indentasi yang berbeda dari instruksi lainnya.

Di sisi lain, mengapa ada satu blok program dengan indentasi yang berbeda-beda seperti blok program A? Coba kita memahami lebih dalam terkait pengertian blok program. Pada prinsipnya, blok program A itu adalah serangkaian aksi dengan tujuan untuk menghasilkan sup. Artinya, serangkaian aksi tersebut dapat dikatakan blok program walaupun memiliki indentasi yang berbeda dengan blok program B, C, dan D.

Mengapa kita sampai mempelajari blok program? Nyatanya, blok program ini memungkinkan kita untuk memahami alur jalannya program. Bayangkan serangkaian aksi memasak sup menjadi seperti berikut.

|  |
| --- |
| 1. Menyiapkan bahan dan peralatan memasak\* 2. Lakukan sub-aksi berikut hingga selesai sebanyak 100 kali. 3. Mencuci sayuran. 4. Lakukan sub-aksi berikut hingga selesai sebanyak 100 kali. 5. Memotong sayur menjadi bagian kecil-kecil. 6. Lakukan sub-aksi berikut hingga selesai sebanyak 100 kali. 7. Memasak dan menyajikan makanan.\*\*   Di mana:  \*) Menyiapkan bahan dan peralatan memasak berisi:   * Mengambil talenan di rak dan mengambil sayuran di rak. * Jika sering meninggalkan bekas kotor saat memasak, pakailah celemek.   \*\*) Memasak dan menyajikan makanan berisi:   * Memasak kaldu dalam panci. * Memasukkan sayur yang telah terpotong ke dalam panci. * Menyajikan sup dalam mangkuk. |

Apakah Anda bisa memahami serangkaian aksi di atas? Rasanya tidak. Kita sebagai pelaku aksi akan merasa ada tahapan yang salah dan tidak bisa dijalankan dengan benar. Itulah alasan kita perlu memahami blok program.

Blok program ini menjadi hal krusial bagi beberapa bahasa pemrograman. Adakalanya, bahasa pemrograman mensyaratkan adanya pembatasan yang jelas antara satu blok program dengan blok program lainnya. Salah satu bahasa pemrograman yang menerapkan syarat tersebut adalah Python.

Indentasi pada Python memiliki peran penting dalam meningkatkan keterbacaan kode. Tidak hanya untuk mengatur tata letak kode, tetapi juga untuk memberi tanda awal dan akhir blok program secara jelas. Ada beberapa komponen Python, seperti pengambilan keputusan, fungsi, dan masih banyak lainnya, yang menerapkan blok program untuk menjaga struktur kode tetap rapi. Berikut adalah contoh kode Python yang menerapkan blok program secara rapi.

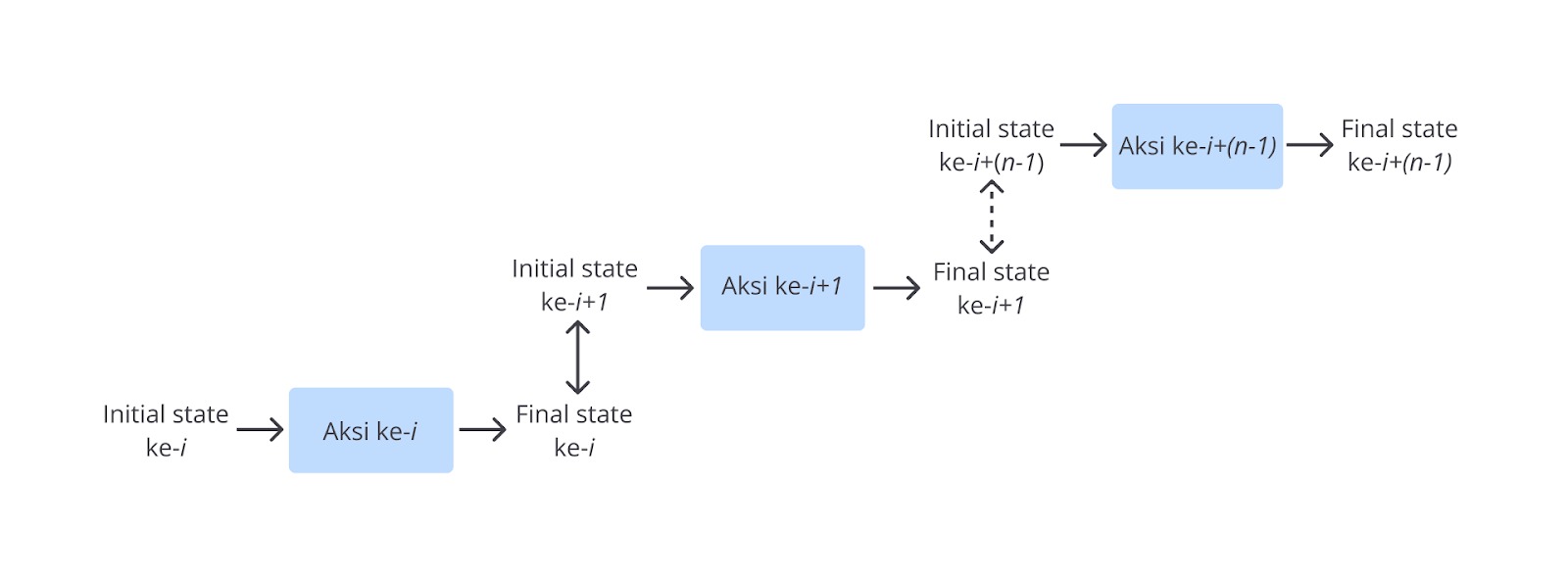
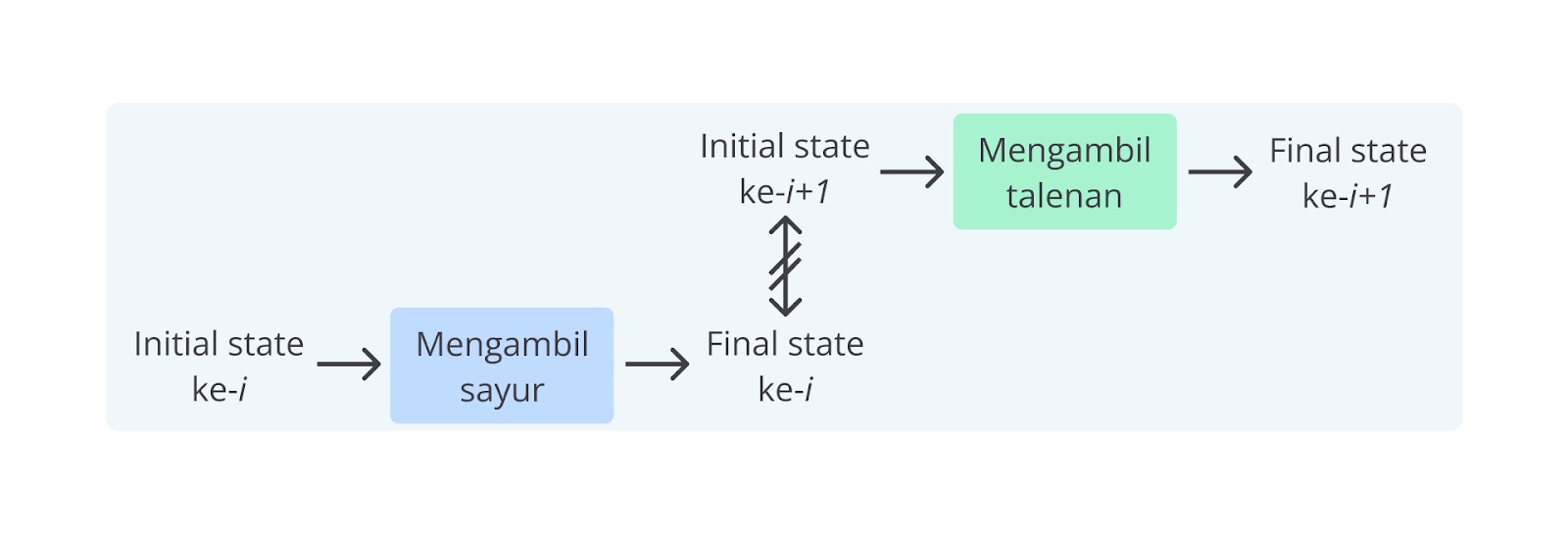
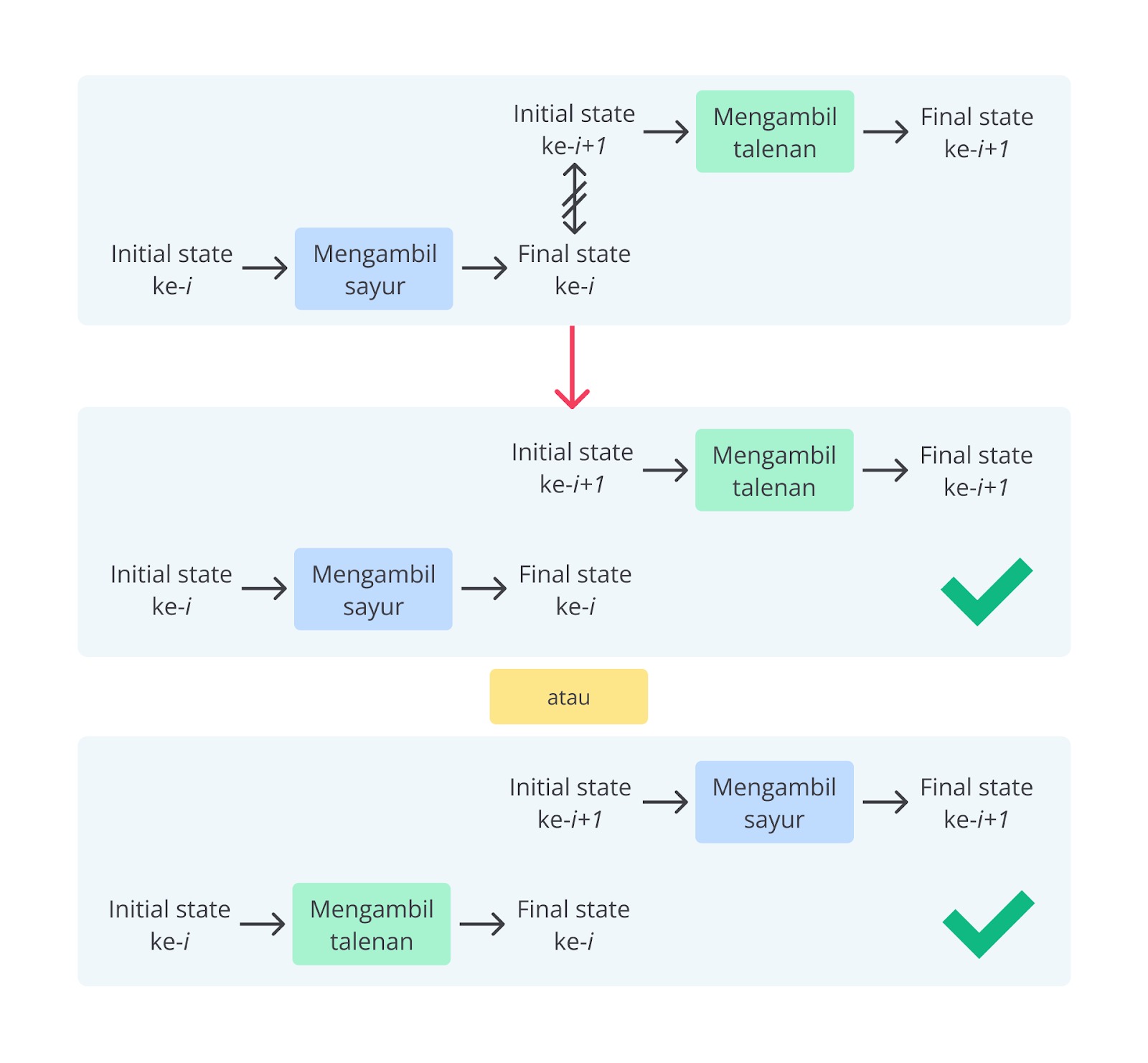
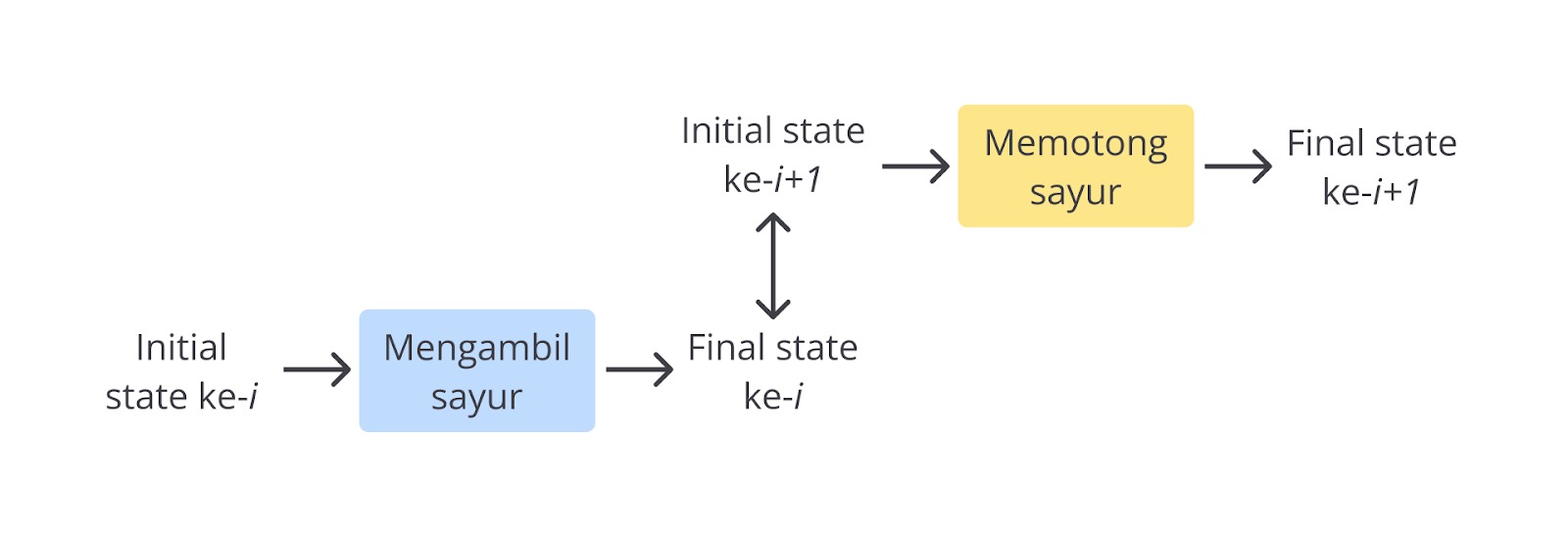
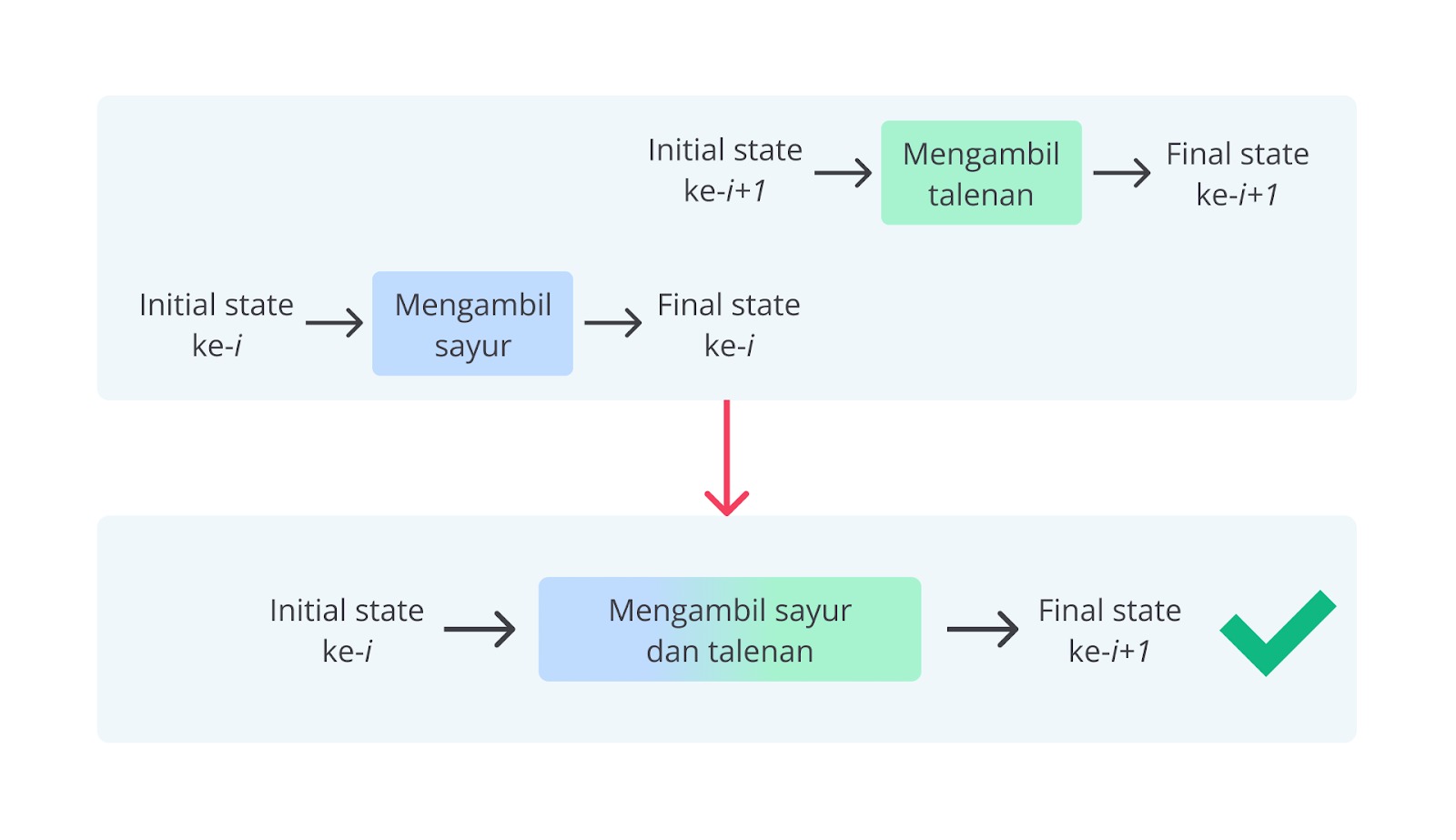
1. isReady = True
3. if isReady:
4. print("memasak sup")
6. print("aksi telah berakhir")

Apabila Anda memiliki pertanyaan terkait materi ini, silakan bertanya di Forum Diskusi. Selain itu, Anda juga menambah wawasan terkait materi ini melalui tautan berikut.

* [Execution Model](https://docs.python.org/3/reference/executionmodel.html)

## Rangkuman Berpikir Secara Prosedural

Selamat! Anda sudah berada di penghujung materi terkait Berpikir Secara Prosedural. Sejauh ini, Anda telah berlatih cara mengamati suatu algoritma program lebih dalam. Berikut adalah ringkasan materi yang sudah Anda pelajari.

* Pemrograman prosedural terdiri dari algoritma program yang tersusun secara sekuensial. Kita harus merancang suatu program dengan serangkaian instruksi secara berurutan dalam mencapai hasil yang diinginkan.
* Aksi sekuensial atau sequential statement adalah instruksi primitif atau aksi yang dilakukan berdasarkan urutan penulisannya. Tahapan harus diselesaikan satu demi satu dan berurutan.
* Aksi sekuensial dapat diartikan sebagai deretan instruksi yang terdiri dari tahapan 1, 2, 3, 4, hingga n. Nilai n ini dapat didefinisikan berapa pun asalkan selalu bernilai bilangan bulat positif lebih dari 0.
* Dalam aksi sekuensial, ada dua state yang saling terhubung antara aksi satu dengan lainnya, yaitu **initial state** dan **final state**.
  + **Initial state**adalah status aksi yang harus dipenuhi; sedangkan
  + **Final state**adalah status setelah instruksi atau aksi terlaksana.
* Pada instruksi atau aksi ke-i, final state akan menjadi initial state untuk i+1. Dengan kata lain, setiap aksi akan saling terhubung dan tersusun dengan baik.   
  [](https://www.dicoding.com/academies/620/tutorials/34973?from=34968)
* Setiap menjalankan aksi atau sub-aksi tertentu, pastikan seluruh initial state terpenuhi dengan baik. Apabila tidak terpenuhi, aksi tersebut tidak akan bisa berjalan dengan semestinya.
* Setelah aksi pertama terlaksana, muncullah final state yang memiliki status bahwa aksi telah berakhir. Menjelang melangkah ke aksi kedua, final state tadi berubah menjadi initial state. State inilah yang perlu dicek pada tahapan kedua.
* Apa yang terjadi ketika initial state tidak memenuhi kriteria pada suatu tahapan tertentu? Keseluruhan aksi akan gagal dilakukan dengan baik karena tidak mencapai keluaran yang diinginkan.
* Kita bisa memanipulasi kondisi tidak adanya initial state supaya aksi tetap berjalan. Alih-alih memaksa aksi berjalan tanpa initial state, kita dapat menambahkan aksi baru. Cara ini memungkinkan kita untuk melanjutkan keseluruhan aksi tanpa adanya kegagalan.
* Saat ada sekumpulan tahapan yang sekiranya tidak perlu disusun secara sekuensial, kita bisa menggabungkannya menjadi satu tahapan saja. Hal ini juga dilihat dari segi initial dan final state dari sekumpulan tahapan tersebut. Apabila state tidak berimbas pada tahapan lainnya, tahapan tersebut dapat dikumpulkan menjadi satu.
* Terkadang, kita tidak bisa menormalisasikan atau menyamaratakan suatu rangkaian aksi. Terkadang, ada keadaan bahwa kita membutuhkan adanya pengambilan keputusan pada aksi tertentu. Jadi, kita membutuhkan suatu pernyataan opsional untuk menjalankan aksi tersebut.
* Ada beberapa aksi yang tidak perlu dilakukan secara berulang dan ada juga aksi yang perlu diulang berkali-kali. Hal ini bergantung dengan tujuan atau keluaran yang ingin kita capai. Kita perlu mengamati kembali susunan aksi yang berulang dan tidak perlu berulang.
* Selain itu, ada kumpulan yang dapat dijadikan satu kelompok dengan tujuan serupa. Hal ini dapat dilakukan supaya meningkatkan struktur dan kejelasan suatu aksi serta mempermudah pemahaman bagi pelaku aksi.
* Bagaimana kita mengelompokkan kumpulan aksi yang memiliki satu tujuan? Cara paling mudah adalah menjadikan satu fungsi atau prosedur. Dengan mengelompokkan aksi yang serupa, kita dapat memisahkan bagian-bagian aksi terkait secara terpisah.
* Terkadang, kita telah menuliskan program dengan algoritma yang runtut. Dengan memberikan pemisah berupa new line atau paragraf baru, kita dapat memahami dan memudahkan orang lain dalam mengamati setiap instruksi yang sudah dibuat. Namun, adakalanya kita dapat membuat instruksi tersebut lebih ringkas atau sederhana.
* Adakalanya, suatu perintah tidak terikat dengan perintah sebelumnya atau setelahnya. Apabila menemukan perintah dengan kondisi tersebut, kita dapat melakukan dua hal.
  + Kita dapat mengubah posisi instruksi secara acak. Memindahkan perintah kedua menjadi pertama atau tetap seperti sedia kala.
  + Kita dapat menggabungkan kedua instruksi tersebut menjadi satu supaya berjalan secara bersamaan.
* Saat ada perintah seperti kondisi tersebut, kedua instruksi memiliki initial dan final state yang tidak beririsan. Umumnya, kedua state saling berhubungan, tetapi kondisi ini berbeda. Ia tidak terikat satu dengan lainnya.  
  [](https://www.dicoding.com/academies/620/tutorials/34973?from=34968)
* Apabila ada dua perintah yang tidak terikat, kita bisa memanipulasi susunan aksi untuk meletakkannya menjadi posisi pertama atau kedua.  
  [](https://www.dicoding.com/academies/620/tutorials/34973?from=34968)
* Pada bahasa pemrograman Python, kita bisa mengibaratkan kedua perintah ini adalah proses inisialisasi variabel. Kedua variabel tersebut pastinya tidak saling terikat satu sama lain. Dengan cara yang sama, kita bisa mengubah susunan instruksi menjadi urutan pertama atau kedua.
  + getVegetablesDone = True
  + getCuttingBoardDone = True
  + # atau
  + getCuttingBoardDone = True
  + getVegetablesDone = True
* Berbeda cerita ketika melaksanakan suatu instruksi yang terikat dengan instruksi sebelumnya. Perintah tersebut harus memiliki initial state yang mendukung perintah tersebut. Apabila tidak tersedia, perintah tidak dapat terlaksana dengan baik.  
  [](https://www.dicoding.com/academies/620/tutorials/34973?from=34968)
* Pada bahasa pemrograman Python, saat ada perintah yang terikat dengan perintah sebelumnya, urutan perintah tersebut tidak dapat bertukar.
  + getVegetablesDone = True
  + cutVegetables = getVegetablesDone and True
  + # tidak boleh menjadi seperti berikut
  + cutVegetables = getVegetablesDone and True
  + getVegetablesDone = True
* Selain mengubah urutan instruksi, ada opsi lain yaitu menggabungkan dua instruksi menjadi satu. Dua perintah yang tidak saling terhubung dapat digabung menjadi satu perintah saja.  
  [](https://www.dicoding.com/academies/620/tutorials/34973?from=34968)
* Pada bahasa pemrograman Python, ketika mempunyai dua perintah yang tidak terikat, kita bisa menggabungkan menjadi satu baris perintah saja. Konsepnya hanya menggabungkan dua baris perintah dan dipisahkan dengan karakter semicolon“;”.
  + getVegetablesDone = True
  + getCuttingBoardDone = True
  + # menjadi
  + getVegetablesDone = True; getCuttingBoardDone = True
* Ada konsep lain yang bernama one-liner, yakni menulis suatu kode atau fungsi dalam satu baris yang singkat dan padat. Konsep ini dapat diterapkan pada bahasa pemrograman Python, seperti mengetahui kata palindrom, menukar dua variabel, dll.
  + getVegetablesDone = True
  + getCuttingBoardDone = True
  + # menjadi
  + getVegetablesDone, getCuttingBoardDone = True, True
* Konsep one-liner ini memiliki kelebihan, yaitu membantu meningkatkan keterbacaan kode dan mempercepat penulisan kode supaya lebih ringkas. Terkadang, perintah yang kompleks dalam suatu perintah one-liner memengaruhi keterbacaan dan pemahaman kode.
* Blok program atau blok kode adalah kumpulan instruksi yang berjalan pada satu unit yang sama untuk mencapai tujuan tertentu.
* Sub-aksi dalam indentasi yang sama merupakan satu kesatuan blok program. Berikut adalah contoh satu blok program.

|  |
| --- |
| Memasak dan menyajikan makanan berisi:   * + Memasak kaldu dalam panci.   + Memasukkan sayur ke dalam panci yang telah terpotong.   + Menyajikan sup dalam mangkuk. |

* Adakalanya, satu blok program memiliki indentasi yang berbeda-beda. Walaupun begitu, kumpulan instruksi tersebut masih memiliki tujuan yang sama. Jadi, sekumpulan instruksi tersebut masih bisa disebut sebagai satu blok program. Berikut adalah contoh satu blok program dengan indentasi yang berbeda.

|  |
| --- |
| * + Menyiapkan bahan dan peralatan memasak\*   + Lakukan sub-aksi berikut hingga selesai sebanyak 100 kali.     1. Mencuci sayuran.   + Lakukan sub-aksi berikut hingga selesai sebanyak 100 kali.     1. Memotong sayur menjadi bagian kecil-kecil.   + Lakukan sub-aksi berikut hingga selesai sebanyak 100 kali.     1. Memasak dan menyajikan makanan\*\* |

* Blok program ini menjadi hal krusial bagi beberapa bahasa pemrograman. Adakalanya, bahasa pemrograman mensyaratkan adanya pembatasan yang jelas antara satu blok program dengan blok program lainnya.
* Indentasi pada Python memiliki peran penting dalam meningkatkan keterbacaan kode. Tidak hanya untuk mengatur tata letak kode, tetapi juga untuk memberi tanda awal dan akhir blok program secara jelas.
* Ada beberapa komponen Python, seperti pengambilan keputusan, fungsi, dan masih banyak lainnya, yang menerapkan blok program untuk menjaga struktur kode tetap rapi. Berikut adalah contoh kode Python yang menerapkan blok program secara rapi.
  + isReady = True
  + if isReady:
  + print("memasak sup")
  + print("aksi telah berakhir")