

# Modul 4 Praktikum Algoritma Pemrograman

Topik Praktikum : Perulangan (Iterasi/Looping) dalam Bahasa Pemrograman Python 3

Tujuan Praktikum:

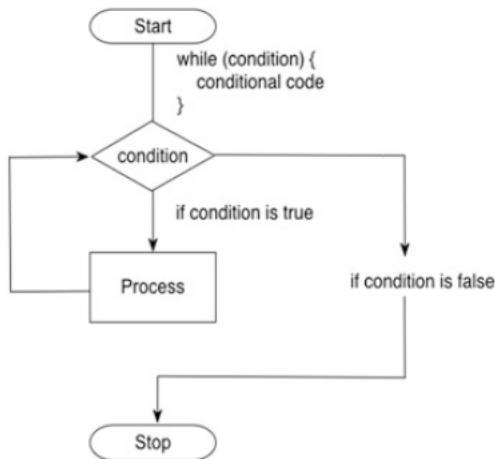
1. Mahasiswa dapat memahami dan menggunakan perulangan while
2. Mahasiswa dapat memahami dan menggunakan perulangan For
3. Mahasiswa dapat memahami dan menggunakan algoritma control flow dengan perulangan dan pengkondisian

## 1 Pendahuluan

Dalam pemrograman, terdapat berbagai cara untuk mengotomatisasi tugas yang melibatkan eksekusi berulang, dan salah satu cara paling efektif adalah melalui penggunaan loop. Di Python, terdapat dua jenis loop utama yang sering digunakan, yaitu while loop dan for loop. Kedua loop ini memiliki peran penting dalam mengontrol jalannya eksekusi perintah secara berulang, berdasarkan kondisi tertentu atau elemen-elemen dalam sebuah koleksi data. Melalui while loop, eksekusi berulang dilakukan selama suatu kondisi masih terpenuhi, sedangkan for loop memungkinkan kita untuk mengulang eksekusi berdasarkan iterasi di dalam sebuah urutan atau objek yang dapat diulang (iterable). Pemahaman yang baik tentang cara kerja kedua loop ini sangat penting untuk membantu menyelesaikan berbagai masalah pemrograman yang membutuhkan otomatisasi proses berulang secara efisien.

## 2 While Loop

While loop ada di hampir semua bahasa pemrograman dan digunakan untuk melakukan iterasi (atau pengulangan) satu atau lebih pernyataan kode selama kondisi (ekspresi) pengujian bernilai True. Struktur iterasi ini biasanya digunakan ketika jumlah pengulangan blok kode yang diperlukan tidak diketahui. Misalnya, kode mungkin perlu diulang sampai suatu solusi ditemukan atau sampai pengguna memasukkan nilai tertentu. Perilaku dari while loop diilustrasikan dalam diagram di bawah ini.



*While loop* dalam Python memiliki bentuk dasar sebagai berikut:

```
while <kondisi-tes=true>:
    pernyataan atau pernyataan-pernyataan
```

Seperti yang ditunjukkan dalam diagram dan bisa disimpulkan dari kode: selama kondisi/ekspresi tes bernilai **True**, maka pernyataan atau blok pernyataan dalam *while loop* akan dieksekusi.

Perlu dicatat bahwa tes dilakukan sebelum setiap iterasi, termasuk iterasi pertama. Oleh karena itu, jika kondisi gagal pada putaran pertama, pernyataan atau blok pernyataan mungkin tidak akan dieksekusi sama sekali.

Sama seperti pada pernyataan *if*, indentasi adalah kunci di sini karena pernyataan yang termasuk dalam pernyataan *while* ditentukan oleh indentasi. Setiap pernyataan yang diindentasi pada level yang sama setelah kondisi *while* adalah bagian dari *while loop*.

Namun, segera setelah sebuah pernyataan tidak lagi mengikuti blok *while*, maka pernyataan itu bukan lagi bagian dari *while loop* dan eksekusinya tidak lagi berada di bawah kendali kondisi tes.

Berikut ini adalah contoh *while loop* dalam Python:

```
count = 0
print('Starting')
while count < 10:
    print(count, ' ', end='') # bagian dari while loop
    count += 1 # juga bagian dari while loop
print() # bukan bagian dari while loop
print('Done')
```

Dalam contoh ini, selama variabel *count* kurang dari 10, *while loop* akan terus diulang (diulang berulang kali). *While loop* itu sendiri berisi dua pernyataan; yang pertama mencetak nilai variabel *count*, dan yang lainnya menambah nilai *count* (ingat bahwa *count*  $\doteq$  1 setara dengan *count* = *count* + 1).

Kita menggunakan versi dari fungsi *print()* yang tidak mencetak baris baru saat mencetak nilai (ini ditunjukkan oleh opsi *end=''* yang diberikan ke fungsi *print()*).

Hasil dari menjalankan contoh ini adalah:

```
Starting
0 1 2 3 4 5 6 7 8 9
Done
```

Seperti yang Anda lihat, pernyataan yang mencetak pesan ‘Starting’ dan ‘Done’ hanya dijalankan sekali. Namun, pernyataan yang mencetak variabel `count` dijalankan 10 kali (mencetak nilai dari 0–9).

Begitu nilai dari `count` mencapai 10, loop selesai (atau berhenti). Perlu dicatat bahwa kita perlu menginisialisasi variabel `count` sebelum loop dimulai. Ini karena nilai tersebut diperlukan untuk iterasi pertama dari *while loop*. Sebelum *while loop* melakukan apa pun, program harus sudah mengetahui nilai awal `count` agar bisa melakukan pengujian pertama tersebut. Ini adalah salah satu karakteristik dari perilaku *while loop*.

### 3 For Loop

Seringkali, kita mengetahui jumlah pasti iterasi yang diperlukan untuk satu atau lebih pernyataan. Meskipun *while loop* bisa digunakan, *for loop* menawarkan cara yang lebih ringkas dan jelas, terutama bagi programmer lain, karena menunjukkan dengan pasti jumlah iterasi yang dibutuhkan.

*For loop* digunakan untuk mengatur sebuah variabel melalui serangkaian nilai hingga suatu tes tertentu terpenuhi. Perilaku dari *for loop* diilustrasikan di bawah ini.

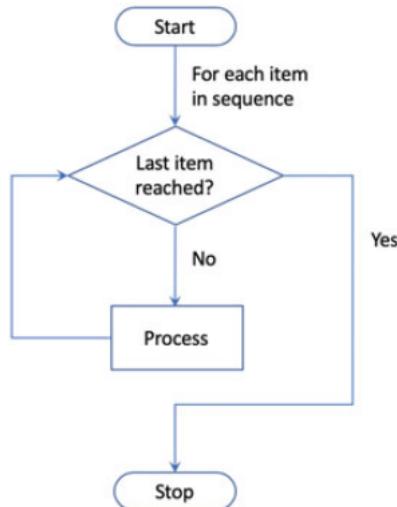


Diagram alir ini menunjukkan bahwa beberapa urutan nilai (misalnya semua nilai integer antara 0 dan 9) akan digunakan untuk mengiterasi sebuah blok kode untuk diproses. Ketika item terakhir dalam urutan telah dicapai, loop akan berakhir.

Banyak bahasa memiliki *for loop* dalam bentuk:

```
for i = from 0 to 10
    pernyataan atau pernyataan-pernyataan
```

Dalam hal ini, variabel `i` akan mengambil nilai 0, 1, 2, 3, dan seterusnya hingga 10.

Di Python, pendekatannya sedikit berbeda karena nilai-nilai dari 0 hingga 10 diwakili oleh sebuah **range**. Ini sebenarnya adalah sebuah fungsi yang akan menghasilkan rentang nilai untuk digunakan sebagai urutan dalam *for loop*. Karena *for loop* di Python sangat fleksibel dan dapat mengiterasi tidak hanya melalui rentang nilai integer tetapi juga sekumpulan nilai yang disimpan dalam struktur data seperti daftar integer atau string.

Format dari *for loop* di Python saat menggunakan rentang nilai adalah:

```
for <nama-variabel> in range(...):
    pernyataan
    pernyataan
```

Contoh di bawah ini setara dengan *while loop* yang kita lihat sebelumnya:

```
# Loop di atas sekumpulan nilai dalam range
print('Cetak nilai-nilai dalam rentang')
for i in range(0, 10):
    print(i, ' ', end=' ')
print()
print('Selesai')
```

Ketika kita menjalankan kode ini, hasilnya adalah:

```
Cetak nilai-nilai dalam rentang
0 1 2 3 4 5 6 7 8 9
Selesai
```

Seperti yang dapat dilihat dari hasil di atas; hasil akhirnya adalah kita menghasilkan *for loop* yang menghasilkan sekumpulan nilai yang sama seperti *while loop* sebelumnya. Namun,

- Kode lebih ringkas,
- Jelas bahwa kita memproses rentang nilai dari 0 hingga 9 (perhatikan bahwa nilainya adalah hingga tetapi tidak termasuk nilai akhir), dan
- Kita tidak perlu mendefinisikan variabel loop terlebih dahulu.

Untuk alasan-alasan ini, *for loops* lebih sering digunakan dalam program pada umumnya dibandingkan *while loops*.

Dalam *while loop*, kita tidak harus menambah variabel `count` sebesar satu. Misalnya, kita bisa menambah `count` sebesar 2 setiap kali loop berulang. Fungsi **range** memungkinkan hal ini dengan menyediakan argumen ketiga sebagai nilai kenaikan variabel loop, misalnya:

```
# Sekarang gunakan nilai dalam range tetapi tambahkan 2
print('Cetak nilai-nilai dalam rentang dengan penambahan 2')
for i in range(0, 10, 2):
    print(i, ' ', end=' ')
print()
print('Selesai')
```

Ketika kita menjalankan kode ini, hasilnya adalah:

```
Cetak nilai-nilai dalam rentang dengan penambahan 2
0 2 4 6 8
Selesai
```

Dengan demikian, nilai dari variabel loop telah melompat sebesar 2 dimulai dari 0. Begitu nilainya mencapai 10 atau lebih, loop berhenti. Tentu saja, bukan hanya nilai 2 yang bisa kita gunakan; kita bisa menambah dengan integer yang bermakna lainnya, seperti 3, 4, atau 5.

Satu variasi menarik pada *for loop* adalah penggunaan **wild card** (karakter '\_') sebagai pengganti variabel looping; Hal ini bisa berguna jika Anda hanya tertarik untuk mengulang sejumlah kali tertentu dan tidak peduli dengan nilai variabel loop itu sendiri, misalnya:

```
# Sekarang gunakan variabel loop 'anonim'
for _ in range(0, 10):
    print('..', end=' ')
print()
```

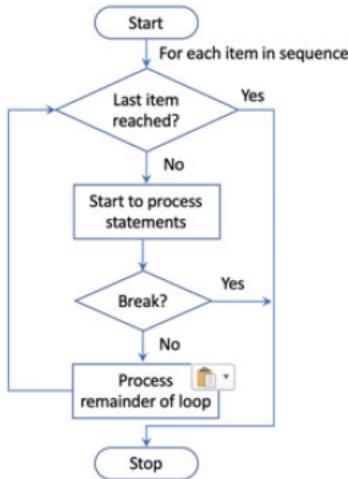
Dalam hal ini, kita tidak tertarik dengan nilai yang dihasilkan oleh rentang itu sendiri, hanya dalam mengulang 10 kali sehingga tidak ada manfaat dalam merekam variabel loop. Variabel loop diwakili oleh karakter garis bawah ('\_').

Perlu dicatat bahwa cara ini menggunakan variabel yang valid dan dapat dirujuk dalam loop; namun, secara konvensi, variabel tersebut dianggap anonim.

## 4 Break Loop Statement

Python memungkinkan programmer untuk memutuskan apakah mereka ingin keluar dari loop lebih awal atau tidak (baik menggunakan *for loop* atau *while loop*). Hal ini dilakukan menggunakan pernyataan **break**.

Pernyataan **break** memungkinkan pengembang untuk mengubah siklus normal loop berdasarkan kriteria tertentu yang mungkin tidak dapat diprediksi sebelumnya (misalnya, berdasarkan input dari pengguna). Saat pernyataan **break** dieksekusi, loop saat ini akan dihentikan, dan program akan melompat ke baris pertama setelah loop. Diagram berikut menunjukkan cara kerjanya pada *for loop*:



Biasanya, pernyataan penjaga (*if statement*) ditempatkan pada **break** sehingga pernyataan **break** diterapkan secara kondisional ketika diperlukan.

Contoh berikut menunjukkan aplikasi sederhana di mana pengguna diminta memasukkan angka yang mungkin berada dalam jangkauan yang didefinisikan oleh *for loop*. Jika angka tersebut berada dalam jangkauan, maka kita akan melakukan loop hingga nilai tersebut tercapai dan kemudian *break* loop (yaitu menghentikan lebih awal tanpa memproses sisa nilai dalam loop):

```

print('Only print code if all iterations completed')
num = int(input('Enter a number to check for: '))
for i in range(0, 6):
    if i == num:
        break
    print(i, ' ', end=' ')
print('Done')

```

Jika kita menjalankan ini dan memasukkan nilai 7 (yang berada di luar jangkauan), maka semua nilai dalam loop akan dicetak:

```

Enter a number to check for: 7
0 1 2 3 4 5 Done

```

Namun, jika kita memasukkan nilai 3, hanya nilai 0, 1, dan 2 yang akan dicetak sebelum loop dihentikan lebih awal:

```

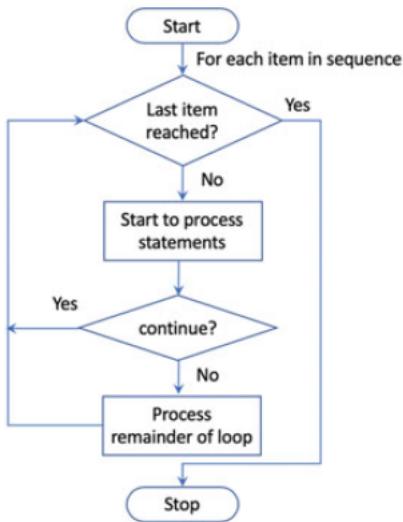
Enter a number to check for: 3
0 1 2 Done

```

Perhatikan bahwa string 'Done' dicetak dalam kedua kasus karena merupakan baris pertama setelah *for loop* yang tidak diindentasi, sehingga tidak menjadi bagian dari loop. Pernyataan **break** dapat ditempatkan di mana saja dalam blok kode loop, baik *for* maupun *while*, sehingga pernyataan lain bisa berada sebelum atau sesudahnya.

## 5 Continue Loop Statement

Pernyataan **continue** juga mempengaruhi alur kontrol dalam konstruksi loop *for* dan *while*. Namun, pernyataan ini tidak menghentikan keseluruhan loop; sebaliknya, pernyataan ini hanya menghentikan iterasi saat ini dari loop. Hal ini memungkinkan kita melewati sebagian iterasi loop untuk nilai tertentu, namun tetap melanjutkan dengan nilai-nilai lainnya dalam urutan.



Pernyataan penjaga (misalnya *if statement*) dapat digunakan untuk menentukan kapan pernyataan **continue** harus dijalankan.

Sama seperti pernyataan **break**, pernyataan **continue** dapat ditempatkan di mana saja dalam badan konstruksi loop. Yang berarti ada pernyataan yang akan dieksekusi untuk setiap nilai dalam urutan dan beberapa pernyataan yang hanya akan dijalankan jika pernyataan **continue** tidak dieksekusi.

Contohnya, dalam program ini, pernyataan **continue** hanya dijalankan untuk bilangan ganjil, sehingga dua pernyataan **print()** hanya dijalankan jika nilai *i* adalah bilangan genap:

```
for i in range(0, 10):
    print(i, ' ', end=' ')
    if i % 2 == 1:
        continue
    print('hey its an even number')
    print('we love even numbers')
print('Done')
```

Ketika kita menjalankan kode ini, hasilnya adalah:

```
0 hey its an even number
we love even numbers
1
2 hey its an even number
we love even numbers
3
```

```

4 hey its an even number
we love even numbers
5
6 hey its an even number
we love even numbers
7
8 hey its an even number
we love even numbers
9
Done

```

Seperti yang bisa kita lihat, pesan tentang bilangan genap hanya dicetak ketika nilai-nilai tersebut adalah 0, 2, 4, 6, dan 8.

## 6 For Loop dengan Else

Sebuah *for loop* dapat memiliki blok *else* opsional di akhir loop. Bagian *else* akan dieksekusi **hanya jika** semua item dalam urutan telah diproses. *For loop* mungkin gagal memproses semua elemen dalam loop jika terjadi kesalahan dalam program (misalnya jika ada kesalahan sintaks) atau jika kita menghentikannya loop menggunakan pernyataan **break**.

Berikut adalah contoh *for loop* dengan bagian *else*:

```

# Hanya mencetak kode jika semua iterasi selesai pada sebuah list
print('Only print code if all iterations completed')
num = int(input('Enter a number to check for: '))
for i in range(0, 6):
    if i == num:
        break
    print(i, ' ', end=' ')
else:
    print()
    print('All iterations successful')

```

Jika kita menjalankan kode ini dan memasukkan nilai integer 7 sebagai angka yang akan diperiksa, maka blok *else* akan dieksekusi karena tes *if* di dalam pernyataan *for* tidak pernah bernilai **True** dan loop tidak pernah dihentikan, sehingga semua nilai diproses:

```

Only print code if all iterations completed
Enter a number to check for: 7
0 1 2 3 4 5
All iterations successful

```

Namun, jika kita memasukkan nilai 3 sebagai angka yang akan diperiksa, maka pernyataan *if* akan bernilai **True** saat variabel loop *i* memiliki nilai 3. Akibatnya, hanya nilai 0, 1, dan 2 yang akan diproses oleh loop. Dalam situasi ini, bagian *else* tidak akan dieksekusi karena tidak semua nilai dalam urutan diproses:

```

Only print code if all iterations completed
Enter a number to check for: 3
0 1 2

```

## 7 Catatan tentang Penamaan Variabel Loop

Pada modul sebelumnya, kita menyebutkan bahwa nama variabel harus bermakna, dan penggunaan nama seperti ‘a’ dan ‘b’ biasanya kurang tepat. Pengecualian untuk aturan ini adalah pada variabel loop dalam *for loop* yang menggunakan rentang nilai, di mana nama seperti ‘i’ dan ‘j’ sangat umum.

Konvensi ini begitu umum sehingga orang cenderung menganggap variabel ‘i’ atau ‘j’ sebagai variabel loop. Oleh karena itu:

- Gunakan nama ini untuk variabel loop, dan
- Hindari menggunakannya di luar konteks loop.

Mengapa ‘i’ dan ‘j’? Ini berasal dari bahasa pemrograman **Fortran** yang dikembangkan pada 1950-an. Di Fortran, variabel loop harus dinamai ‘i’, ‘j’, dan seterusnya. Fortran sangat dominan di bidang pemrograman ilmiah dan matematika, sehingga konvensi ini menyebar ke bahasa lain tanpa batasan tersebut.

## 8 Permainan Lempar Dadu

Program singkat berikut mengilustrasikan bagaimana *while loop* dapat digunakan untuk mengontrol eksekusi dari kode utama. Dalam permainan ini, kita akan terus melempar sepasang dadu sampai pengguna menyatakan bahwa mereka tidak ingin melanjutkan (kita menggunakan modul **random** untuk ini, merupakan modul built-in python). Saat pengguna memutuskan untuk berhenti, *while loop* akan berakhir:

```
import random

MIN = 1
MAX = 6
roll_again = 'y'

while roll_again == 'y':
    print('Rolling the dices...')
    print('The values are....')
    dice1 = random.randint(MIN, MAX)
    print(dice1)
    dice2 = random.randint(MIN, MAX)
    print(dice2)
    roll_again = input('Roll the dices again? (y / n): ')
```

Contoh keluaran program:

```
Rolling the dices...
The values are....
2
6
Roll the dices again? (y / n): y
Rolling the dices...
The values are....
4
```

```

1
Roll the dices again? (y / n): y
Rolling the dices...
The values are.....
3
6
Roll the dices again? (y / n): n

```

Program ini akan terus melempar dadu dan menampilkan hasilnya sampai pengguna memasukkan 'n' untuk berhenti.

## 9 Dokumentasi Lainnya

- <https://docs.python.org/3/tutorial/controlflow.html> the online Python flow of control tutorial.

## 10 Latihan

### 10.1 Menghitung Faktorial dari Sebuah Angka

Tugas: - Program harus menghitung faktorial dari angka yang diberikan oleh pengguna.

- Faktorial dari angka ( n ) (ditulis sebagai ( n! )) adalah hasil perkalian semua bilangan dari 1 hingga ( n ).
- Faktorial tidak didefinisikan untuk bilangan negatif dan faktorial dari 0 adalah 1 (yaitu, ( 0! = 1 )).

Langkah-langkah yang harus dilakukan:

1. Jika angka kurang dari nol, tampilkan pesan kesalahan.
2. Jika angka adalah nol, tampilkan hasilnya sebagai 1.
3. Jika tidak, gunakan loop untuk menghitung faktorial dan cetak hasilnya.

*Penjelasan Program:*

1. Program meminta input dari pengguna dan memeriksa apakah input tersebut merupakan bilangan positif menggunakan `isnumeric()`.
2. Jika bilangan yang dimasukkan kurang dari nol, program menampilkan pesan bahwa faktorial tidak didefinisikan untuk bilangan negatif.
3. Jika angka yang dimasukkan adalah nol, hasilnya adalah 1 karena ( 0! = 1 ).
4. Untuk angka positif, program menggunakan loop untuk menghitung faktorial dengan men-galikan semua bilangan dari 1 hingga angka yang dimasukkan.

```

[1]: # Meminta input dari pengguna
num = input("Enter a positive integer: ")

# Memeriksa apakah input adalah angka positif
if not num.isnumeric():
    print("Error: Please enter a valid positive integer.")
else:
    num = int(num)

```

```

# Kasus ketika angka kurang dari nol
if num < 0:
    print("Error: Factorial is not defined for negative numbers.")

# Kasus ketika angka adalah nol
elif num == 0:
    print("The factorial of 0 is 1.")

# Menghitung faktorial menggunakan loop
else:
    factorial = 1
    for i in range(1, num + 1):
        factorial *= i
    print(f"The factorial of {num} is {factorial}.")

```

The factorial of 5 is 120.

## 10.2 Mencetak Semua Bilangan Prima dalam Rentang

Bilangan prima adalah bilangan yang hanya dapat dibagi oleh 1 dan dirinya sendiri. Contohnya, bilangan 2, 3, 5, dan 7 adalah bilangan prima karena tidak dapat dibagi oleh bilangan bulat lainnya. Namun, bilangan seperti 4 dan 6 bukan bilangan prima karena bisa dibagi oleh 2 (dan 6 juga bisa dibagi oleh 3).

Anda diminta untuk menulis program yang menghitung bilangan prima dari angka 1 hingga angka yang diinput oleh pengguna.

Jika pengguna memasukkan angka di bawah 2, tampilkan pesan kesalahan. Untuk setiap angka lebih besar dari 2, gunakan loop untuk memeriksa apakah angka tersebut dapat dibagi oleh bilangan lain (Anda mungkin perlu menggunakan dua for loop; satu di dalam yang lain). Untuk setiap angka yang tidak dapat dibagi oleh bilangan lain (artinya angka tersebut adalah bilangan prima), cetaklah angka tersebut.

[2]:

```

# Meminta input dari pengguna
num = input("Enter a number greater than 1: ")

# Memeriksa apakah input adalah angka dan lebih dari 1
if not num.isnumeric():
    print("Error: Please enter a valid number.")
else:
    num = int(num)

    if num < 2:
        print("Error: Please enter a number greater than 1.")
    else:
        print(f"Prime numbers between 2 and {num} are:")
        for i in range(2, num + 1):
            # Asumsikan angka adalah prima

```

```

is_prime = True
for j in range(2, int(i ** 0.5) + 1): # Loop untuk mengecek faktor
    if i % j == 0: # Jika dapat dibagi, maka bukan bilangan prima
        is_prime = False
        break
if is_prime:
    print(i, end=' ')

```

Prime numbers between 2 and 7 are:

2 3 5 7

*Penjelasan Program:*

1. Program meminta input dari pengguna dan memeriksa apakah input adalah angka lebih besar dari 1 menggunakan isnumeric() dan validasi bilangan.
2. Jika angka kurang dari 2, program menampilkan pesan kesalahan.
3. Program menggunakan dua for loop:
  1. Loop pertama for i in range(2, num + 1) untuk memeriksa semua angka dari 2 hingga angka yang diinput pengguna.
  2. Loop kedua for j in range(2, int(i \*\* 0.5) + 1) digunakan untuk memeriksa apakah angka tersebut bisa dibagi oleh bilangan lain selain 1 dan dirinya sendiri.
  3. Jika i tidak bisa dibagi oleh bilangan lain, maka is\_prime tetap True dan angka tersebut adalah bilangan prima.
4. Setiap angka prima yang ditemukan akan dicetak.