**Percabangan dan Ternary Operators**

*Control flow* adalah sebuah cara untuk memberi tahu program mengenai instruksi yang harus dijalankan dan di mana harus memulai dan berakhir. ada materi sebelumnya, Anda telah mempelajari aksi sekuensial. Python akan menjalankan kode Anda berdasarkan deretan instruksi yang dibuat secara sekuensial.

Dalam Python, program dapat berupa blok kode. Python sangat memperhatikan indentasi untuk membangun sebuah blok kode. Salah satu blok pemrograman adalah perulangan. Perulangan adalah satu dari beberapa control flow.

Control flow memungkinkan program untuk berjalan berdasarkan jalur eksekusi. Control flow terbagi menjadi beberapa jenis, yakni kondisi tertentu (percabangan), mengulang blok kode secara berulang (perulangan), melewati sebagian kode dan berhenti di kode tertentu, hingga mendefinisikan fungsi.

**Percabangan**

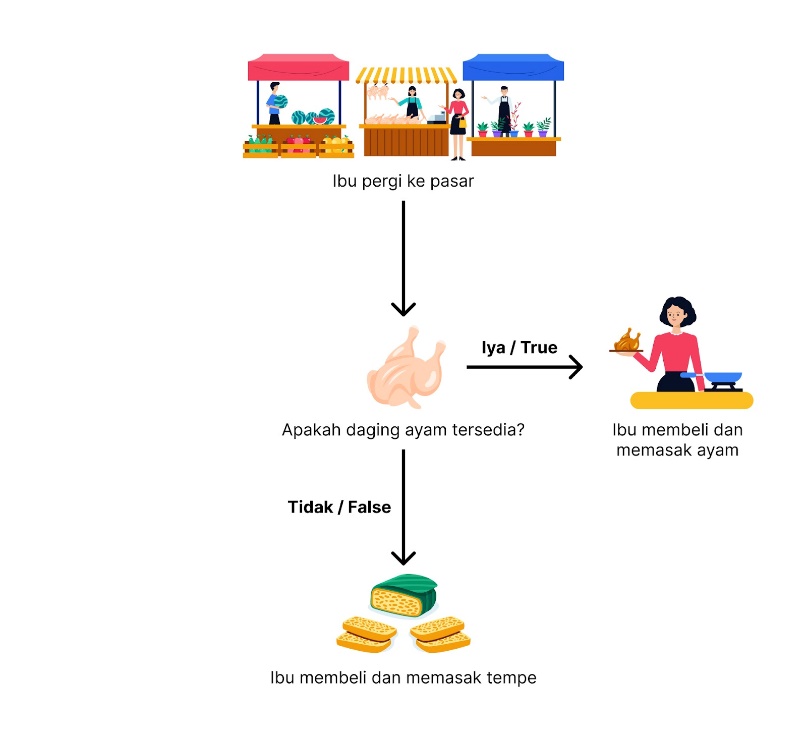
Dalam pemrograman, sebuah kode program dapat berjalan berdasarkan kondisi tertentu. Maknanya, Anda dapat memberikan instruksi berdasarkan **"Jika-maka"**(if-else).

Misalnya dalam keadaan seperti berikut.

1. **Jika**Anda tidak menyelesaikan kelas Memulai Pemrograman dengan Python, **maka** Anda tidak lulus dari kelas Memulai Pemrograman dengan Python.
2. **Jika** jumlah variabel nama kurang dari dua, **maka** variabel tersebut tidak memenuhi kriteria kondisi.

Sebenarnya, kasus percabangan sering kita jumpai dalam kehidupan sehari-hari. Simak ilustrasi berikut.

"Setiap hari, Ibu selalu pergi ke pasar untuk membeli bahan makanan. Ibu selalu mengutamakan untuk membeli daging ayam di pasar. **Jika** daging ayam tidak tersedia, **maka**Ibu akan membeli tempe sebagai pengganti, lalu memasaknya."



Layaknya ilustrasi di atas, setiap kondisi akan mengembalikan nilai *true* atau *false*. Dengan nilai boolean ini, Anda dapat menentukan instruksi selanjutnya. Misalnya, **jika**ayam tersedia (bernilai *true*), **maka**ibu akan membeli ayam dan memasaknya.

ketersediaan = "Daging ayam"

if ketersediaan == "Daging ayam":

print("Ibu membeli dan memasak ayam")

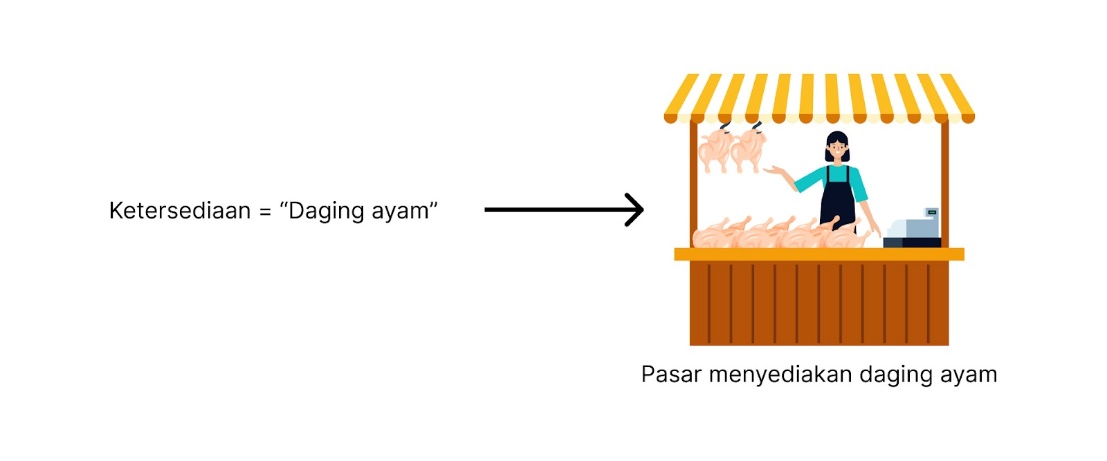
else:

print("Ibu membeli dan memasak tempe")

Output:

Ibu membeli dan memasak ayam

Kode di atas merupakan program percabangan. **Jika**variabel "ketersediaan" bernilai "Daging ayam", **maka**akan mengembalikan string "Ibu membeli dan memasak ayam". Kita bisa asumsikan variabel "ketersediaan" sama seperti ketersediaan bahan makanan dari pasar yang ibu kunjungi. Jika pasar tersebut menyediakan "Daging ayam", variabel tersebut bernilai "Daging ayam".



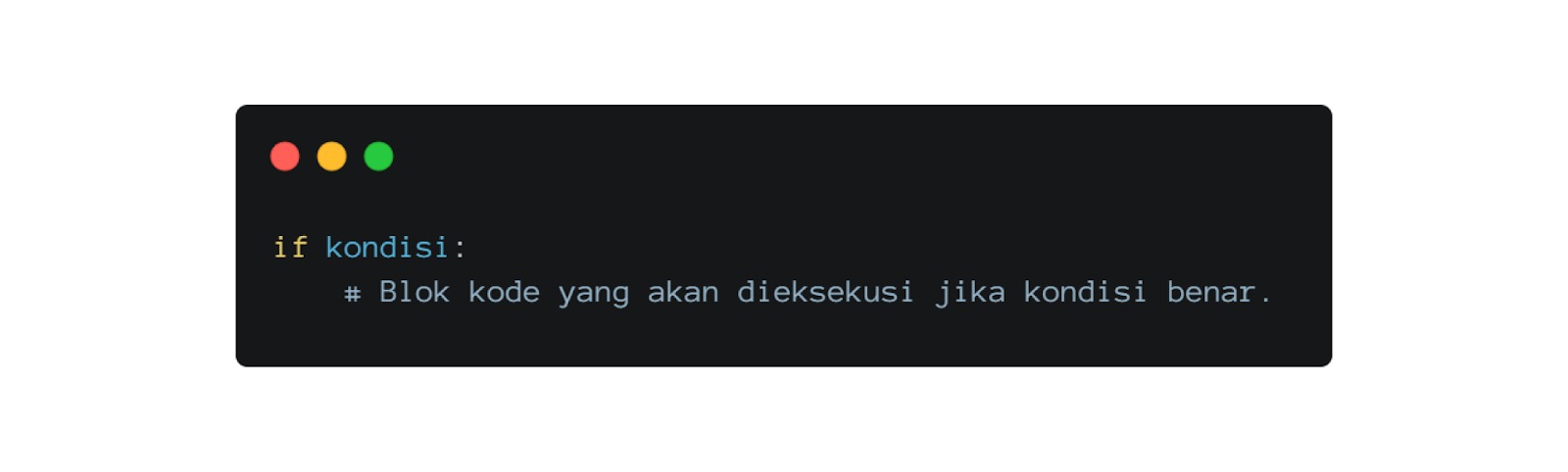
Jika daging ayam tidak tersedia di pasar, maknanya variabel "ketersediaan" akan bernilai kosong atau bernilai bahan makanan lain. Dengan begitu, jika variabel "ketersediaan" tidak memiliki nilai "Daging ayam", variabel ketersediaan tidak lagi memenuhi kondisi "**if ketersediaan == 'Daging ayam'**". Jadi, program akan mengembalikan teks atau string "Ibu membeli dan memasak tempe".

Ingat bahwa Python adalah bahasa pemrograman *case-sensitive*, hal ini berlaku juga pada percabangan. Buktikan sendiri dengan mengubah **"Daging ayam"**menjadi **"Daging Ayam"**. Apakah output program masih sama?

Dalam ilustrasi di atas, kita paham bahwa percabangan melibatkan **if** dan **else** statement. Anda dapat mengasumsikan statement sebagai instruksi. Selain **if** dan **else** statement, sebenarnya Python masih memiliki satu statement lagi yang sering digunakan, yakni **elif**. Mari kita pelajari satu per satu.

**If**

*If*adalah statement Python yang akan mengecek nilai variabel di dalamnya memenuhi kriteria suatu kondisi atau tidak. Jika memenuhi kriteria, kondisi tersebut bernilai *true*. Jika tidak memenuhi kriteria, kondisi akan bernilai *false*. Jika kondisi if bernilai *true*, kode yang berada dalam blok kode **if** akan dieksekusi.



Perlu diingat bahwa if merupakan blok kode. Jadi, Anda perlu memperhatikan indentasi untuk menjalankan kode, seperti yang ditunjukkan gambar.

Hal lain yang perlu diperhatikan adalah Python menganggap setiap nilai kosong (zero) dan null sebagai False. Sebaliknya, nilai yang tidak kosong (non-zero) dan tidak null (non-null) akan bernilai True. Untuk lebih mengetahui maksudnya, mari lihat kode berikut.

x = ""

if x:

print("Ini True")

Output:

Pada baris pertama kode program di atas, variabel "x" diinisialisasikan dengan string kosong "". Kemudian if statement mengevaluasi variabel "x" dan menghasilkan nilai salah (False). Hal ini terjadi karena variabel "x" berisi string kosong dan Python mengevaluasinya sebagai False. Sebab hasil kondisinya adalah False, blok kode dalam **if** tidak dijalankan.

Beberapa nilai yang dianggap sebagai false oleh Python sebagai berikut.

1. Nilai yang sudah didefinisikan bernilai salah: **None dan False**.
2. Angka nol dari semua tipe numerik: **0, 0.0, 0j, Decimal(0), Fraction(0,1)**.
3. Urutan (*sequence*) dan koleksi (*collection*) yang kosong: **"", (), {}, set(), range(0)**.

Selain nilai di atas, Python akan menganggap semua nilai sebagai true.

Terakhir, if statement memiliki versi one-liner-nya. Ini memungkinkan Anda untuk membuat kode **if** dalam bentuk *single statement* atau satu baris, tanpa perlu memperhatikan indentasi dan membuat blok kode.

score = 100

if score == 100: print("Nilai Anda sempurna!")

Output:

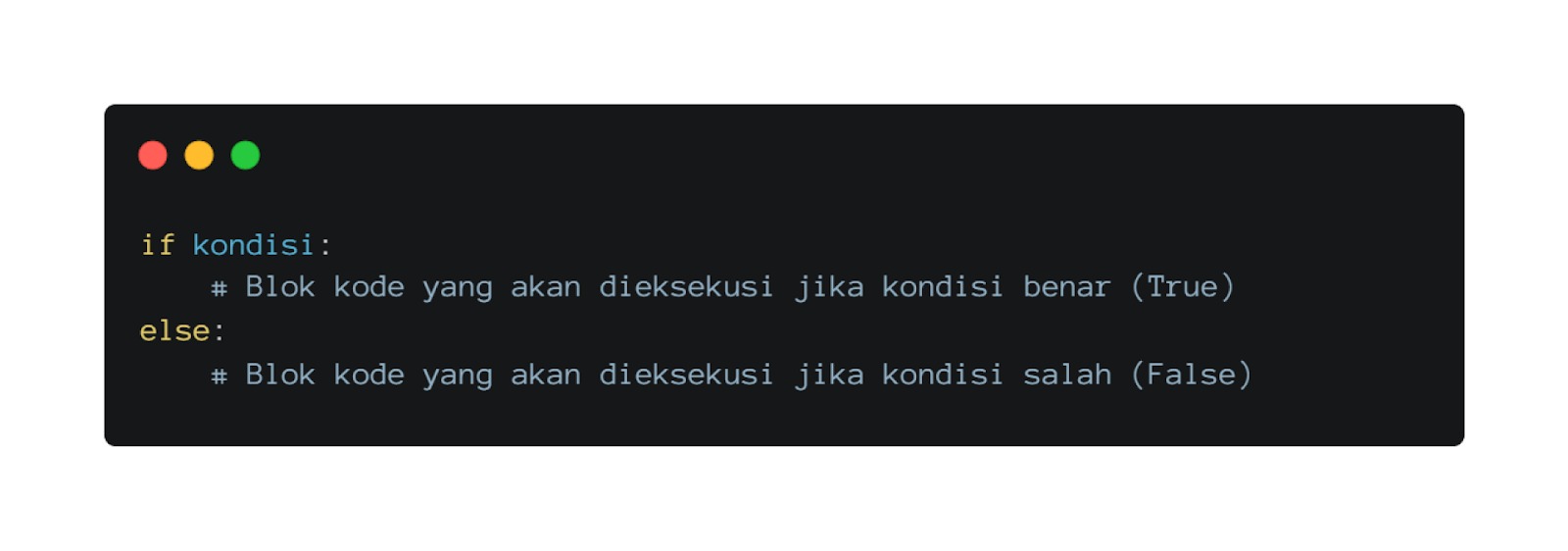
Nilai Anda sempurna!

Kode program di atas merupakan program yang sama dengan contoh pertama sebelumnya. Perhatikan bahwa kode "print()" disimpan setelah tanda titik dua ":". Program tidak menganggap itu sebagai kesalahan sehingga dapat menghasilkan output yang diharapkan, yakni teks/string "Nilai Anda sempurna!".

**Else**

*Else* adalah statement yang menjadi jalan keluar saat kondisi atau hasil evaluasi if statement bernilai *false*. Maksudnya adalah program akan menjalani blok kode **if** terlebih dahulu dan jika hasilnya adalah false, program akan menjalankan **else** statement sebagai jalan keluar atau kondisi terakhir.

Jika kita gabungkan if dan else, struktur berikut akan dihasilkan.



Perhatikan gambar di atas, secara sekuensial program akan menjalankan kondisi if statement terlebih dahulu. Jika hasil kondisi adalah true, blok kode dalam if statement akan dieksekusi. Namun, jika kondisi if statement bernilai false, else statement akan dijalankan dan blok kode dalam else statement akan dieksekusi.

Else termasuk statement bersifat opsional. Umumnya, else statement digunakan ketika memiliki kondisi terakhir saat semua kondisi tidak terpenuhi. Mari tinjau penerapannya dalam kasus pengecekan tinggi badan suatu pengunjung untuk menaiki *roller coaster*.

tinggi\_badan = 120

if tinggi\_badan >=160:

print("Anda boleh menaiki roller coaster")

else:

print("Anda tidak diperbolehkan menaiki roller coaster")

Output:

Anda tidak diperbolehkan menaiki roller coaster

Pada program di atas, Anda seolah berkata **"Jika tinggi badannya adalah 160 cm ke atas, maka diperbolehkan naik roller coaster. Jika di bawah 160 cm, maka pengunjung tidak diperbolehkan menaiki roller coaster"**.

Baris pertama, variabel tinggi\_badan diinisialisasikan dengan nilai 120. Ini artinya pengunjung memilki tinggi badan 120 cm. Program akan mengevaluasi variabel tersebut pada blok kode **if** statement. Sebab tinggi badannya adalah 120, kondisi akan bernilai false pada **if** statement tersebut dan tidak menjalankan kode di dalamnya.

Selanjutnya program menjalankan blok else statement sebagai upaya terakhir. Jika if statement menyatakan "tinggi badan harus di atas atau sama dengan 160", else statement menyatakan sebaliknya, "tinggi badan harus di bawah 160 atau tepatnya 159 ke bawah".

Jika pengunjung atau variabel "tinggi\_badan" bernilai 120, kondisi ini akan memenuhi else statement. Lalu, blok kode dalam else statement dijalankan.

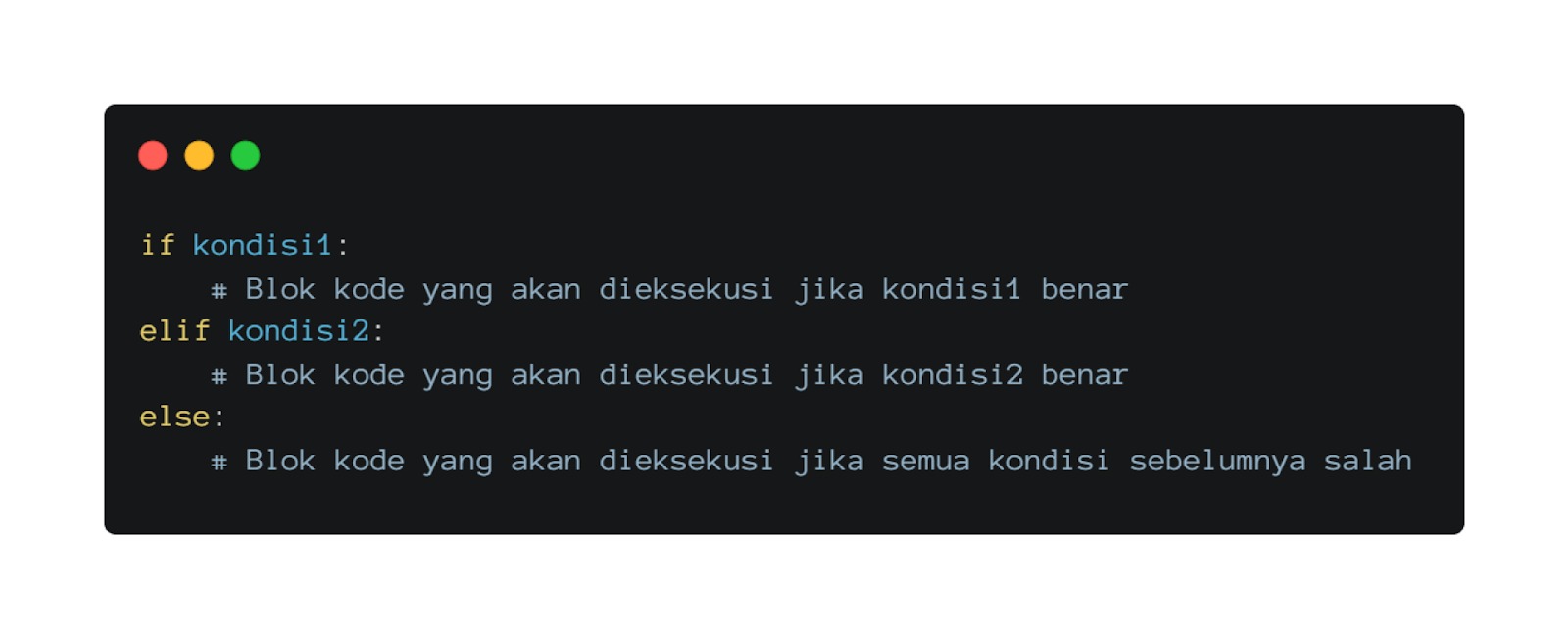
Apabila Anda memiliki pertanyaan seperti “bagaimana jika kondisi else statement tidak terpenuhi?” Jawabannya adalah kondisi else statement sudah pasti terpenuhi karena merupakan jalan keluar terakhir dalam suatu percabangan. Anda harus menggunakan else statement sebagai kondisi terakhir dalam percabangan dan bukan kondisi ke-2, kondisi ke-3, dan seterusnya.

Jika ingin menambah kondisi baru, seperti kondisi ke-2, kondisi ke-3, dan seterusnya, jangan gunakan else statement. Untuk hal itu, Anda bisa menggunakan **elif** statement.

**Elif**

*Elif* merupakan kependekan dari *else if* dan alternatif untuk if bertingkat atau *switch case*. Elif statement berada pada posisi setelah if. Anda dapat menambahkan elif statement lebih dari satu karena tidak dibatasi dan opsional.

Struktur keseluruhan percabangan jika kita gabungkan antara **if**, **elif**, dan **else** adalah berikut.



Perhatikan gambar di atas, secara sekuensial program akan menjalankan **if statement**terlebih dahulu. Jika kondisinya bernilai true, blok kode di dalamnya akan dieksekusi. Jika kondisinya false, **elif statement**akan dijalankan.

Jika kondisi **elif statement** menghasilkan true, blok kode di dalamnya akan dieksekusi. Kondisi **else statement**akan dijalankan dan kode di dalamnya akan dieksekusi jika semua kondisi sebelumnya salah atau menghasilkan false.

nilai = 65

if nilai>=80:

print("Selamat! Anda mendapat nilai A")

print("Pertahankan!")

elif nilai>=70:

print("Hore! Anda mendapat nilai B")

print("Tingkatkan!")

elif nilai>=60:

print("Hmm.. Anda mendapat nilai C")

print("Ayo semangat!")

else:

print("Waduh, Anda mendapat nilai D")

print("Yuk belajar lebih giat lagi!")

Output:

Hmm.. Anda mendapat nilai C

Ayo semangat!

Program di atas merupakan contoh penerapan if, else, dan elif statement. Kasus yang digunakan adalah penilaian tugas siswa. Jikanilai siswa lebih atau sama dengan dari 80, siswa akan mendapatkan nilai A. Jika nilai siswa lebih atau sama dengan dari 70 dan kurang dari 80, siswa akan mendapatkan nilai B. Jika nilai siswa lebih atau sama dengan dari 60 dan kurang dari 70, siswa akan mendapatkan nilai C. Jika nilai siswa kurang dari 60, siswa akan mendapatkan nilai D.

Pada program di atas, diasumsikan bahwa siswa memiliki nilai 65. Jadi, variabel "nilai" diinisialisasi dengan nilai 65. Program akan mengevaluasi satu persatu secara sekuensial dari if statement pertama hingga else statement.

Kondisi if pertama tidak memenuhi kriteria karena nilai harus lebih dari 80, program lanjut ke elif statement pertama dan tidak memenuhi kriteria karena harus bernilai lebih dari 70. Program berlanjut ke elif statement kedua, hasil evaluasinya ternyata memenuhi kriteria, yakni nilai lebih dari 60. Program mengeksekusi blok kode di dalamnya dengan menampilkan teks "Hmm.. Anda mendapat nilai C Ayo semangat!".

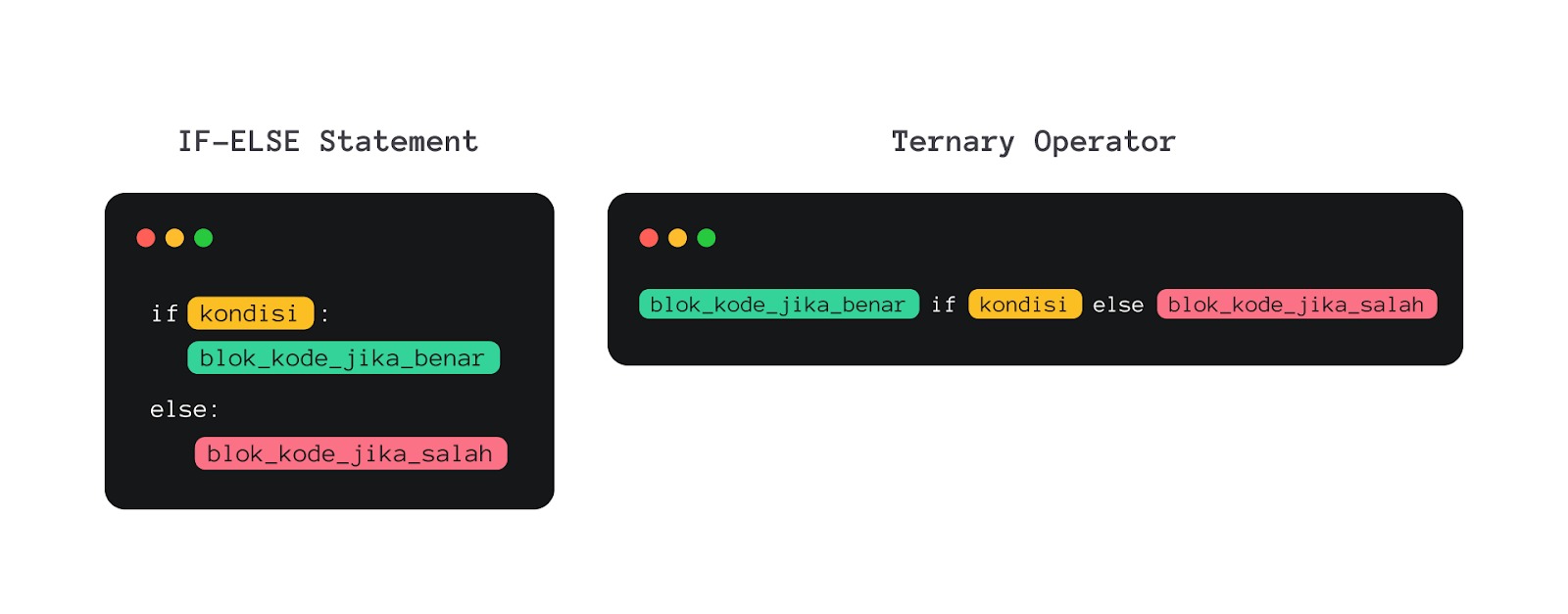
Perlu diingat bahwa else statement tidak akan dijalankan jika kondisi sebelumnya terpenuhi seperti pada kasus di atas.

Untuk informasi tambahan, kita juga dapat menambahkan **'and'** atau **'or'**operator dalam kondisi percabangan.

**Ternary Operators**

*Ternary operators* termasuk *conditional expressions* pada Python. *Conditional expressions* adalah bentuk ekspresi yang bertujuan untuk mengevaluasi kondisi dan mengembalikan nilai berdasarkan hasil evaluasinya. Anda bisa asumsikan bahwa ternary operators ini merupakan versi one-liner dari if dan else.

Untuk memahaminya, perhatikan gambar berikut.



Ternary operators dibangun dengan menempatkan "blok kode jika benar" pada posisi awal, lalu diikuti oleh "if statement" serta "kondisi"-nya. Kemudian "else statement" ditempatkan di akhir beserta dengan "blok kode jika salah".

lulus = True

print("selamat") if lulus else print("perbaiki")

Output:

selamat

Kode program di atas menampilkan pesan teks "selamat" jika kondisi bernilai true dan menampilkan pesan teks "perbaiki" jika kondisi bernilai false. Jika kita transformasikan menjadi bentuk blok, berikut adalah kodenya.

lulus = True

if lulus:

print("Selamat")

else:

print("Perbaiki")

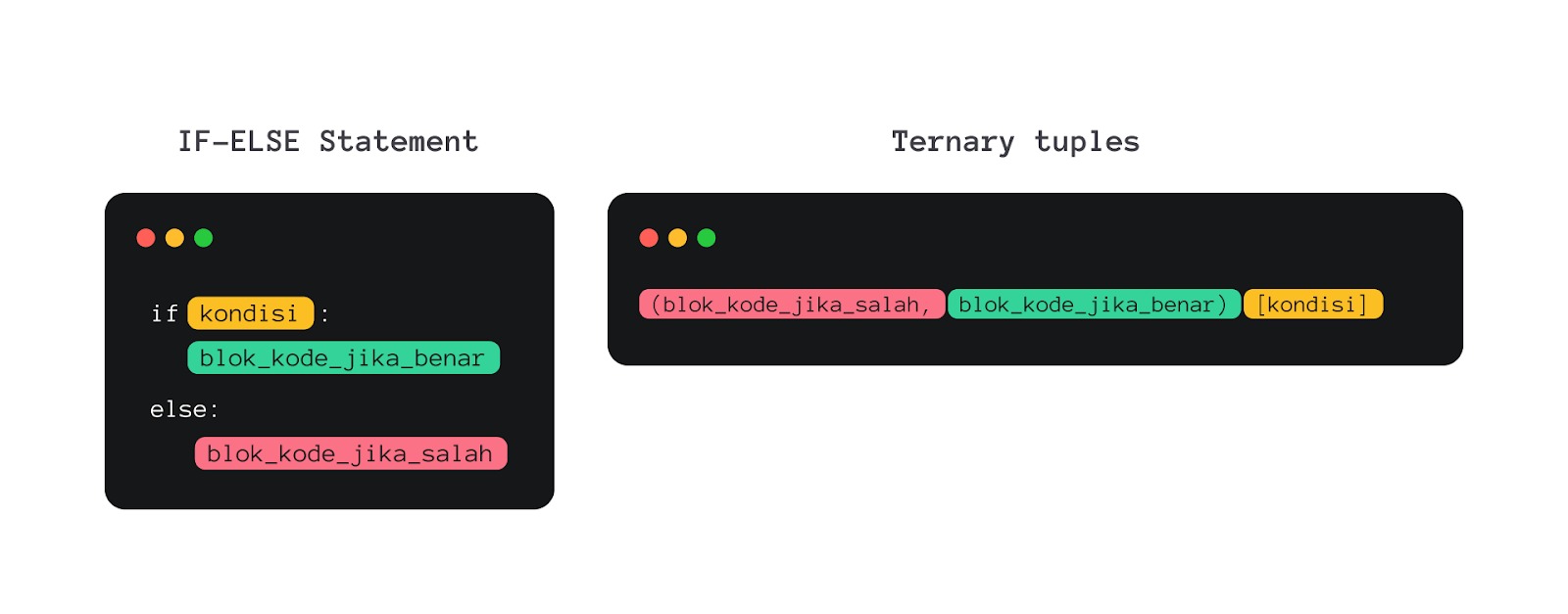
Output:

Selamat

Kode di atas merupakan program yang sama dengan contoh sebelumnya. Program akan menampilkan pesan teks "selamat" jika kondisi bernilai true dan menampilkan pesan teks "perbaiki" jika kondisi bernilai false. Silakan telaah secara perlahan untuk memahami maksudnya.

Perlu diingat bahwa tujuan dari one-liner bukanlah sekadar untuk memudahkan kode dibaca karena hanya dibuat dalam satu baris, melainkan untuk membuat kode menjadi lebih singkat dan jelas.

Opsi lain dari ternary operators adalah melibatkan tuple.



Perhatikan bahwa pada ternary tuples kita menggunakan indeks ke-0 tuples sebagai kode jika kondisi salah, sedangkan indeks ke-1 sebagai kode jika kondisi benar.

lulus = True

kelulusan = ("Perbaiki, Anda belum lulus.","Selamat, Anda lulus!")[lulus]

print(kelulusan)

Output:

Selamat, Anda lulus!

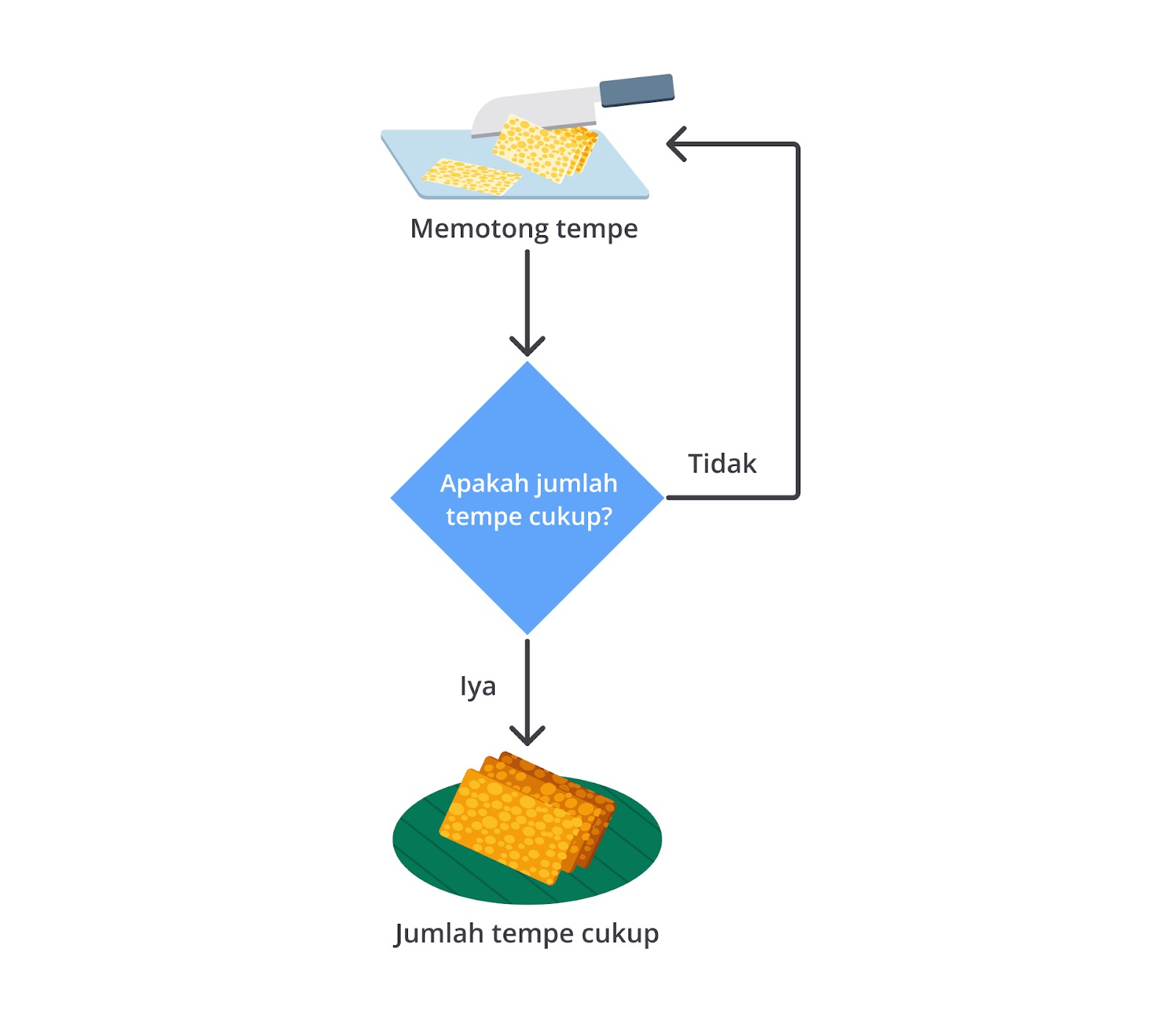
Kode program di atas menampilkan pesan teks "Selamat, Anda lulus!" jika kondisi bernilai true dan menampilkan pesan teks "Perbaiki, Anda belum lulus." jika kondisi bernilai false. Jika kita ubah menjadi blok kode IF, berikut penerapannya.

**Perulangan**

Dalam kehidupan sehari-hari, sering kali kita menemui situasi yang harus dilakukan berulang kali. Misalnya dalam skenario berikut.

"Setiap hari Rabu, pasar yang selalu dikunjungi oleh Ibu (pasar yang sama dengan cerita sebelumnya) selalu tidak menyediakan daging ayam. Maka dari itu, Ibu selalu membeli tempe sebagai gantinya.  Pada minggu biasa, Ibu hanya akan memotong 3 balok tempe karena jumlah anggota keluarga adalah 3 orang. Namun, pada minggu lain, Ibu kedatangan keluarga besar untuk makan bersama. Kali ini, Ibu tidak mengetahui total keluarga yang datang. Jadi, setelah memotong 1 balok tempe, Ibu akan selalu mengecek bahwa jumlah tersebut cukup atau tidak."

Pada skenario berikut, Ibu kedatangan keluarga besar di rumahnya dan berencana untuk membuat hidangan makanan berupa tempe. Ibu tidak mengetahui jumlah keluarga yang hadir sehingga setiap kali ada keluarganya yang datang, Ibu akan memotong 1 balok tempe untuk disajikan kepada 1 orang.



Perhatikan pada diagram di atas, Ibu akan selalu melakukan aktivitas berulang untuk memotong tempe hingga kondisinya terpenuhi. Kondisi yang dimaksud adalah jumlah keluarga yang hadir sama dengan jumlah tempe yang disajikan.

Dalam pemrograman, kita juga akan sering menemui masalah serupa yang mengharuskan untuk melakukan kode berulang. Contohnya menampilkan angka 1 hingga 10.

print(1)

:

:

print(10)

Pada kode di atas, program menampilkan angka dari 1 hingga 10 menggunakan sintaks yang berulang. Terlihat tidak efektif, bukan? Itulah yang menjadi tujuan dari materi perulangan ini, Anda akan belajar untuk membuat kode program yang efektif dan mudah dibaca oleh *programmer* lain.

Dalam Python, ada beberapa sintaks atau statement untuk melakukan perulangan.

**For**

*For* termasuk sintaks dalam Python yang bersifat *definite iteration*. *Definite* *iteration* adalah sebuah proses iterasi atau perulangan ketika jumlah pengulangannya ditentukan secara eksplisit sebelumnya.

Format dari perulangan for sebagai berikut.



[<iterable>](https://wiki.python.org/moin/Iterator) merupakan segala *object* dalam Python yang dapat diiterasi seperti list, tuple, hingga string. Ada pula <var> merupakan variabel yang akan mengambil elemen berikutnya dari <iterable> setiap kali iterasi berjalan.

var\_list = [1,2,3,4,5,6,7,8,9,10]

for i in var\_list:

print(i)

Kode di atas merupakan program yang bertujuan untuk menampilkan angka dari 1 hingga 10 berdasarkan variable list yang sudah diinisialisasikan sebelumnya. Perhatikan bahwa program di atas sebenarnya sama dengan program pada contoh sebelumnya. Jika contoh sebelumnya menggunakan sintaks "print()" yang berulang, program di atas menggunakan sintaks atau statement for.

Pada program di atas, kita melakukan perulangan untuk menampilkan angka dari 1 hingga 10 yang sebelumnya telah diinisialisasikan pada variabel "var\_list". Setiap iterasi atau perulangan yang berjalan, "i" akan mengambil elemen dari "var\_list" satu per satu. Lalu, blok kode "print(i)" akan dijalankan dengan nilai "i" adalah nilai yang sudah diambil sebelumnya.

Anda juga bisa menggunakan tipe data string sebagai object <iterable>, silakan ubah variable "var\_list" di atas dengan string apa pun yang Anda inginkan. Hasilnya program akan menampilkan setiap huruf dari string tersebut.

Anda juga dapat melakukan perulangan berdasarkan panjang suatu nilai dengan menggunakan fungsi "range()".

for i in range(10):

print(i)

Jika Anda perhatikan lebih baik, program di atas menampilkan angka dari 0 hingga 9 padahal kita menentukannya "10". Mengapa itu terjadi? Pada dasarnya, "range()" adalah fungsi bawaan dalam Python yang akan menghasilkan urutan bilangan dimulai dari indeks ke-0.

Sintaksis umum dari fungsi "range()" sebagai berikut.



Berikut adalah penjelasan detail terkait fungsi "range()".

1. "Start" merupakan nilai awal dari urutan bilangan yang bersifat opsional, jika Anda tidak memasukkannya, nilai awal akan dianggap 0.
2. "Stop" merupakan nilai batas yang wajib dimasukkan. Urutan akan berhenti sebelum mencapai nilai "stop" (eksklusif).
3. "Step" merupakan nilai penambahan antara setiap dua bilangan dalam urutan yang bersifat opsional. Jika nilai tersebut tidak diberikan, secara *default*nilai yang dimasukkan adalah 1.

for i in range(1,10,2):

print(i)

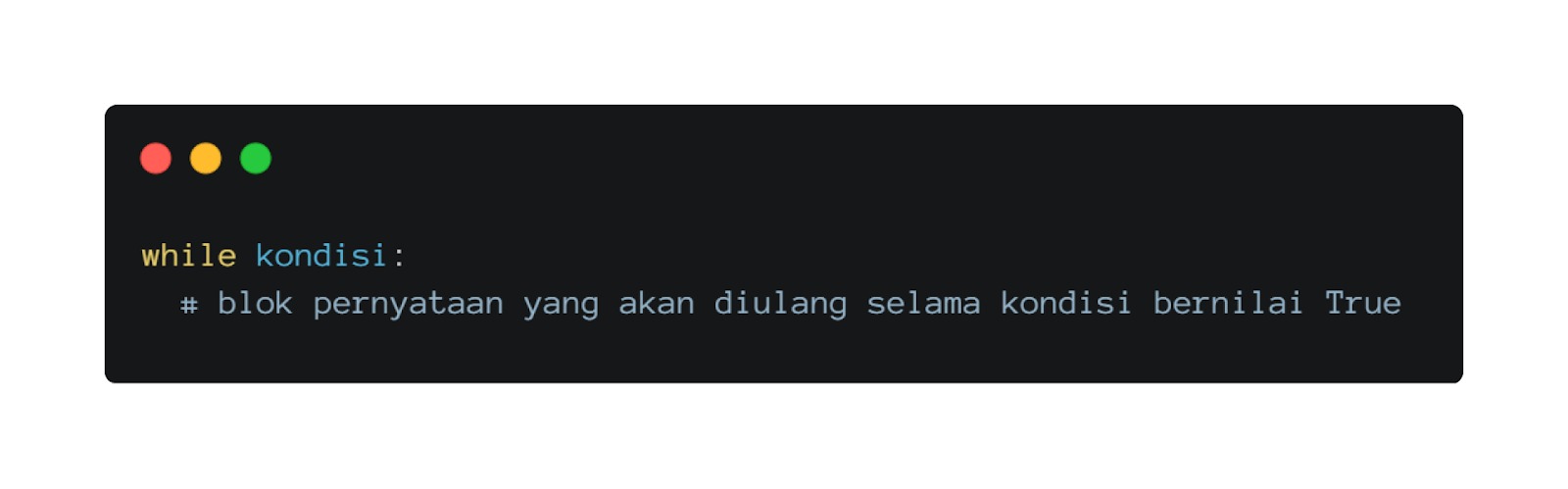
Pada program di atas, kita menampilkan bilangan ganjil yang dimulai dari 1 hingga 10. Perhatikan bahwa program di atas mendefinisikan nilai "1" sebagai "start", nilai "10" sebagai "stop", dan nilai "2" sebagai "step". Ingat bahwa "stop" bersifat eksklusif, yang artinya nilai terakhirnya tidak akan disertakan.

Dengan begitu, program di atas akan menampilkan kode dari 1 hingga 10 dengan setiap bilangan ke-2 dan kelipatannya akan dilewati atau tidak dicetak.

**While**

*While* termasuk sintaks dalam Python yang bersifat *indefinite iteration*. *Indefinite iteration* adalah sebuah proses iterasi yang akan berhenti ketika memenuhi kondisi tertentu.

Format dari perulangan while sebagai berikut.



Kondisi merupakan ekspresi yang akan dievaluasi dan menghasilkan nilai true atau false. Selama hasil evaluasi bernilai true, program akan terus berjalan hingga menghasilkan nilai false.

Berikut implementasinya.

counter = 1

while counter <= 5:

print(counter)

counter += 1 # Increment

Pada contoh di atas, kita menggunakan perulangan "while" untuk menampilkan angka 1 hingga 5. Variabel "counter" diinisialisasi dengan nilai 1 sebelum perulangan dimulai. Ini artinya perulangan akan dimulai dari 1 berdasarkan nilai variabel tersebut. Perulangan lalu berjalan dengan mengevaluasi variabel "counter" yang memiliki nilai "1". Hasil dari evaluasi tersebut bernilai true sehingga blok kode di dalamnya akan dijalankan.

Perhatikan bahwa kita menggunakan jenis ekspresi uner untuk melakukan *increment*. *Increment* adalah pola untuk menambahkan suatu variabel dengan nilai tetap. Dengan hal ini, setiap perulangan bernilai true maka variabel "counter" akan terus ditambah dengan nilai "1". Pada perulangan di atas, nilai variabel "counter" akan bertambah hingga nilainya adalah "5". Ketika nilai variabel "counter" menyentuh "5" maka hasil evaluasi akan bertambah menjadi "6", tetapi angka 6 tersebut tidak memenuhi kondisi "<=5" sehingga kode print() tidak akan dijalankan dan hanya akan memunculkan angka hingga 5.

Namun, Anda harus berhati-hati untuk tidak melakukan *infinite loop*, yakni sebuah kondisi ketika perulangan tidak berhenti karena tidak memenuhi kondisi yang diinginkan. Contohnya adalah ketika melakukan perulangan, kita tidak memberikan **increment**yang menyebabkan variabel atau counter tidak akan memenuhi kondisi **while**.

counter = 1

while counter <= 5:

print(counter)

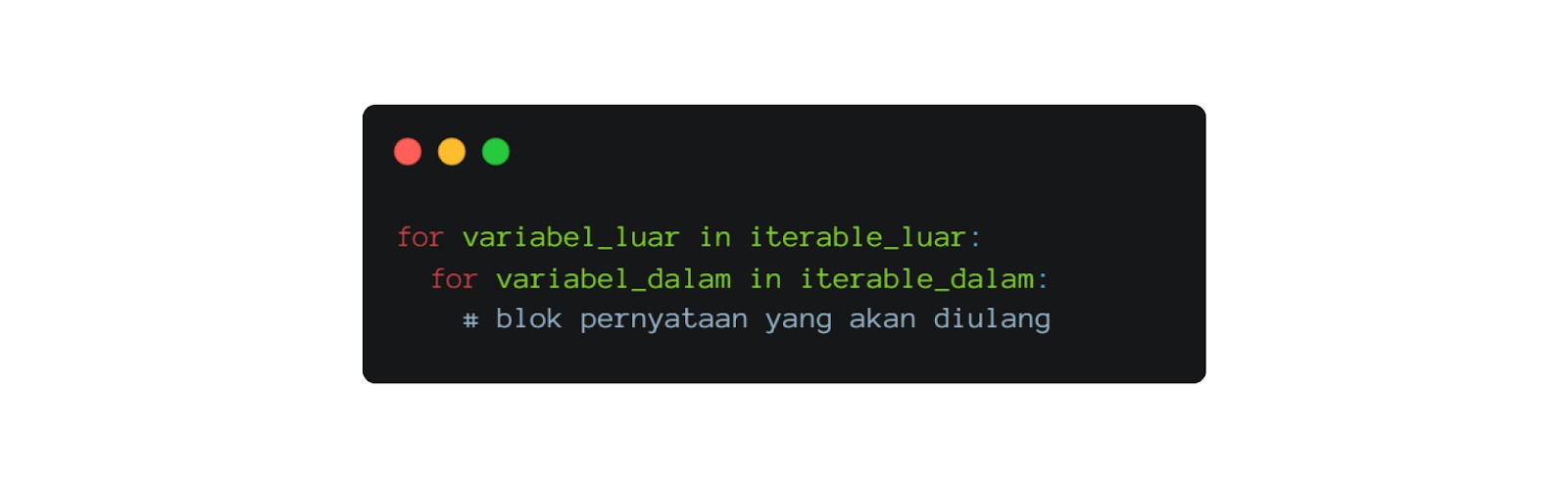
Pada contoh di atas, kita melakukan perulangan while, tetapi tidak melakukan increment di baris akhir kode. Hal ini menyebabkan program akan terus berjalan dan akhirnya berhenti karena *run time exceeded*atau waktu berjalan melebihi yang ditentukan.

Jika Anda menjalankan kode di atas pada IDE seperti notebook, program akan terus berjalan dan harus dihentikan dengan menekan **CTRL + C**.

**For Bersarang**

Ketika Anda membuat perulangan, sering kali menemukan perulangan dalam perulangan atau disebut sebagai *nested loop*.

Format dari nested loop sebagai berikut.



Anda dapat asumsikan bahwa ada dua perulangan, yakni "perulangan luar" dan "perulangan dalam". Program akan melakukan "perulangan luar" terlebih dahulu, lalu akan melakukan "perulangan dalam". "variabel\_luar" akan mengambil nilai dari "iterable\_luar", sedangkan "variabel\_dalam" akan mengambil nilai dari "iterable\_dalam".

for i in range(1, 3):

for j in range(1, 3):

print(i,j)

Output:

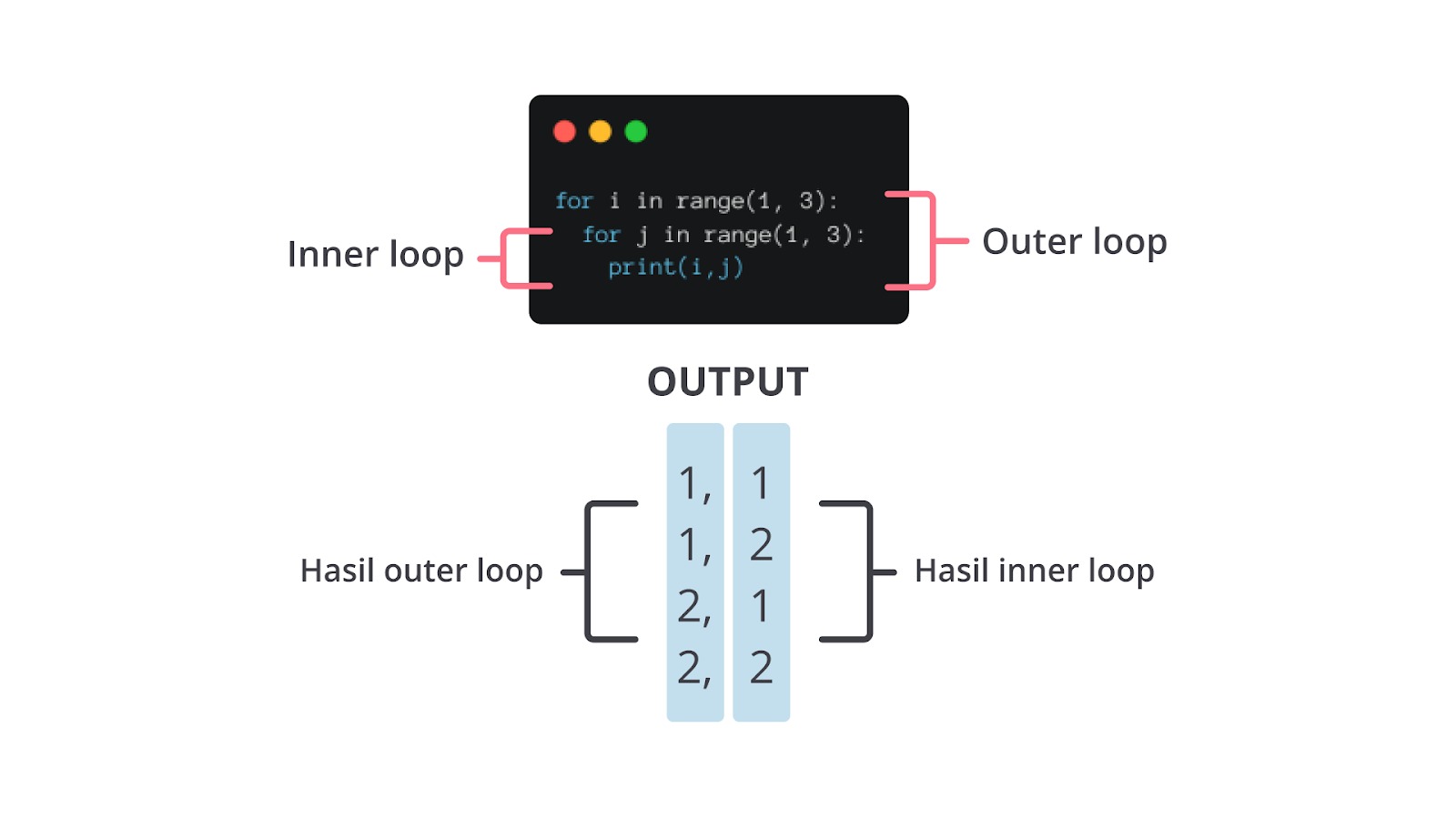
1,1

1,2

2,1

2,2

Pada contoh implementasi di atas, kita melakukan perulangan for luar dengan variabel "i" yang mengulang dari 1 hingga 2. Lalu melakukan perulangan "j" yang akan mengulang dari 1 hingga 2 juga.



Output dari sebelah kiri dihasilkan dari perulangan for luar, sedangkan output dari sebelah kanan dihasilkan dari perulangan for dalam. Perhatikan lebih detail bahwa "perulangan luar" atau outer loop akan dilanjutkan jika "perulangan dalam" atau inner loop telah selesai. Semua perulangan tersebut dilakukan hingga kedua perulangan menghasilkan false dan berhenti.

**Kontrol Perulangan**

Selain membuat perulangan, kita juga dapat mengontrol perulangan dengan menggunakan beberapa pernyataan di antaranya sebagai berikut.

**Break**

Break statement adalah pernyataan untuk menghentikan perulangan dan kemudian program akan otomatis keluar dari perulangan tersebut, lalu dilanjutkan dengan mengeksekusi blok perulangan selanjutnya. Jika Anda memiliki perulangan yang bertingkat seperti for bersarang, break akan menghentikan perulangan sesuai dengan tingkatan atau letak perulangannya berada.

for i in range(2): # Perulangan tingkat pertama

print("Perulangan luar:", i)

for j in range(10): # Perulangan tingkat kedua

print("Perulangan dalam:", j)

if j == 1:

break # Menghentikan perulangan dalam jika j = 1

Output:

Perulangan luar: 0

Perulangan dalam: 0

Perulangan dalam: 1

Perulangan luar: 1

Perulangan dalam: 0

Perulangan dalam: 1

Pada kode di atas, kita melakukan perulangan untuk menampilkan bilangan 0 hingga 1 pada "perulangan luar" atau for pertama. Lalu, kita membuat perulangan kedua untuk menampilkan bilangan dari 0 hingga 9. Namun, pada perulangan kedua atau "perulangan dalam" tersebut, kita akan melakukan break jika bertemu angka "1". Alhasil, perulangan dalam hanya akan menampilkan angka hingga 1 saja. Program akan berhenti karena ada statement break yang diberikan jika bertemu angka "1".

**Continue**

Continue statement adalah pernyataan untuk membuat iterasi berhenti, kemudian melanjutkan ke iterasi berikutnya. Continue seolah **mengabaikan**pernyataan (statement) yang berada antara continue hingga akhir blok.

for huruf in 'Dico ding':

if huruf == ' ':

continue

print('Huruf saat ini: {}'.format(huruf))

Output:

Huruf saat ini: D

Huruf saat ini: i

Huruf saat ini: c

Huruf saat ini: o

Huruf saat ini: d # Perhatikan bahwa harusnya sebelum ini ada spasi, namun dilewati.

Huruf saat ini: i

Huruf saat ini: n

Huruf saat ini: g

Pada contoh di atas, kita membuat perulangan yang sama dengan contoh sebelumnya. Namun, alih-alih ada spasi maka program akan berhenti, program akan mengabaikan spasi tersebut dan melanjutkannya pada perulangan selanjutnya.

Dalam contoh di atas, program berusaha untuk menampilkan teks "Dico ding" dengan kondisi jika bertemu " " (spasi) akan dilewati dan melakukan perulangan selanjutnya. Alhasil output yang dihasilkan adalah "d", "i", "c", "o", "d", "i", "n", "g" tanpa adanya spasi.

**Else setelah For**

Pada Python juga dikenal else setelah for yang berfungsi untuk perulangan bersifat pencarian. Else setelah for ini bisa dikatakan sebagai memberikan jalan keluar program saat pencarian tidak ditemukan.

numbers = [1, 2, 3, 4, 5]

for num in numbers:

if num == 6:

print("Angka ditemukan! Program berhenti!")

break

else:

print("Angka tidak ditemukan.")

Output:

Angka tidak ditemukan.

Pada contoh di atas, kita membuat program untuk melakukan pencarian terhadap bilangan 6. Jika bilangan 6 tersebut merupakan elemen atau nilai yang berada list, program akan berhenti dan menampilkan teks "Angka ditemukan! Program berhenti!".

Namun, pada contoh di atas angka 6 bukan merupakan elemen dari list maka program akan menampilkan teks "Angka tidak ditemukan". Apa jadinya jika program menemukan angka? Silakan ganti "if num == 6" dengan "if num == 4" atau angka lain yang merupakan nilai yang berada dalam list.

Perlu diperhatikan oleh Anda, if dan else pada contoh tersebut berkaitan walaupun berbeda blok. **Pada else setelah for, statement else tidak akan dieksekusi saat if pernah sekali saja benar.**Dengan kata lain, **break dalam if harus tidak terjadi untuk memicu else setelah for.**

**Else setelah While**

Berbeda dengan else setelah for, pada statement else setelah while,**blok statement else akan selalu dieksekusi saat kondisi pada while menjadi salah**.

count = 0

while count < 3:

print("Dicoding Indonesia")

count += 1

else:

print("Blok else dieksekusi karena kondisi pada while salah (3<3 == False).")

Output:

Dicoding Indonesia

Dicoding Indonesia

Dicoding Indonesia

Blok else dieksekusi karena kondisi pada while salah (3<3 == False).

Pada contoh di atas, perulangan while akan terus terjadi dan else tidak akan dieksekusi jika kondisi while benar. Kondisi while akan terus benar pada kode di atas ketika variabel "count" bertambah dari 1 hingga 2 dan akan berhenti ketika variabel "count" bernilai 3 karena "3<3" adalah false atau salah.

n = 10

while n > 0:

n = n - 1

if n == 7:

break

print(n)

else:

print("Loop selesai")

Output:

9

8

Pada contoh di atas, kita mencoba menampilkan angka dari 9 hingga 1. Program akan berhenti ketika angka tersebut adalah 7. Namun, lihat baik-baik bahwa else tidak tercetak di sini. Hal ini disebabkan **while tersebut masih bernilai benar walaupun program keluar karena "break"**.

**Pass**

Pass statement adalah pernyataan yang digunakan jika Anda menginginkan sebuah pernyataan atau blok pernyataan (statement), tetapi tidak ada tindakan atau program tidak melakukan apa pun.

x = 10

if x > 5:

pass

else:

print("Nilai x tidak memenuhi kondisi")

Output:

Program di atas tidak menampilkan apa pun karena jika kondisi terpenuhi, program tidak akan melakukan apa pun.

Statement pass digunakan dalam situasi-situasi ketika Python memerlukan adanya pernyataan, tetapi tidak memiliki tindakan yang perlu dilakukan pada saat itu. Biasanya itu adalah kondisi ketika Anda membutuhkan *placeholder* untuk menunjukkan bahwa tidak ada operasi yang perlu dilakukan. Hal ini dapat membantu kita mengatur struktur kode secara rapi dan memungkinkan penambahan implementasi di kemudian hari.

**List Comprehension**

Masih terkait perulangan, terkadang ada kalanya Anda perlu membuat sebuah list baru berdasarkan list yang sudah ada.

angka = [1, 2, 3, 4]

pangkat = []

for n in angka:

pangkat.append(n\*\*2)

print(pangkat)

Output:

[1, 4, 9, 16]

Pada contoh di atas, kita mencoba melakukan operasi perpangkatan dari variabel list "angka". Hasil dari operasi tersebut kemudian disimpan pada variabel baru bernama "pangkat". Anda menggunakan fungsi ".append()" untuk menambahkan nilai baru ke dalam variabel "pangkat".

Namun, alih-alih membuat kode program seperti di atas. Anda dapat melakukan hal berikut.

angka = [1, 2, 3, 4]

pangkat = [n\*\*2 for n in angka]

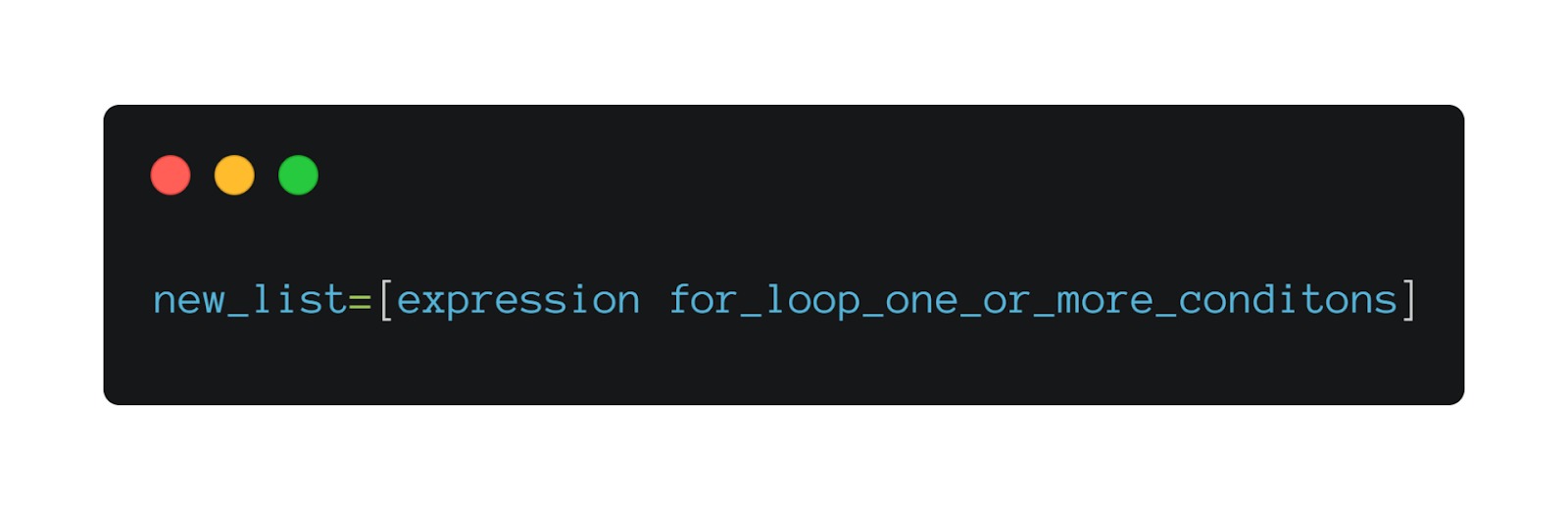
print(pangkat)

Output:

[1, 4, 9, 16]

Pada kode di atas, kita melakukan perulangan dengan memasukkan operasi perulangan tersebut ke dalam inisialisasi variabel "pangkat". Hal ini memudahkan kita sehingga tidak perlu menggunakan fungsi ".append()" untuk menambahkan nilai baru.

Konsep ini disebut sebagai list comprehension, sebuah cara untuk menghasilkan list baru berdasarkan list atau *iterables* yang telah ada sebelumnya. Sintaks dasarnya adalah berikut.



Mari bedah satu per satu struktur tersebut.

1. **new\_list**merupakan variabel yang dideklarasikan oleh Anda.
2. **expression**merupakan ekspresi yang akan dijalankan seiring perulangan bernilai benar.

**for\_loop\_one\_or\_more\_conditions**merupakan perulangan for yang Anda definisikan. Misalnya "for n in angka" yang ada pada contoh sebelumnya.

**Penanganan Kesalahan (Error Handling and Exception Handling)**

Saat Anda membuat program, sering kali menemukan setidaknya dua jenis kesalahan berdasarkan kejadiannya.

1. Kesalahan sintaks (*syntax errors*) atau sering disebut juga sebagai kesalahan penguraian (*parsing errors*).
2. Pengecualian (*exceptions*) atau sering disebut juga sebagai kesalahan saat beroperasi (*runtime* *errors*).

Mari kita bahas satu per satu jenis kesalahan tersebut.

**Kesalahan Sintaks (Syntax Errors)**

Kesalahan sintaks (*syntax errors*) adalah jenis kesalahan yang terjadi ketika Python tidak mengerti perintah Anda. Ini mengakibatkan pesan kesalahan muncul sebelum program tersebut berjalan.

Salah satu tipe kesalahan sintaks yang sering ditemui adalah kesalahan indentasi (**IndentationError**). Berikut contoh kodenya.

if 9>10:

print("Hello World!")

Output:

File "/home/glot/main.py", line 2

print("Hello World!")

^

IndentationError: expected an indented block

Pada contoh di atas, program memunculkan pesan error "IndentationError" karena seharusnya terdapat indentasi sebelum sintaks "print()". Perhatikan bahwa secara langsung Python menunjukkan baris indentasi dengan memberi tanda panah ke atas atau dikenal dengan caret "^".

Tipe kesalahan yang sering dijumpai selanjutnya adalah kesalahan sintaksis (**SyntaxErrors***)*. Berikut contoh kodenya.

i = 1

while i<3

print("Dicoding")

i+=1

Output:

File "/home/glot/main.py", line 2

while i<3

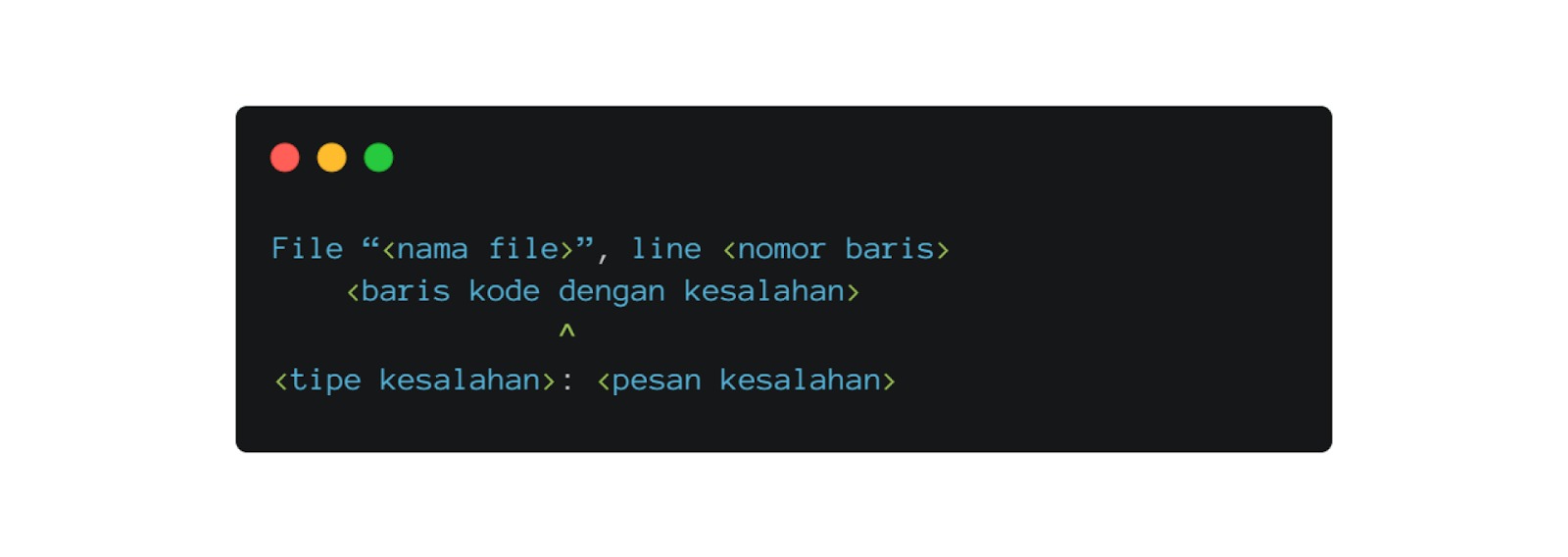
^

SyntaxError: invalid syntax

Pada tipe kesalahan sintaksis (*syntax errors*) program akan memunculkan pesan error "SyntaxError". Program pun memberi petunjuk bahwa terdapat sintaks yang tidak valid berada di posisi i<3. Dapatkah Anda menemukan kesalahannya? Ya, kesalahannya adalah tidak terdapat tanda titik dua ":" setelah while statement.

Jika kita telaah lebih dalam, kedua contoh tersebut memiliki tipe kesalahan yang berbeda. Contoh pertama menampilkan kesalahan *indentation error*dan kedua adalah *syntax error*. Kemudian setelah penyebutannya terdapat pesan detail kesalahan atau keterangan yang diberikan, misalnya sintaks yang tidak valid.

Secara umum struktur kesalahan sintaks sebagai berikut.



Mari bedah satu per satu poin di atas.

1. "<nama file>" merupakan file Python yang Anda eksekusi. Jika Anda menggunakan mode script melalui lokal komputer dan program Anda menghasilkan Error, pesan ini akan memunculkan nama script atau file Python Anda.
2. <nomor baris> merupakan nomor baris kode dalam file Anda yang mengalami kesalahan.
3. <baris kode> merupakan kode yang mengalami kesalahan dalam file Anda.
4. <tipe kesalahan> merupakan kelompok atau tipe kesalahan yang Anda alami, contohnya SyntaxErrordan IndentationError.
5. <pesan kesalahan> merupakan pesan detail kesalahan atau keterangan yang diberikan oleh program. Contohnya adalah “invalid syntax” dan “expected an indented block”.

**Pengecualian (Exceptions)**

Pengecualian adalah kesalahan yang terjadi ketika Python mengerti perintah Anda, tetapi mendapatkan masalah saat mengikutinya. Umumnya, pengecualian bisa terjadi ketika aplikasi sudah mulai beroperasi.

Jenis kesalahan ini paling sering ditemui ketika Anda membuat kode program yang kompleks. Meskipun kode atau ekspresi dari Python yang Anda tulis sudah benar, ada kemungkinan terjadi kesalahan ketika perintah tersebut dieksekusi.

Mari lihat contoh kode pengecualian di bawah ini.

print(angka)

Output:

Traceback (most recent call last):

File "/home/glot/main.py", line 1, in <module>

print(angka)

NameError: name 'angka' is not defined

Contoh kode di atas merupakan tipe kesalahan yang menunjukkan bahwa program tersebut tidak mendefinisikan variabel "angka". Jenis kesalahan ini terjadi ketika Anda tidak melakukan deklarasi dan inisialisasi variabel, tetapi langsung memanggil variabel tersebut. Program juga menampilkan keterangan atau pesan detail setelah memberi tahu tipe kesalahannya.

Mari lihat contoh pengecualian selanjutnya.

bukan\_angka = '1'

bukan\_angka + 2

Output:

Traceback (most recent call last):

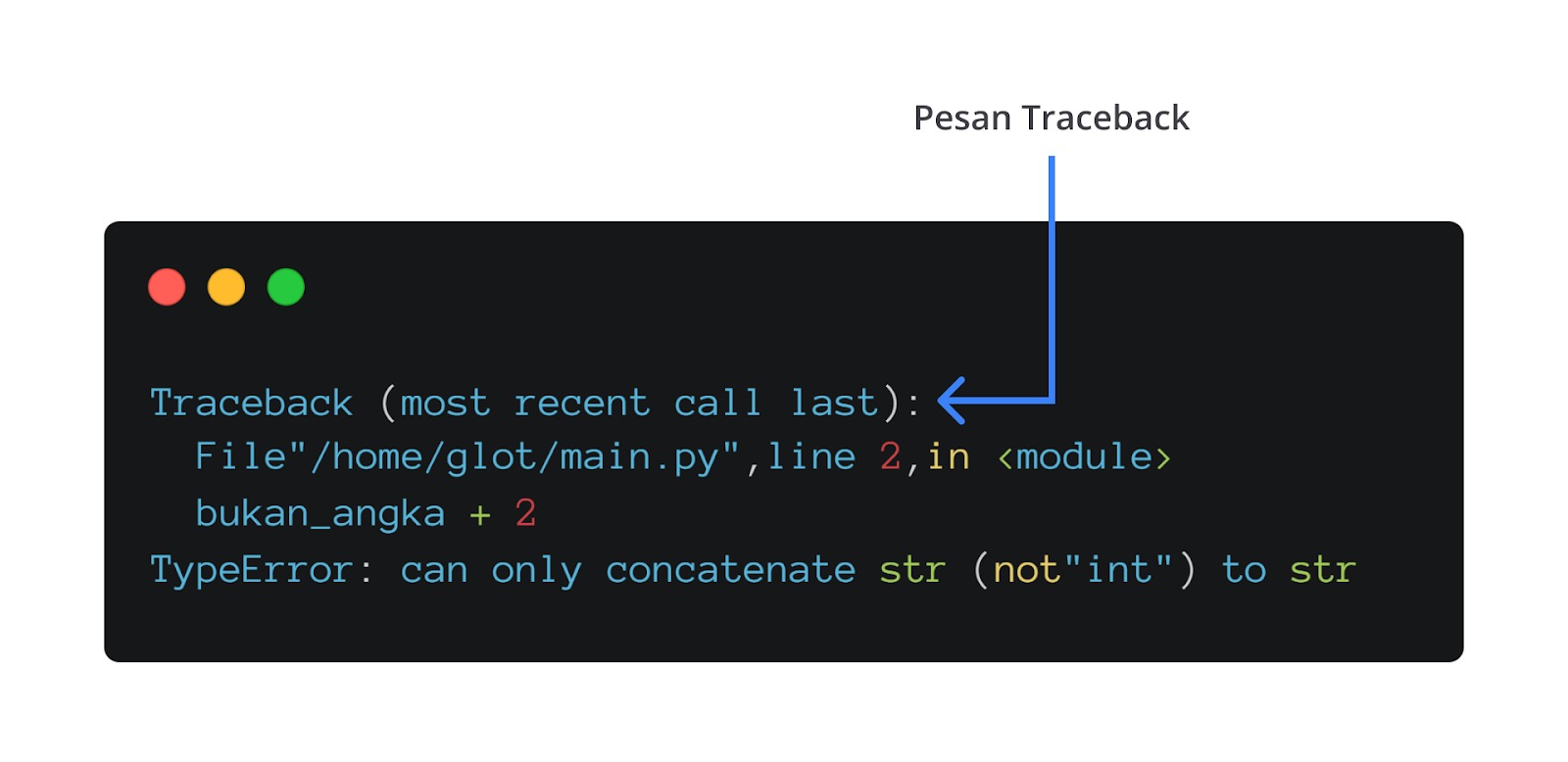
File "/home/glot/main.py", line 2, in <module>

bukan\_angka + 2

TypeError: can only concatenate str (not "int") to str

Pada contoh di atas, program memunculkan kesalahan karena variabel "bukan\_angka" termasuk tipe data string. Di sisi lain, program berusaha melakukan operasi aritmetika yang mengharuskan kedua variabel atau operan bertipe data integer. Pesan tipe kesalahan yang dimunculkan adalah TypeError dengan keterangan atau pesan detailnya adalah "can only concatenate str (not "int") to str".

Jika Anda telaah lebih dalam, struktur pengecualian hampir sama dengan struktur kesalahan sintaks. Namun, terdapat perbedaan di sana, pengecualian memberikan pesan "traceback (most recent call last)".



Pesan ini mengacu pada informasi yang ditampilkan ketika terjadi kesalahan atau pengecualian (*exception*). Pesan *traceback* ini menyediakan "jejak" dari kode yang dieksekusi sehingga Anda dapat melacak kembali jalur eksekusi program sebelum mencapai titik error.

**Penanganan Pengecualian**

Lalu, bagaimana menangani kesalahan atau pengecualian tersebut? Program Python yang Anda bangun dapat dilengkapi penanganan terhadap pengecualian dari tipe kesalahan yang Anda tentukan. Konsep ini dikenal dengan *exceptions handling* yang menggunakan pernyataan **try-except**untuk menangani pengecualian tersebut.

Mari lihat jenis pengecualian yang berbeda dari contoh sebelumnya. Kali ini, pengecualiannya adalah kesalahan pembagian angka dengan nol.

z = 0

print(1/z)

Output:

Traceback (most recent call last):

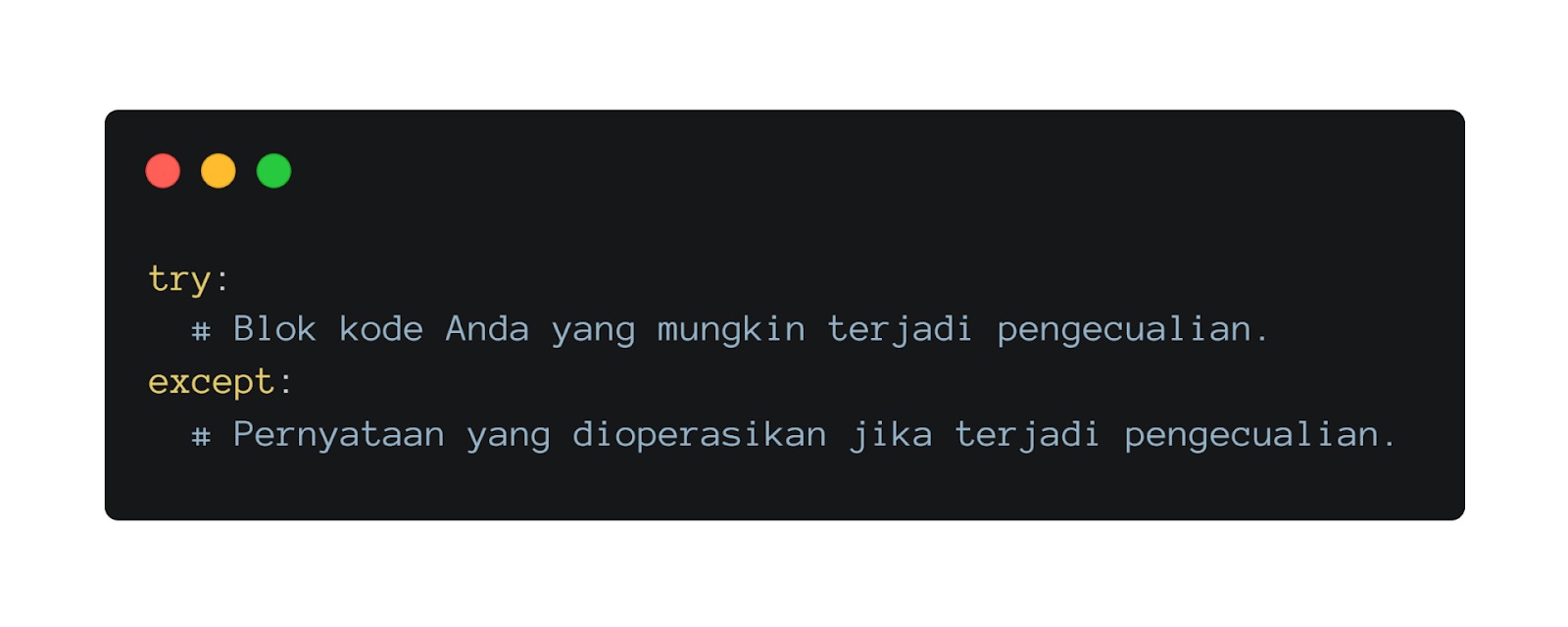
File "/home/glot/main.py", line 2, in <module>

print(1/z)

ZeroDivisionError: division by zero

Pada contoh di atas, program memunculkan error karena melakukan operasi aritmetika pembagian dengan nilai nol. Kita tahu bahwa pembagian dengan nilai nol tidak bisa dilakukan karena tidak terdefinisikan (*undefined*). Perhatikan bahwa tipe pengecualian yang terjadi adalah "ZeroDivisionError".

Kita bisa lakukan penanganan terhadap pengecualian tersebut dengan menggunakan pernyataan **try-except**. Berikut strukturnya.



Anda bisa simpan kode yang dibuat di dalam "try" statement. Kode tersebut merupakan barisan kode yang mungkin terjadi pengecualian. Ada pun statement "except" adalah statement yang akan digunakan ketika pengecualian tersebut terjadi.

Sederhananya, program akan mencoba (try) mengeksekusi kode yang berada dalam try statement. Lalu jika terjadi pengecualian, kode yang berada pada except statement akan dieksekusi.

z=0

try:

print(1/z)

except ZeroDivisionError:

print("Anda tidak bisa membagi angka dengan nilai nol.")

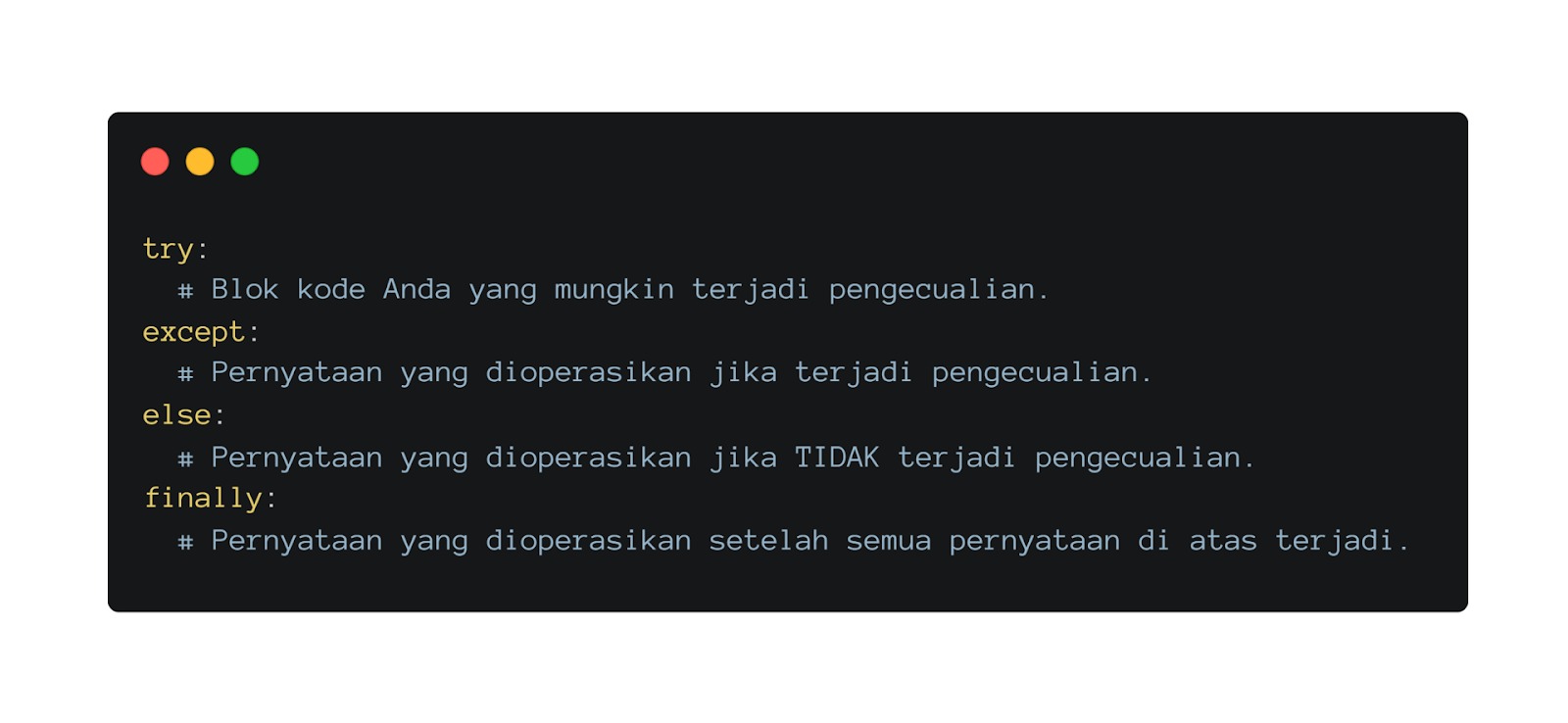
Output:

Anda tidak bisa membagi angka dengan nilai nol.

Pada contoh di atas, program akan menjalankan try statement terlebih dahulu dengan mengeksekusi sintaks "print(1/z)". Kita tahu bahwa program tersebut akan mengalami error "ZeroDivisionError". Alih-alih program menampilkan pesan "**ZeroDivisionError: division by zero**", kita ingin program menampilkan teks "Anda tidak bisa membagi angka dengan nilai nol.”

Kita simpan pernyataan "ZeroDivisionError" setelah **except statement**. Dengan begini, kita memerintahkan program untuk mengeksekusi kode dalam except **jika** pengecualian "ZeroDivisionError" terjadi. Kode dalam except tersebut menampilkan pesan "Anda tidak bisa membagi angka dengan nilai nol."

Sebenarnya, struktur lengkap dari pernyataan try tidak hanya melibatkan except. Berikut adalah struktur lengkap pernyataan try.



Struktur lengkap seperti ini biasanya diterapkan ketika Anda membangun program atau aplikasi yang lebih kompleks.

Pada try statement, program akan menjalankan blok kode yang mungkin terjadi pengecualian. Pada except statement, program akan mengeksekusi statement ini jika terjadi pengecualian. Pada else statement, program akan mengeksekusi statement ini jika **tidak** terjadi pengecualian. Pada finally statement, program akan mengeksekusi statement ini setelah semua pernyataan di atas terjadi.

Mari lihat contoh penerapannya di bawah ini, contoh tersebut merupakan program untuk menampilkan key dalam tipe data dictionary. Jenis pengecualian yang akan dilakukan adalah **KeyError** dan **TypeError.** Kita akan coba untuk mengakses **value** dictionary dari **key**yang tidak ada dan mencoba mengoperasikan **value** dictionary tersebut yang termasuk string dengan operasi aritmetika. Untuk pengingat, **dictionary** merupakan tipe data yang melibatkan **key-value**.

Mari mulai dengan kondisi yang tidak menampilkan error.

var\_dict = {"rata\_rata": "1.0"}

try:

print(f"rata-rata adalah {var\_dict['rata\_rata']}")

except KeyError:

print("Key tidak ditemukan.")

except TypeError:

print("Anda tidak bisa membagi nilai dengan tipe data string")

else:

print("Kode ini dieksekusi jika tidak ada exception.")

finally:

print("Kode ini dieksekusi terlepas dari ada atau tidaknya exception.")

Output:

rata-rata adalah 1.0

Kode ini dieksekusi jika tidak ada exception.

Kode ini dieksekusi terlepas dari ada atau tidaknya exception.

Pada contoh di atas, langkah pertama yang dilakukan adalah melakukan inisialisasi variable **"var\_dict"** dengan nilai dictionary. Lalu, kita mencoba mengakses **value** dari **key**"rata\_rata". Program di atas tidak menimbulkan error karena kita memberikan sintaks yang benar untuk mengakses **value** berdasarkan **key**, yakni **var\_dict['rata\_rata']**.

Perlu diperhatikan bahwa **else statement** ikut dieksekusi disebabkan pengecualian tidak terjadi (**Error Exception tidak ada**). Selain itu, **finally statement**akan selalu dieksekusi baik jika ada pengecualian ataupun tidak.

Bagaimana membangkitkan **KeyError** dan **TypeError**? Perhatikan langkah berikut.

1. Untuk membangkitkan KeyError, ubah kode "**print(f"rata-rata adalah {var\_dict['rata\_rata']}")**" dengan kode " **print(f"rata-rata adalah {var\_dict['ratarata']}")** ". Anda bisa langsung *copy-paste*kode tersebut.
   1. Dengan mengubah kode tersebut, program akan menampilkan **KeyError**. Hal ini terjadi karena kita mengakses key **"ratarata"**, padahal seharusnya adalah **"rata\_rata"**.
   2. Perhatikan bahwa else statement tidak dieksekusi, karena *exception* atau pengecualian terjadi. Hal ini dibuktikan dengan tidak adanya teks "Kode ini dieksekusi jika tidak ada exception".
2. Untuk membangkitkan TypeError, ubah kode "**print(f"rata-rata adalah {var\_dict['rata\_rata']}")**" dengan kode " **print(f"rata-rata adalah {var\_dict['rata\_rata']/2}")** ". Anda bisa langsung *copy-paste*kode tersebut.
   1. Dengan mengubah kode tersebut, program akan menampilkan **TypeError**. Hal ini terjadi karena kita melakukan operasi aritmetika terhadap value dari key **"rata\_rata"** yang merupakan tipe data string.
   2. Perhatikan bahwa else statement tidak dieksekusi, karena *exception* atau pengecualian terjadi. Hal ini dibuktikan dengan tidak adanya teks "Kode ini dieksekusi jika tidak ada exception".

**Raise Exception**

Jika sebelumnya kita menangani kesalahan yang **TIDAK DISENGAJA**, kali ini kita akan mempelajari cara menangani kesalahan yang **DISENGAJA**. Umumnya, ketika membuat kode program kita ingin membatasi program tersebut dengan kondisi tertentu.

Perlu diingat bahwa umumnya, raise digunakan bersamaan dengan if-else statement.

Misalnya, Anda ingin membuat kode program untuk menampilkan angka dari 1 hingga 10 berdasarkan input atau masukan pengguna. Namun, dalam program tersebut kita ingin mengontrol dengan cara berikut: jika user memberikan input berupa bilangan negatif, program akan memunculkan pesan error dengan keterangan "Bilangan negatif tidak diperbolehkan".

Mari lihat penerapannya di bawah ini. Asumsikan user memasukkan input berupa bilangan "-1".

var = -1

if var < 0:

raise ValueError("Bilangan negatif tidak diperbolehkan")

else:

for i in range(var):

print(i+1)

Traceback (most recent call last):

File "/home/glot/main.py", line 3, in <module>

raise ValueError("Bilangan negatif tidak diperbolehkan")

ValueError: Bilangan negatif tidak diperbolehkan

Pada contoh di atas, kita menggunakan percabangan untuk melakukan evaluasi jika nilai variabel "var" adalah bilangan negatif (kurang dari 0), program akan menampilkan error dengan teks "Bilangan negatif tidak diperbolehkan". Selain itu, program akan mengeksekusi else statement jika nilai dari variabel "var" bukanlah bilangan negatif (lebih besar atau sama dengan 0).

Ke depannya, Anda bisa mengontrol flow program yang Anda bangun dengan melibatkan error dan exceptions handling. Terutama ketika Anda membangun program yang lebih kompleks.