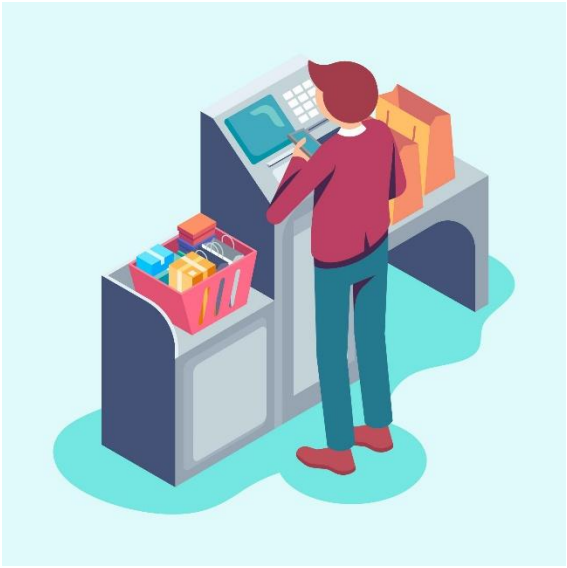
STRUKTUR DATA
LIST, STACK, DAN QUEUE

BAB I PENDAHULUAN

A. Pengantar

Dalam kegiatan sehari-hari di lingkungan sekolah, proses pengelolaan data merupakan hal yang tidak dapat dihindari. Berbagai aktivitas administratif, seperti pencatatan murid yang datang ke Tata Usaha (TU), pengaturan urutan pelayanan, hingga proses perbaikan kesalahan input data, dilakukan secara rutin. Kegiatan-kegiatan tersebut, meskipun terlihat sederhana, sebenarnya melibatkan proses pengolahan data yang membutuhkan ketelitian dan keteraturan.



Gambar 1.1. Ilustrasi Petugas TU

Apabila proses pengelolaan data tersebut dilakukan secara manual, berbagai potensi permasalahan dapat muncul. Contohnya, antrian pelayanan yang tidak tertib dapat menyebabkan ketidakadilan dalam pelayanan, kesalahan pencatatan data dapat mengakibatkan informasi yang tidak akurat, serta kesulitan dalam mengembalikan data ke kondisi sebelumnya dapat menghambat proses administrasi. Permasalahan-permasalahan ini

menunjukkan bahwa pengelolaan data yang tidak terstruktur dapat menurunkan efisiensi dan efektivitas suatu sistem.

Seiring dengan perkembangan teknologi informasi, berbagai permasalahan tersebut dapat diselesaikan dengan memanfaatkan konsep dalam pemrograman, khususnya melalui penerapan struktur data. Struktur data merupakan metode untuk menyimpan, mengatur, dan mengelola data sehingga dapat digunakan secara lebih terstruktur, sistematis, dan efisien. Dengan menggunakan struktur data yang tepat, proses pengolahan data dapat dilakukan dengan lebih cepat, akurat, dan mudah untuk dikembangkan lebih lanjut.

Dalam pemrograman, struktur data tidak hanya berfungsi sebagai tempat penyimpanan data, tetapi juga menentukan bagaimana data tersebut diakses dan dimanipulasi. Oleh karena itu, pemahaman terhadap struktur data menjadi salah satu kompetensi dasar yang harus dimiliki oleh peserta didik, khususnya dalam bidang teknologi informasi dan pemrograman.

Pada materi ini, peserta didik akan mempelajari tiga struktur data dasar dalam bahasa pemrograman Python, yaitu list, stack, dan queue. List digunakan sebagai struktur data linear dasar yang fleksibel, sedangkan stack dan queue merupakan struktur data yang memiliki aturan khusus dalam pengolahan elemennya. Stack menggunakan prinsip Last In First Out (LIFO), sedangkan queue menggunakan prinsip First In First Out (FIFO).

Ketiga struktur data tersebut akan dipelajari tidak hanya dari segi konsep, tetapi juga melalui implementasi dalam bentuk program sederhana. Salah satu contoh penerapan yang akan dikembangkan adalah sistem layanan antrian di lingkungan sekolah. Melalui pendekatan ini, peserta didik diharapkan dapat memahami bagaimana konsep struktur data dapat digunakan untuk menyelesaikan permasalahan nyata dalam kehidupan sehari-hari.

Dengan demikian, pembelajaran struktur data tidak hanya bersifat teoritis, tetapi juga aplikatif. Peserta didik diharapkan mampu mengembangkan keterampilan berpikir logis, sistematis, serta mampu merancang solusi berbasis teknologi untuk menyelesaikan berbagai permasalahan yang berkaitan dengan pengelolaan data.

B. Tujuan Pembelajaran

Setelah mempelajari materi ini, peserta didik diharapkan mampu memahami konsep dasar struktur data serta mengimplementasikannya

dalam bahasa pemrograman Python. Secara lebih rinci, tujuan pembelajaran ini adalah sebagai berikut:

1. Murid mampu menjelaskan konsep dan karakteristik struktur data list, stack, dan queue.
2. Murid mampu menentukan penggunaan struktur data list, stack, dan queue yang sesuai pada permasalahan dalam kehidupan sehari-hari.
3. Murid mampu menerapkan struktur data list, stack, dan queue dalam bahasa Python untuk menyelesaikan permasalahan sederhana.

C. Manfaat Pembelajaran

Pembelajaran mengenai struktur data pada materi ini memberikan manfaat dalam membantu peserta didik memahami cara mengelola data sederhana yang sering dijumpai dalam aktivitas di lingkungan sekolah maupun praktik kejuruan. Melalui pengenalan konsep list, stack, dan queue, peserta didik dapat memahami bagaimana data disusun dalam bentuk daftar, bagaimana urutan pekerjaan dikelola, serta bagaimana sistem antrian berjalan secara teratur.

Peserta didik akan terbantu dalam memahami berbagai situasi yang melibatkan pengelolaan urutan, seperti menyusun daftar tugas, mencatat data secara berurutan, maupun memahami alur pelayanan yang menggunakan sistem antrian. Konsep list membantu dalam menyimpan dan mengelola kumpulan data, sedangkan konsep stack dan queue membantu dalam memahami urutan pengambilan dan pemrosesan data sesuai aturan tertentu.

Selain itu, pembelajaran ini melatih peserta didik untuk berpikir secara runtut dan sistematis dalam menyelesaikan suatu proses. Peserta didik akan terbiasa mengikuti langkah-langkah yang jelas, seperti menambahkan data, menghapus data, serta memahami urutan prioritas dalam suatu kegiatan. Kemampuan ini dapat membantu peserta didik dalam mengerjakan tugas praktik yang membutuhkan ketelitian dan keteraturan.

Melalui contoh program sederhana menggunakan Python, peserta didik juga memperoleh gambaran bagaimana suatu sistem dapat dibuat untuk mengelola data secara otomatis, meskipun dalam bentuk yang sederhana. Hal ini membantu peserta didik dalam memahami dasar cara kerja sistem digital yang sering digunakan dalam berbagai kegiatan.

D. Ruang Lingkup Materi

Materi dalam modul ini difokuskan pada pembahasan struktur data linear yang meliputi list, stack, dan queue. Pembahasan dimulai dari pengenalan konsep dasar struktur data, kemudian dilanjutkan dengan penjelasan masing-masing jenis struktur data secara lebih mendalam. Pada bagian list, materi akan membahas pengertian, karakteristik, serta berbagai operasi dasar yang dapat dilakukan. Selanjutnya, pada bagian stack akan dijelaskan konsep LIFO beserta implementasinya dalam Python. Pada bagian queue, akan dibahas konsep FIFO serta penggunaannya dalam simulasi antrian. Setiap materi dilengkapi dengan contoh kode program dan ilustrasi untuk mempermudah pemahaman peserta didik. Dengan pendekatan ini, diharapkan peserta didik dapat memahami konsep secara menyeluruh, baik secara teoritis maupun praktis.

BAB II KONSEP DASAR STRUKTUR DATA

A. Pengertian Struktur Data

Dalam pemrograman, data merupakan komponen utama yang selalu digunakan dalam setiap proses. Data dapat berupa angka, teks, simbol, maupun informasi lain yang memiliki makna tertentu. Namun, keberadaan data saja tidak cukup untuk menghasilkan suatu sistem yang baik. Data perlu diatur dan disusun dengan cara tertentu agar dapat digunakan secara efektif dan efisien. Cara penyusunan dan pengelolaan data inilah yang disebut sebagai struktur data.

Struktur data merupakan metode atau cara untuk mengorganisasikan data di dalam komputer sehingga data tersebut dapat disimpan, diakses, diproses, dan diperbarui dengan lebih mudah. Dengan adanya struktur data, data yang awalnya tidak teratur dapat disusun menjadi lebih sistematis, sehingga memudahkan dalam proses pengolahan.

Dalam suatu program, struktur data tidak hanya berfungsi sebagai tempat penyimpanan, tetapi juga menentukan bagaimana data tersebut akan digunakan. Misalnya, dalam beberapa kondisi, data perlu diakses secara berurutan, sedangkan dalam kondisi lain, data mungkin perlu diambil berdasarkan urutan tertentu. Perbedaan kebutuhan ini menunjukkan bahwa cara pengelolaan data sangat bergantung pada struktur data yang digunakan.

Apabila data tidak dikelola dengan struktur yang baik, berbagai permasalahan dapat muncul. Data dapat menjadi sulit ditemukan, proses pengolahan menjadi lambat, serta berpotensi terjadi kesalahan dalam penggunaan data. Oleh karena itu, penggunaan struktur data yang tepat sangat penting dalam pengembangan program.

Dalam kehidupan sehari-hari, konsep struktur data sebenarnya sering digunakan tanpa disadari. Sebagai contoh, dalam layanan Tata Usaha (TU), petugas harus menyimpan data murid, mengatur antrean pelayanan, serta melakukan perbaikan apabila terjadi kesalahan input. Tanpa pengelolaan yang baik, proses tersebut dapat menjadi tidak teratur, seperti antrean yang tidak tertib atau kesulitan dalam mencari data tertentu. Struktur data membantu menyelesaikan permasalahan tersebut dengan cara yang lebih terstruktur dan sistematis.

B. Fungsi Struktur Data

Struktur data memiliki berbagai fungsi yang sangat penting dalam mendukung proses pengolahan data di dalam suatu program. Secara umum, fungsi-fungsi tersebut berkaitan dengan bagaimana data disusun, diakses, serta dikelola secara efisien. Salah satu fungsi utama struktur data adalah membantu mengorganisasikan data agar tersusun secara rapi dan sistematis. Data yang tersusun dengan baik akan lebih mudah dipahami dan digunakan sesuai dengan kebutuhan. Selain itu, struktur data juga mempermudah proses akses terhadap data, sehingga data dapat diambil dengan lebih cepat, baik untuk dibaca, diperbarui, maupun dihapus.

Selain dua fungsi tersebut, terdapat beberapa fungsi lain dari struktur data yang perlu diperhatikan, yaitu sebagai berikut:

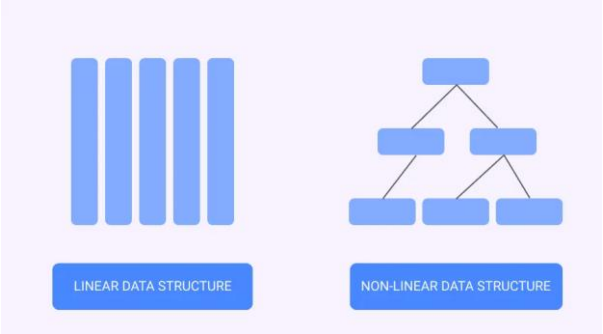
1. Meningkatkan efisiensi program
   Struktur data membantu program bekerja lebih optimal dengan penggunaan memori dan waktu proses yang lebih efisien. Hal ini penting terutama ketika program harus mengelola data dalam jumlah yang cukup banyak.
2. Mempermudah pengelolaan data
   Dengan struktur data yang tepat, proses menambahkan, menghapus, maupun memperbarui data dapat dilakukan dengan lebih teratur dan tidak membingungkan.
3. Mendukung pengembangan program

Program yang memiliki struktur data yang jelas akan lebih mudah dipahami oleh pengembang, sehingga memudahkan dalam proses perbaikan maupun pengembangan lanjutan.

4. Mengurangi kesalahan dalam pengolahan data
   Data yang terstruktur dengan baik akan mengurangi kemungkinan terjadinya kesalahan, baik dalam proses input, pengolahan, maupun output data.

C. Klasifikasi Struktur Data

Struktur data dapat diklasifikasikan berdasarkan cara penyusunan dan hubungan antar elemennya. Salah satu klasifikasi yang paling dasar adalah struktur data linear dan struktur data non-linear. Pemahaman terhadap klasifikasi ini membantu peserta didik dalam membedakan jenis struktur data berdasarkan karakteristiknya.



Gambar 2.1. Klasifikasi Struktur Data

Struktur data linear adalah struktur data yang elemen-elemennya disusun secara berurutan dalam satu garis. Setiap elemen memiliki posisi tertentu dan dapat diakses berdasarkan urutannya. Karena disusun secara berurutan, struktur data linear cenderung lebih mudah dipahami dan diimplementasikan, sehingga sering digunakan dalam pembelajaran dasar. Contoh struktur data linear yang akan dipelajari dalam materi ini adalah list, stack, dan queue. Meskipun memiliki bentuk penyusunan yang sama, ketiga struktur data ini memiliki aturan yang berbeda dalam pengelolaan data. Perbedaan ini menjadi hal penting yang perlu dipahami oleh peserta didik.

Di sisi lain, struktur data non-linear merupakan struktur data yang tidak disusun secara berurutan dalam satu garis. Elemen-elemen dalam struktur ini dapat memiliki hubungan yang lebih kompleks, seperti bercabang atau saling terhubung. Contoh struktur data non-linear antara lain

adalah tree dan graph. Struktur data non-linear biasanya digunakan untuk permasalahan yang lebih kompleks, sehingga pembahasannya dilakukan pada tingkat yang lebih lanjut. Dalam pembelajaran kali ini, pembahasan difokuskan pada struktur data linear sebagai dasar untuk memahami konsep pengelolaan data.

D. Struktur Data Linear

Struktur data linear merupakan jenis struktur data yang menyusun elemen-elemennya secara berurutan dari satu elemen ke elemen berikutnya. Setiap elemen memiliki hubungan langsung dengan elemen sebelumnya dan elemen sesudahnya. Penyusunan yang berurutan ini memudahkan dalam proses penelusuran data. Struktur data linear banyak digunakan dalam berbagai aktivitas yang melibatkan urutan. Dalam kehidupan sehari-hari, banyak proses yang dilakukan secara berurutan, seperti menyusun daftar pekerjaan, mencatat data, atau mengelola antrian. Hal ini menunjukkan bahwa struktur data linear memiliki keterkaitan yang erat dengan aktivitas sehari-hari.

Dalam pembelajaran ini, terdapat tiga jenis struktur data linear yang akan dipelajari, yaitu list, stack, dan queue. List digunakan untuk menyimpan sekumpulan data dalam bentuk daftar. Stack digunakan untuk mengelola data dengan prinsip tumpukan, sedangkan queue digunakan untuk mengelola data dalam bentuk antrian. Meskipun ketiganya termasuk dalam struktur data linear, masing-masing memiliki cara kerja yang berbeda. Perbedaan ini terletak pada aturan dalam menambahkan dan mengambil data. Oleh karena itu, pemahaman terhadap masing-masing struktur data menjadi penting agar dapat digunakan secara tepat sesuai kebutuhan**.**

Gambar 2.2. Ilustrasi Queue

Gambar 2.3. Ilustrasi Stack

E. Karakteristik Struktur Data Linear

Struktur data linear memiliki beberapa karakteristik yang menjadi ciri khasnya. Karakteristik ini membantu dalam memahami bagaimana data disusun dan dikelola dalam suatu sistem. Secara umum, struktur data linear memiliki pola yang sederhana karena elemen-elemennya disusun secara berurutan. Salah satu karakteristik utama adalah data disusun secara berurutan, sehingga setiap elemen memiliki posisi tertentu dalam urutan tersebut. Posisi ini dapat digunakan untuk mengidentifikasi dan mengakses data dengan lebih mudah. Selain itu, setiap elemen juga memiliki hubungan yang jelas dengan elemen lainnya, yaitu elemen sebelum dan sesudahnya, sehingga memudahkan dalam proses penelusuran data. Untuk memahami lebih jelas, berikut beberapa karakteristik utama dari struktur data linear:

1. Data disusun secara berurutan
   Setiap elemen berada dalam suatu urutan tertentu, sehingga data dapat diakses berdasarkan posisi atau indeksnya.
2. Memiliki hubungan antar elemen
   Setiap elemen terhubung dengan elemen sebelumnya dan elemen sesudahnya, sehingga memudahkan proses penelusuran dari awal hingga akhir.
3. Memiliki aturan dalam pengelolaan data
   Setiap jenis struktur data linear memiliki aturan tersendiri dalam mengelola data. Misalnya, pada stack digunakan prinsip Last In First Out (LIFO), sedangkan pada queue digunakan prinsip First In First Out (FIFO).

BAB III STRUKTUR DATA LIST

A. Pengertian List

List merupakan salah satu struktur data yang paling dasar dan sering digunakan dalam bahasa pemrograman Python. List digunakan untuk menyimpan sekumpulan data dalam satu variabel, sehingga memudahkan dalam pengelolaan data yang memiliki jumlah lebih dari satu.

Dalam list, data disusun secara berurutan dan setiap elemen memiliki posisi tertentu yang disebut dengan indeks. Indeks ini digunakan untuk mengakses data yang terdapat di dalam list. Dengan adanya indeks,

pengguna dapat mengambil atau mengubah data secara langsung tanpa harus menelusuri seluruh isi list.

List bersifat fleksibel karena dapat menyimpan berbagai jenis data, seperti angka, teks, maupun kombinasi dari keduanya. Selain itu, jumlah elemen dalam list dapat berubah sesuai kebutuhan, sehingga sangat cocok digunakan dalam berbagai kondisi. Sebagai contoh, list dapat digunakan untuk menyimpan daftar nama siswa, daftar nilai, maupun daftar pekerjaan yang harus dilakukan.

B. Cara Kerja List

List bekerja dengan cara menyimpan data secara berurutan di dalam suatu wadah. Data yang terdapat di dalam list disebut sebagai elemen. Setiap elemen merupakan isi dari list yang dapat berupa teks (string), angka (integer), maupun jenis data lainnya. Agar setiap elemen dapat dikenali dan diakses dengan mudah, setiap elemen memiliki posisi tertentu yang disebut dengan indeks. Indeks merupakan nomor yang digunakan untuk menunjukkan letak suatu elemen di dalam list.

Dalam Python, penomoran indeks dimulai dari angka 0, bukan dari 1. Hal ini berarti elemen pertama dalam list berada pada indeks ke-0, elemen kedua berada pada indeks ke-1, dan seterusnya. Dengan adanya indeks, pengguna dapat mengambil atau mengubah data secara langsung berdasarkan posisinya tanpa harus membaca seluruh isi list. Adapun contoh list ditunjukkan pada Gambar 3.1.

Gambar 3.1. Contoh List

maka posisi masing-masing elemen berdasarkan Gambar 3.1 adalah:

1. "Andi" berada pada indeks 0
2. "Budi" berada pada indeks 1
3. "Citra" berada pada indeks 2

C. Membuat List dengan Python

Dalam bahasa pemrograman Python, list dapat dibuat dengan menggunakan tanda kurung siku [ ]. Di dalam tanda tersebut, data atau elemen dituliskan dan dipisahkan menggunakan tanda koma. Cara ini

digunakan untuk menyimpan beberapa data dalam satu variabel secara berurutan. Sebagai contoh, berikut merupakan cara membuat list yang berisi beberapa nama siswa yang ditunjukkan pada Gambar 3.2.

Gambar 3.2. Cara Membuat List

Pada contoh tersebut, variabel nama_siswa menyimpan tiga elemen, yaitu "Andi", "Budi", dan "Citra". Ketiga data tersebut disusun dalam satu list sehingga dapat dikelola secara bersamaan.

List juga dapat dibuat dengan cara lain, yaitu menggunakan tanda kurung siku [] dalam keadaan kosong. List kosong biasanya digunakan ketika data belum tersedia atau akan ditambahkan secara bertahap selama program berjalan. Contoh pembuatan list kosong ditunjukkan pada Gambar 3.3.

Gambar 3.3. Cara Membuat List Kosong

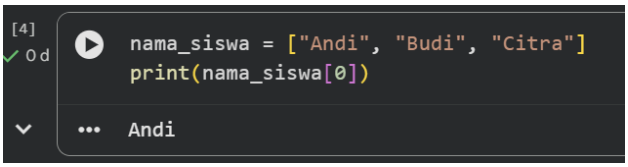
List kosong tersebut dapat diisi dengan data sesuai kebutuhan menggunakan operasi penambahan data yang akan dipelajari pada bagian berikutnya.

Dari penjelasan ini dapat dipahami bahwa list dalam Python dapat dibuat dengan cara yang sederhana, baik dengan langsung mengisi data maupun dengan membuat list kosong terlebih dahulu. Cara ini memberikan keleluasaan bagi pengguna untuk menyesuaikan bagaimana data akan dikelola sejak awal program dibuat. Pemahaman ini menjadi langkah awal sebelum data dapat diolah lebih lanjut, karena hampir setiap proses dalam list selalu diawali dari bagaimana list tersebut dibuat.

D. Mengakses Data dalam List

Data yang terdapat di dalam list dapat diakses dengan menggunakan indeks. Indeks merupakan nomor posisi dari setiap elemen dalam list yang digunakan untuk menunjukkan letak data tersebut. Dengan adanya indeks, pengguna dapat mengambil data tertentu secara langsung tanpa harus

membaca seluruh isi list. Untuk mengakses data dalam list, indeks dituliskan di dalam tanda kurung siku [] setelah nama variabel. Cara ini memungkinkan pengguna untuk memilih elemen berdasarkan posisinya. Sebagai contoh, perhatikan program pada Gambar 3.4.



Gambar 3.4. Program untuk Mengakses List

Pada contoh tersebut, nama_siswa[0] digunakan untuk mengambil elemen pada indeks ke-0. Karena indeks dimulai dari 0, maka data yang diambil adalah elemen pertama, yaitu "Andi".
Jika ingin mengakses elemen lainnya, maka indeks dapat disesuaikan. Sebagai contoh:

1. nama_siswa[1] akan menghasilkan "Budi"
2. nama_siswa[2] akan menghasilkan "Citra"

Dalam Python, indeks tidak hanya menggunakan angka positif, tetapi juga dapat menggunakan indeks negatif. Indeks negatif digunakan untuk mengakses elemen dari bagian akhir list, bukan dari awal. Jika indeks positif dimulai dari angka 0 (elemen pertama), maka indeks negatif dimulai dari -1 untuk elemen terakhir. Maka posisi indeksnya dapat dilihat seperti ini:

1. "Andi" → indeks 0 atau -3
2. "Budi" → indeks 1 atau -2
3. "Citra" → indeks 2 atau -1

Dengan demikian, setiap elemen dalam list dapat diakses secara langsung berdasarkan posisinya.

Beberapa hal penting yang perlu diperhatikan dalam mengakses data pada list adalah sebagai berikut:

1. Indeks dimulai dari angka 0
   Elemen pertama selalu berada pada indeks ke-0, bukan ke-1. Hal ini sering menjadi kesalahan bagi pemula yang baru belajar pemrograman.
2. Akses menggunakan tanda kurung siku []
   Indeks harus dituliskan di dalam tanda kurung siku setelah nama variabel, misalnya nama_siswa[0].

3. Indeks harus sesuai dengan jumlah elemen
   Jika indeks yang digunakan melebihi jumlah elemen dalam list, maka program akan menghasilkan error.

Perhatikan contoh-contoh kesalahan cara mengakses list berikut.

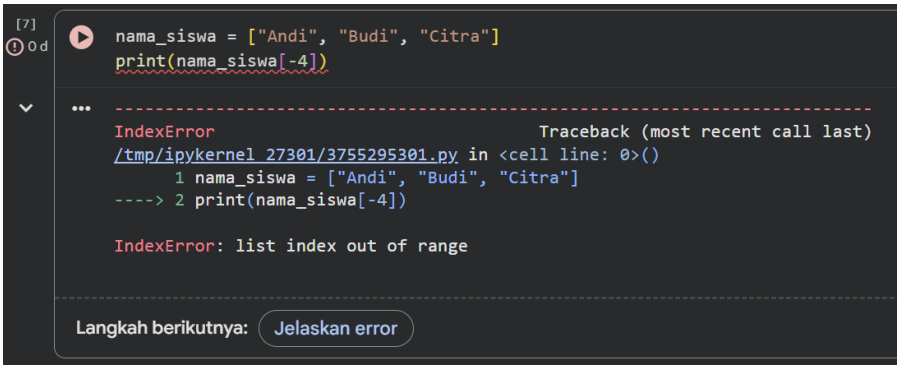
1. Index tidak sesuai dengan yang tersedia dalam list
   Kesalahan yang sering terjadi dalam penggunaan list adalah ketika indeks yang digunakan tidak sesuai dengan jumlah elemen yang tersedia di dalam list. Hal ini berarti pengguna mencoba mengakses posisi data yang sebenarnya tidak ada dalam list tersebut. Perhatikan contoh pada gambar berikut.

Gambar 3.5. Contoh Pertama Kesalahan Mengakses List

Pada Gambar 3.5. indeks yang tersedia adalah sebagai berikut:

a. "Andi" → indeks 0 atau -3
b. "Budi" → indeks 1 atau -2
c. "Citra" → indeks 2 atau -1

Maka indeks ke-5 tidak tersedia dan akan menyebabkan error.



Gambar 3.6. Contoh Kedua Kesalahan Mengakses List

Pada Gambar 3.6. indeks yang tersedia adalah sebagai berikut:

a. "Andi" → indeks 0 atau -3
b. "Budi" → indeks 1 atau -2
c. "Citra" → indeks 2 atau -1

Maka indeks ke- -4 tidak tersedia dan akan menyebabkan error.

2. Salah penulisan tanda kurung

Kesalahan lain yang sering terjadi dalam mengakses data pada list adalah penggunaan tanda kurung yang tidak sesuai. Dalam Python, elemen pada list harus diakses menggunakan tanda kurung siku [], bukan tanda kurung biasa ( ). Hal ini penting karena setiap jenis tanda kurung dalam Python memiliki fungsi yang berbeda. Perhatikan contoh pada gambar berikut.

Gambar 3.7. Contoh Ketiga Kesalahan Mengakses List

Pada Gambar 3.7. tanda kurung yang digunakan adalah tanda kurung biasa ( ). Sedangkan python meminta menggunakan tanda kurung siku [ ]. Kesalahan ini akan menyebabkan program error dan list gagal diakses.

3. Variabel list belum dibuat

Kesalahan lain yang cukup sering terjadi adalah ketika program mencoba mengakses sebuah list yang sebenarnya belum dibuat atau belum dideklarasikan sebelumnya. Dalam Python, setiap variabel harus dibuat terlebih dahulu sebelum digunakan. Jika sebuah list belum didefinisikan, maka Python tidak akan mengenali nama tersebut. Kesalahan ini biasanya terjadi karena pengguna lupa

membuat list terlebih dahulu, atau terjadi kesalahan penamaan variabel saat proses pengaksesan data. Perhatikan contoh pada gambar berikut.

Gambar 3.8. Contoh Keempat Kesalahan Mengakses List

Pada Gambar 3.8. python akan mencoba mengambil elemen pertama dari list bernama data_siswa. Masalahnya saat python mencoba mencari list bernama nama_siswa python tidak menemukannya. List yang ada pada gambar tersebut adalah nama_siswa.

4. Salah menulis nama variabel list

   Kesalahan ini sebenarnya sama dengan kesalahan variabel list yang belum dibuat. Namun pada kali ini kesalahan lebih disebabkan pada kekeliruan satu dua huruf seperti typo. Perhatikan contoh pada gambar berikut.

Gambar 3.9. Contoh Kelima Kesalahan Mengakses List

Pada Gambar 3.9. variabel list yang ada yaitu nama_siswa, sedangkan saat diakses menggunakan nama_siswA. nama_siswa dan nama_siswA adalah dua hal yang berbeda. Oleh karena itu pastikan saat mengakses list nama varibelnya benar-benar sama.

E. Operasi Dasar pada List

Dalam penggunaan list di Python, terdapat beberapa operasi dasar yang digunakan untuk mengelola data. Operasi ini sangat penting karena memungkinkan pengguna untuk menambahkan, menghapus, mengubah, dan menampilkan data yang tersimpan di dalam list.

1. Menambahkan Data

   Menambahkan data ke dalam list merupakan salah satu operasi dasar yang digunakan untuk memasukkan elemen baru ke dalam list yang sudah dibuat sebelumnya. Dalam Python, terdapat beberapa cara untuk menambahkan data ke dalam list, tergantung pada posisi data yang ingin dimasukkan. Secara umum, penambahan data dapat dilakukan di bagian akhir list maupun pada posisi tertentu di dalam list.

   a. Menambahkan Data di Akhir List (append())

      Metode append() digunakan untuk menambahkan data ke bagian paling akhir dari list. Data yang ditambahkan akan otomatis menjadi elemen terakhir. Perhatikan contoh pada gambar berikut.

Gambar 3.10. Cara Menambah Data pada List dengan Append

Berikut merupakan penjelasan dari kode pada Gambar 3.10.

1) append() adalah method (fungsi yang dimiliki oleh list)

2) append() selalu menambahkan data di posisi terakhir
3) Setelah "Citra ditambahkan, posisi data menjadi urut di akhir list

b. Menambahkan Data pada Posisi Tertentu (insert())
Metode insert() digunakan untuk menambahkan data pada posisi tertentu dalam list berdasarkan indeks. Perhatikan contoh pada gambar berikut.

Gambar 3.11. Cara Menambah Data pada List dengan Insert

Berikut merupakan penjelasan dari kode pada Gambar 3.11.

1) insert(1, "Dewi") berarti: angka 1 adalah indeks posisi, dan "Dewi" adalah data yang ingin dimasukkan
2) Data baru akan ditempatkan di indeks ke-1
3) Data yang sudah ada sebelumnya pada indeks ke-1 dan seterusnya akan bergeser ke kanan

c. Menambahkan Lebih dari Satu Data (extend())
Metode extend() digunakan ketika kita ingin menambahkan beberapa data sekaligus ke dalam sebuah list. Jika sebelumnya kita harus menambahkan data satu per satu, maka dengan extend() kita bisa langsung memasukkan banyak data dalam satu langkah. Semua data tersebut akan ditambahkan ke bagian akhir list, dan masing-masing elemen akan diperlakukan sebagai data yang terpisah, bukan sebagai satu kesatuan. Perhatikan contoh pada gambar berikut.

```
[13]
✓ 0 d
nama_siswa = ["Andi", "Budi"]

nama_siswa.extend(["Citra", "Doni"])
print(nama_siswa)

['Andi', 'Budi', 'Citra', 'Doni']
```

Gambar 3.12. Cara Menambah Data pada List dengan Extend

Berikut merupakan penjelasan dari kode pada Gambar 3.12.

1) extend() menerima beberapa data sekaligus dalam bentuk list lain. Pada Gambar 10 data yang ditambahkan adalah "Citra" dan "Doni"
2) Semua data akan ditambahkan ke akhir list, yakni setelah "Budi"

2. Menghapus Data

Menghapus data pada list adalah proses untuk menghilangkan elemen yang sudah tidak dibutuhkan lagi dari dalam list. Dalam Python, terdapat beberapa cara untuk menghapus data, tergantung dari bagaimana kita ingin menghapusnya, apakah berdasarkan nilai, berdasarkan posisi (indeks), atau menghapus seluruh isi list sekaligus.

a. Menghapus Data Berdasarkan Nilai (remove())

Metode remove() digunakan untuk menghapus elemen berdasarkan nilai yang ada di dalam list. Artinya, kita tidak perlu mengetahui indeksnya, cukup menyebutkan data yang ingin dihapus. Perhatikan contoh pada gambar berikut.

```
[14]
✓ 0 d
nama_siswa = ["Andi", "Budi", "Citra"]

nama_siswa.remove("Budi")
print(nama_siswa)

['Andi', 'Citra']
```

Gambar 3.13. Menghapus Data Berdasarkan Nilai (remove())

Berikut merupakan penjelasan dari kode pada Gambar 3.13.

1) Python akan mencari nilai "Budi" di dalam list
2) Setelah ditemukan, data tersebut langsung dihapus
3) Jika terdapat lebih dari satu nilai yang sama, maka hanya kemunculan pertama yang akan dihapus.

b. Menghapus Data Berdasarkan Indeks (pop())

Metode pop() digunakan untuk menghapus data berdasarkan posisi indeks dalam list. Kita bisa menentukan indeks mana yang ingin dihapus. Perhatikan contoh pada gambar berikut.

Gambar 3.14. Menghapus Data Berdasarkan Indeks (pop())

Berikut merupakan penjelasan dari kode pada Gambar 3.14.

1) Angka 1 menunjukkan indeks ke-1 (yaitu "Budi")
2) Data pada indeks tersebut akan dihapus
3) Setelah dihapus, elemen di belakangnya akan bergeser ke depan

Catatan penting:

1) Jika pop() digunakan tanpa angka indeks, maka secara otomatis akan menghapus elemen terakhir dalam list.

c. Menghapus Data dengan Indeks (del)

Selain pop(), Python juga menyediakan cara lain untuk menghapus elemen berdasarkan indeks, yaitu menggunakan perintah del. Perintah ini digunakan untuk menghapus data pada posisi tertentu di dalam list dengan menyebutkan indeksnya secara langsung. Berbeda dengan pop(), del tidak hanya terbatas pada penghapusan satu elemen saja, tetapi juga bisa digunakan untuk menghapus beberapa elemen sekaligus menggunakan rentang indeks (slicing). Perhatikan contoh pada gambar berikut.

```
[16]
✓ 0d
nama_siswa = ["Andi", "Budi", "Citra", "Doni", "Eka"]

del nama_siswa[1:4]
print(nama_siswa)

['Andi', 'Eka']
```

Gambar 3.15. Menghapus Data dengan Indeks (del)

Berikut merupakan penjelasan dari kode pada Gambar 3.15.

1) nama_siswa[1:4] disebut slicing. Artinya: mulai dari indeks 1 (termasuk) sampai indeks 4 (tidak termasuk)
2) Jadi elemen yang dihapus adalah: indeks 1 → "Budi", indeks 2 → "Citra", dan indeks 3 → "Doni"

d. Menghapus Semua Data dalam List (clear())

Metode clear() digunakan untuk menghapus semua elemen yang ada di dalam sebuah list sekaligus, sehingga setelah perintah ini dijalankan, list tidak lagi berisi data apa pun dan berubah menjadi list kosong. Berbeda dengan metode penghapusan lainnya yang hanya menghapus sebagian data, clear() bekerja dengan cara mengosongkan seluruh isi list tanpa memandang jumlah maupun jenis elemen di dalamnya. Namun, yang perlu diperhatikan adalah list itu sendiri tetap ada dalam memori, hanya saja isinya saja yang dihapus. Metode ini biasanya digunakan ketika kita ingin menggunakan kembali list yang sama, tetapi dengan kondisi awal yang kosong, misalnya untuk mengisi ulang data baru dari awal tanpa membuat variabel list yang baru. Perhatikan contoh pada gambar berikut.

```
[17]
✓ 0d
nama_siswa = ["Andi", "Budi", "Citra"]

nama_siswa.clear()
print(nama_siswa)

[]
```

Gambar 3.16. Menghapus Semua Data dalam List (clear())

Berikut merupakan penjelasan dari kode pada Gambar 3.16.

1) Semua elemen dihapus sekaligus
2) List masih ada, tetapi tidak berisi data apa pun (kosong)

3. Mengubah Data
    a. Mengubah Satu Elemen Berdasarkan Indeks

    Cara paling dasar untuk mengubah data pada list dilakukan dengan menentukan posisi elemen yang ingin diubah menggunakan indeks, kemudian menggantinya dengan nilai yang baru. Dengan mengetahui letak data melalui indeksnya, pengguna dapat langsung melakukan perubahan tanpa harus mengubah seluruh isi list. Perhatikan contoh pada gambar berikut.

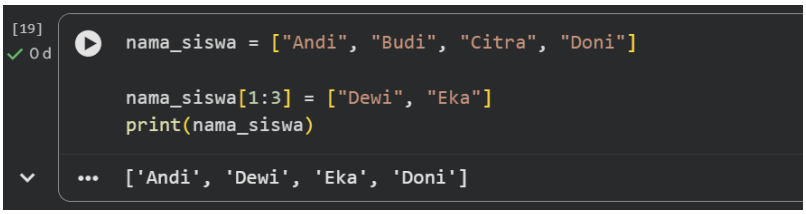
Gambar 3.17. Mengubah Satu Elemen Berdasarkan Indeks

Berikut merupakan penjelasan dari kode pada Gambar 3.17.

1) nama_siswa[1] menunjuk pada elemen "Budi"
2) Nilai tersebut diganti menjadi "Dewi"
3) Elemen lain tidak berubah

    b. Mengubah Beberapa Elemen Sekaligus (Slicing)

    Selain mengubah satu elemen saja, Python juga memungkinkan kita untuk mengubah beberapa elemen sekaligus dalam satu langkah. Hal ini dapat dilakukan menggunakan teknik yang disebut slicing, yaitu cara untuk mengakses atau memilih sebagian data dalam list berdasarkan rentang indeks tertentu. Dengan slicing, kita tidak perlu mengubah data satu per satu. Cukup dengan menentukan bagian mana dari list yang ingin diubah, kemudian menggantinya dengan data baru. Perhatikan contoh pada gambar berikut.



Gambar 3.18. Mengubah Beberapa Elemen Sekaligus (Slicing)

Berikut merupakan penjelasan dari kode pada Gambar 3.18.

1) [1:3] berarti mengambil data mulai dari indeks ke-1 sampai sebelum indeks ke-3, dengan kata lain, yang terpilih adalah indeks ke-1 dan ke-2
2) Pada contoh tersebut, data yang terkena perubahan adalah "Budi" dan "Citra"

Pada Gambar 3.18. jumlah data pengganti jumlahnya sama yakni dua. Selain itu, dengan teknik slicing dapat dilakukan jumlah data pengganti tidak harus sama dengan jumlah data yang diganti. Artinya, dua data bisa diganti menjadi satu data, atau bahkan lebih banyak data, tergantung kebutuhan. Perhatikan contoh pada gambar berikut.

Gambar 3.19. Mengganti Dua Data Menjadi Satu Data

Berikut merupakan penjelasan dari kode pada Gambar 3.19.

1) 1:3 mengambil "Budi" dan "Citra"
2) Kedua data tersebut diganti menjadi satu data, yaitu "Dewi"
3) Akibatnya, jumlah elemen dalam list berkurang

```
[21]
✓ 0d
nama_siswa = ["Andi", "Budi", "Citra", "Doni"]

nama_siswa[1:3] = ["Dewi", "Eka", "Fani"]
print(nama_siswa)

['Andi', 'Dewi', 'Eka', 'Fani', 'Doni']
```

Gambar 3.20. Mengganti Dua Data Menjadi Lebih Banyak Data

Berikut merupakan penjelasan dari kode pada Gambar 3.20.

1) "Budi" dan "Citra" diganti dengan tiga data baru
2) Jumlah elemen dalam list menjadi lebih banyak dari sebelumnya

c. Mengubah Data dengan Nilai yang Dihitung

Data dalam list tidak hanya bisa diubah dengan mengganti langsung nilainya, tetapi juga dapat diperbarui berdasarkan hasil perhitungan atau proses tertentu. Artinya, nilai baru yang dimasukkan ke dalam list bukan berasal dari data tetap, melainkan hasil dari operasi seperti penjumlahan, pengurangan, atau proses lainnya. Cara ini biasanya digunakan ketika data perlu disesuaikan atau diperbarui secara dinamis. Misalnya, dalam kasus nilai siswa yang mendapatkan tambahan poin. Perhatikan contoh pada gambar berikut.

```
[23]
✓ 0d
nilai = [70, 80, 90]

nilai[0] = nilai[0] + 10
print(nilai)

[80, 80, 90]
```

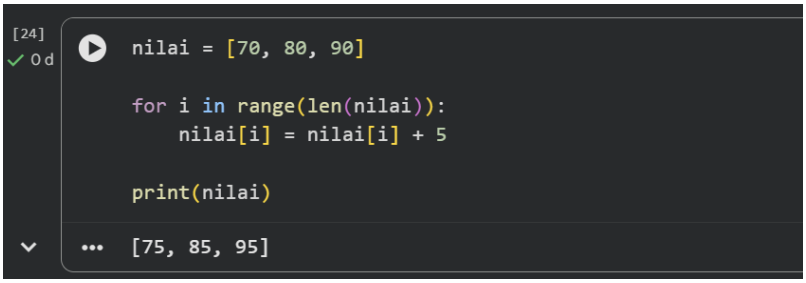
Gambar 3.21. Mengubah Data dengan Nilai yang Dihitung

Berikut merupakan penjelasan dari kode pada Gambar 3.21.

1) nilai[0] awalnya adalah 70
2) Kemudian dilakukan perhitungan 70 + 10
3) Hasilnya (80) disimpan kembali ke posisi yang sama

4) Data lama diganti dengan hasil perhitungan tersebut

Selain cara seperti itu kita bisa memanfaatkan perulangan untuk mengubah setiap elemen dalam list secara bertahap. Perulangan (looping) merupakan proses menjalankan suatu perintah secara berulang-ulang selama kondisi tertentu terpenuhi. Dalam Python, perulangan sering digunakan untuk mengakses setiap elemen dalam list satu per satu, sehingga data dapat diproses tanpa harus ditulis secara berulang secara manual. Perhatikan contoh pada gambar berikut.



Gambar 3.22. Mengubah Banyak Data dengan Nilai yang Dihitung Melalui Perulangan

Berikut merupakan penjelasan dari kode pada Gambar 3.22.

1) for adalah perintah untuk melakukan perulangan
2) range(len(nilai)) digunakan untuk menghasilkan indeks dari list
3) Perulangan akan berjalan dari indeks ke-0 sampai indeks terakhir
4) Pada setiap perulangan:
    a) data diambil dari list
    b) dilakukan perhitungan (ditambah 5)
    c) hasilnya disimpan kembali ke posisi yang sama

4. Menelusuri Data

Menelusuri data pada list merupakan proses untuk membaca seluruh elemen yang terdapat di dalam list secara berurutan. Proses ini

biasanya dilakukan ketika ingin melihat isi data atau mengolah setiap elemen yang ada.

a. Menelusuri Data Secara Langsung (Tanpa Indeks)
   Pada cara ini, perulangan langsung mengambil nilai elemen dari dalam list tanpa perlu mengetahui posisinya. Perhatikan contoh pada gambar berikut.

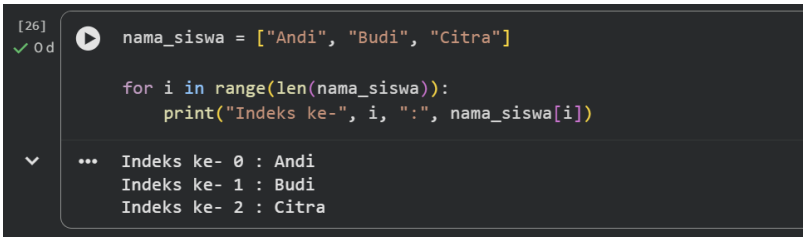
Gambar 3.23. Menelusuri Data Secara Langsung (Tanpa Indeks)

Berikut merupakan penjelasan dari kode pada Gambar 3.23.

1) for nama in nama_siswa berarti: ambil setiap data dalam list satu per satu
2) nama akan berisi: "Andi", lalu "Budi", lalu "Citra"

Cara ini fokus pada isi data, bukan posisinya. Cara ini cocok digunakan ketika hanya ingin menampilkan atau mengolah data dan tidak membutuhkan informasi indeks.

b. Menelusuri Data Menggunakan Indeks
   Pada cara ini, perulangan tidak langsung mengambil data, tetapi mengambil nomor indeksnya terlebih dahulu, kemudian digunakan untuk mengakses data dalam list. Perhatikan contoh pada gambar berikut.



Gambar 3.24. Menelusuri Data Menggunakan Indeks

Berikut merupakan penjelasan dari kode pada Gambar 3.24.

1) range(len(nama_siswa)) menghasilkan angka indeks (0, 1, 2)
2) i menyimpan indeks tersebut
3) nama_siswa[i] digunakan untuk mengambil data berdasarkan indeks

BAB IV STRUKTUR DATA QUEUE

A. Pengertian Queue

Dalam kehidupan sehari-hari, kita sering menjumpai situasi di mana seseorang harus menunggu giliran untuk mendapatkan layanan. Misalnya saat mengantre di kantin, di loket administrasi, atau ketika menunggu pelayanan di Tata Usaha sekolah. Pada kondisi tersebut, terdapat aturan tidak tertulis bahwa siapa yang datang lebih dahulu akan dilayani terlebih dahulu. Konsep inilah yang menjadi dasar dari struktur data queue dalam pemrograman.

Queue adalah struktur data yang digunakan untuk menyimpan sekumpulan data dengan aturan pengolahan tertentu, yaitu mengikuti prinsip FIFO (First In First Out). Prinsip ini berarti bahwa data yang pertama kali masuk ke dalam struktur akan menjadi data pertama yang diproses atau dikeluarkan. Dengan kata lain, queue bekerja seperti sebuah barisan antrean. Setiap data yang masuk akan ditempatkan di bagian belakang antrean, sedangkan data yang berada di bagian depan akan diproses lebih dahulu. Selama proses berlangsung, urutan data tetap terjaga sesuai dengan waktu kedatangannya.

Struktur data ini sangat penting dalam berbagai sistem yang membutuhkan keteraturan dan keadilan dalam pemrosesan data, seperti sistem antrian layanan, pemrosesan tugas pada komputer, hingga

pengelolaan data dalam jaringan. Berbeda dengan struktur data list yang memberikan kebebasan dalam mengakses dan mengolah data, queue memiliki aturan yang lebih terstruktur. Data tidak dapat diambil secara sembarangan, melainkan harus mengikuti urutan yang telah ditentukan. Hal ini membuat queue sangat cocok digunakan pada permasalahan yang berkaitan dengan sistem antrean.

B. Cara Kerja Queue

Bayangkan sebuah situasi antrean di bagian Tata Usaha (TU) sekolah. Beberapa siswa datang untuk mengurus administrasi, lalu berdiri membentuk barisan. Siswa yang datang lebih awal berada di bagian depan, sedangkan siswa yang datang belakangan berada di bagian belakang.

Dalam kondisi tersebut, petugas TU akan melayani siswa satu per satu, dimulai dari siswa yang berada di paling depan. Setelah siswa tersebut selesai dilayani, ia akan keluar dari antrean, dan posisi terdepan akan ditempati oleh siswa berikutnya. Proses ini terus berlangsung selama masih ada siswa dalam antrean. Sementara itu, siswa baru yang datang akan langsung menempati posisi paling belakang tanpa mengganggu urutan yang sudah ada. Konsep inilah yang menggambarkan cara kerja queue dalam pemrograman.

Dalam struktur data queue, terdapat dua bagian penting, yaitu front (depan) dan rear (belakang). Front (depan) adalah bagian tempat data pertama berada dan akan diproses terlebih dahulu. Sedangkan rear (belakang) adalah bagian tempat data baru ditambahkan. Perhatikan Gambar 4.1. untuk memahami bagian penting dari queue.

Gambar 4.1. Cara Kerja Queue

Ketika sebuah data dimasukkan ke dalam queue, data tersebut akan ditempatkan di bagian belakang (rear). Proses ini disebut sebagai enqueue. Sebaliknya, ketika sebuah data dikeluarkan dari queue, data yang diambil

adalah data yang berada di bagian depan (front). Proses ini disebut sebagai dequeue. Dengan mekanisme ini, queue memastikan bahwa setiap data diproses secara adil sesuai dengan urutan kedatangannya. Tidak ada data yang bisa "menyalip" atau diproses lebih dahulu jika datang belakangan. Sebagai ilustrasi sederhana, misalkan terdapat tiga siswa dalam antrean:

1. Andi (datang pertama)
2. Budi
3. Citra (datang terakhir)

Urutan antrean:
[Andi, Budi, Citra], Jika satu siswa dilayani, maka:

1. Andi keluar dari antrean
2. Budi menjadi yang terdepan

Sehingga antrean menjadi:
[Budi, Citra], Jika ada siswa baru, misalnya Dina, maka ia akan masuk ke bagian belakang:
[Budi, Citra, Dina]

C. Membuat Queue dengan Python

Setelah memahami konsep dan cara kerja queue, langkah selanjutnya adalah membuat struktur queue dalam bahasa pemrograman Python. Dalam Python terdapat beberapa cara yang dapat digunakan untuk merepresentasikan queue, seperti menggunakan list, struktur khusus deque, maupun modul queue. Dengan memahami berbagai cara tersebut, pengguna dapat memilih metode yang paling sesuai, baik untuk pembelajaran dasar maupun untuk pengembangan program yang lebih lanjut.

1. Membuat Queue dengan List

   Pada dasarnya, list digunakan sebagai wadah untuk menyimpan data yang tersusun secara berurutan. Urutan ini kemudian dimanfaatkan untuk menggambarkan konsep antrean, di mana elemen pertama berada di posisi awal dan elemen berikutnya berada di posisi setelahnya. Sebagai contoh perhatikan gambar berikut.

Gambar 4.2. Membuat Queue dengan List

Berikut merupakan penjelasan kode dari Gambar 4.2.

a. "Andi" berada pada posisi paling awal (indeks 0)
b. "Budi" berada di tengah
c. "Citra" berada pada posisi terakhir

Urutan ini mencerminkan kondisi antrean, di mana setiap data tersusun berdasarkan urutan tertentu.

a. Makna Posisi dalam List

   Dalam konteks queue yang dibuat menggunakan list, setiap posisi memiliki arti:

   1) Elemen pertama (indeks 0), menunjukkan bagian depan antrean (front), yaitu data yang berada di urutan awal.
   2) Elemen terakhir, menunjukkan bagian belakang antrean (rear), yaitu tempat data baru ditempatkan.

b. Sifat Dinamis List

   Salah satu alasan list dapat digunakan untuk merepresentasikan queue adalah karena sifatnya yang dinamis, yaitu:

   1) Dapat bertambah jumlah elemennya
   2) Dapat berkurang sesuai kebutuhan
   3) Dapat menyimpan berbagai jenis data

c. Kelebihan Membuat Queue dengan List

   Penggunaan list untuk membuat queue memiliki beberapa kelebihan, antara lain:

   1) Mudah dipahami oleh pemula
   2) Tidak memerlukan impor modul tambahan
   3) Fleksibel untuk berbagai jenis data

d. Kekurangan Membuat Queue dengan List

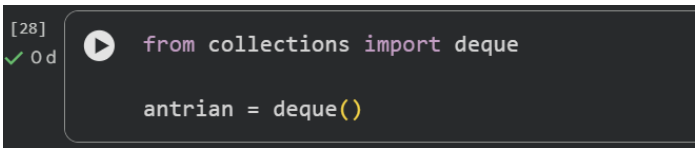
   Meskipun sederhana, penggunaan list sebagai queue memiliki keterbatasan, yaitu:

   1) Kurang efisien jika digunakan untuk data dalam jumlah besar

2. Membuat Queue dengan Dequeue

   Selain menggunakan list, queue dalam Python juga dapat dibuat menggunakan deque yang tersedia dalam modul collections. Deque dirancang untuk menyimpan data secara berurutan dan mendukung pengelolaan data pada kedua ujungnya. Untuk membuat queue

menggunakan deque, langkah pertama adalah mengimpor modul yang diperlukan, kemudian mendefinisikan variabel sebagai deque. Perhatikan gambar berikut.

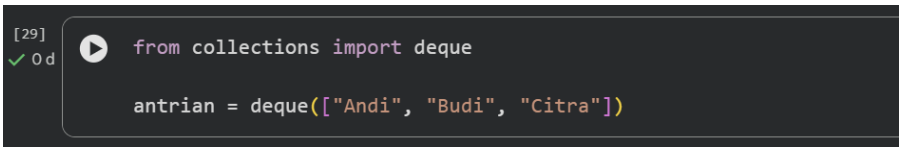


Gambar 4.3. Membuat Queue dengan Dequeue

Berikut merupakan penjelasan kode dari Gambar 4.3.

a. Baris kode from collections import deque digunakan untuk mengambil struktur data deque dari modul collections yang tersedia dalam Python. Modul ini menyediakan berbagai struktur data tambahan yang dapat digunakan untuk mengelola data secara lebih terorganisir, dan deque merupakan salah satu struktur yang dapat digunakan untuk menyimpan data secara berurutan seperti antrean.
b. Baris kode antrian = deque() digunakan untuk membuat sebuah struktur data queue yang masih kosong dan menyimpannya ke dalam variabel antrian. Pada kondisi ini, antrian belum memiliki isi, namun sudah siap digunakan sebagai wadah untuk menampung data yang akan disusun secara berurutan.

Sealin itu, queue juga dapat langsung diisi dengan beberapa data awal, perhatikan gambar berikut.



Gambar 4.4. Membuat Queue dengan Dequeue dengan Beberapa Data Awal

Berikut merupakan penjelasan kode dari Gambar 4.4.

a. Baris kode tersebut digunakan untuk membuat sebuah struktur data deque yang langsung diisi dengan beberapa data awal, yaitu "Andi", "Budi", dan "Citra". Data-data tersebut

disimpan secara berurutan di dalam variabel antrian, sehingga membentuk susunan seperti antrean.

b. Urutan data yang dituliskan di dalam tanda kurung siku [ ] menunjukkan posisi masing-masing elemen dalam struktur queue. Elemen "Andi" berada pada urutan pertama sebagai bagian depan antrean, diikuti oleh "Budi" dan "Citra" yang berada di posisi berikutnya hingga bagian belakang.
c. Dengan cara ini, queue tidak hanya dibuat dalam kondisi kosong, tetapi langsung memiliki isi awal yang dapat digunakan untuk menggambarkan kondisi antrean pada suatu situasi tertentu.

Dalam struktur tersebut, data tetap tersusun secara berurutan, sama seperti pada list. Urutan ini menunjukkan posisi masing-masing elemen dalam antrean.

a. Makna Posisi dalam Deque
   Dalam konteks queue, posisi elemen dalam deque memiliki arti sebagai berikut:
   1) Elemen pertama (indeks 0) menunjukkan bagian depan antrean (front), yaitu data yang berada di urutan awal.
   2) Elemen terakhir menunjukkan bagian belakang antrean (rear), yaitu tempat data baru berada.
      Dengan susunan ini, deque mampu menggambarkan sistem antrean secara lebih jelas sesuai dengan konsep queue.
b. Karakteristik Deque
   Deque memiliki beberapa karakteristik yang mendukung penggunaannya sebagai queue, antara lain:
   1) Data tersusun secara berurutan
   2) Dapat menyimpan banyak elemen dalam satu struktur
   3) Mendukung pengelolaan data dari kedua ujung (depan dan belakang)
c. Kelebihan Membuat Queue dengan Deque
   Penggunaan deque sebagai representasi queue memiliki beberapa kelebihan, yaitu:
   1) Pengelolaan data pada bagian depan dan belakang lebih optimal

2) Tetap sederhana untuk digunakan dalam pemrograman

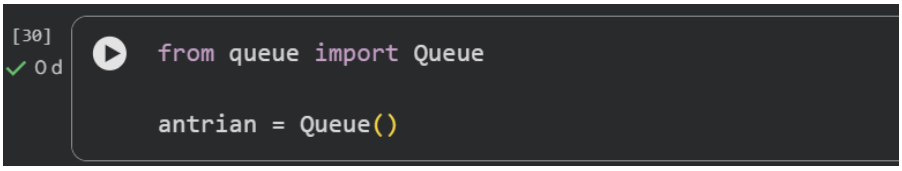
d. Keterbatasan Deque

Meskipun memiliki banyak kelebihan, deque juga memiliki beberapa keterbatasan, antara lain:

1) Memerlukan impor dari modul collections
2) Belum sefleksibel list untuk pengolahan data secara bebas

3. Membuat Queue dengan Modul Queue

Selain menggunakan list dan deque, Python juga menyediakan modul khusus bernama queue yang dapat digunakan untuk merepresentasikan struktur data antrean. Untuk menggunakan modul ini, langkah pertama adalah mengimpor kelas Queue, kemudian membuat objek queue sebagai wadah data. Perhatikan gambar berikut.



Gambar 4.5. Membuat Queue dengan Modul Queue

Berikut merupakan penjelasan kode dari Gambar 4.5.

a. Baris kode from queue import Queue digunakan untuk mengambil kelas Queue dari modul queue yang tersedia dalam Python. Modul ini menyediakan struktur data yang dirancang untuk mengelola data secara teratur, sehingga dapat digunakan untuk merepresentasikan sistem antrean dalam program.
b. Baris kode antrian = Queue() digunakan untuk membuat sebuah objek queue yang masih kosong dan menyimpannya ke dalam variabel antrian. Pada kondisi ini, antrian belum berisi data, tetapi sudah siap digunakan sebagai wadah untuk menyimpan elemen-elemen secara berurutan.

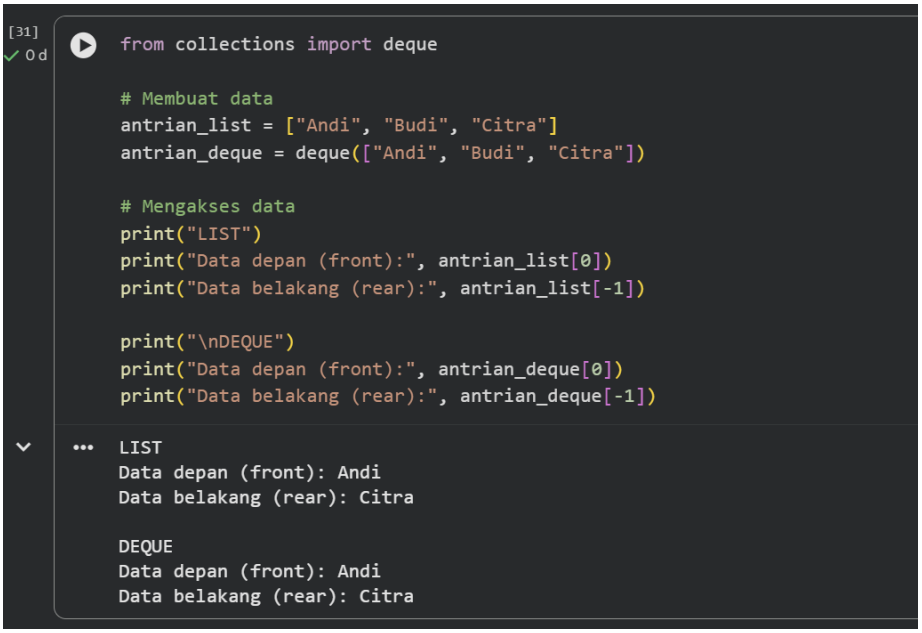
Berdasarkan penjelasan kode di atas, penggunaan modul queue dalam membuat queue (antrean) memiliki beberapa karakteristik, kelebihan, serta keterbatasan yang perlu dipahami.

a. Karakteristik modul queue
   Struktur queue yang dibuat menggunakan modul queue memiliki beberapa karakteristik, antara lain:
   1) Data disimpan secara berurutan sesuai dengan urutan masuknya
   2) Data tidak dapat diakses secara langsung menggunakan indeks
b. Penggunaan modul queue memiliki beberapa kelebihan, antara lain:
   1) Struktur lebih terorganisir
   2) Mendukung pengelolaan data yang lebih kompleks
c. Di sisi lain, terdapat beberapa keterbatasan dalam penggunaan modul ini:
   1) Tidak dapat langsung melihat isi data dengan mudah
   2) Memerlukan pemahaman tambahan untuk penggunaannya

D. Mengakses Data dalam Queue

Langkah berikutnya adalah memahami bagaimana cara mengakses data yang terdapat di dalamnya. Pada queue, data tidak diakses secara bebas, melainkan dilihat berdasarkan posisinya dalam antrean, yaitu bagian depan (front) dan bagian belakang (rear). Bagian depan (front) merupakan posisi data yang berada di urutan pertama. Data pada posisi ini adalah data yang akan diproses terlebih dahulu. Sementara itu, bagian belakang (rear) merupakan posisi data yang berada di urutan terakhir, yaitu tempat data baru berada.

1. Mengakses Data dalam Queue Pada List dan Deque
   Jika queue direpresentasikan menggunakan list atau deque, maka data dapat diakses secara langsung menggunakan indeks. Perhatikan gambar berikut.



```
[31]
✓ 0d

from collections import deque

# Membuat data
antrian_list = ["Andi", "Budi", "Citra"]
antrian_deque = deque(["Andi", "Budi", "Citra"])

# Mengakses data
print("LIST")
print("Data depan (front):", antrian_list[0])
print("Data belakang (rear):", antrian_list[-1])

print("\nDEQUE")
print("Data depan (front):", antrian_deque[0])
print("Data belakang (rear):", antrian_deque[-1])

...
LIST
Data depan (front): Andi
Data belakang (rear): Citra

DEQUE
Data depan (front): Andi
Data belakang (rear): Citra
```

Gambar 4.6. Mengakses Data dalam Queue Pada List dan Deque

Berikut merupakan penjelasan kode dari Gambar 4.6.

a. Baris kode from collections import deque digunakan untuk mengambil struktur data deque dari modul collections, sehingga dapat digunakan sebagai salah satu cara merepresentasikan queue dalam Python.
b. Variabel antrian_list dibuat menggunakan list yang berisi tiga data, yaitu "Andi", "Budi", dan "Citra". Data tersebut tersusun secara berurutan sehingga dapat menggambarkan kondisi antrean.
c. Variabel antrian_deque dibuat menggunakan deque dengan isi data yang sama seperti list, yaitu "Andi", "Budi", dan "Citra". Hal ini bertujuan untuk menunjukkan bahwa kedua struktur memiliki susunan data yang serupa.
d. Perintah print("LIST") digunakan untuk menampilkan judul bagian output yang menunjukkan bahwa data yang ditampilkan berasal dari struktur list.
e. Perintah print("Data depan (front):", antrian_list[0]) digunakan untuk menampilkan elemen pertama pada list, yaitu data yang berada di bagian depan antrean.

f. Perintah print("Data tengah:", antrian_list[1]) digunakan untuk menampilkan elemen yang berada di posisi tengah pada list.
g. Perintah print("Data belakang (rear):", antrian_list[-1]) digunakan untuk menampilkan elemen terakhir pada list, yaitu data yang berada di bagian belakang antrean.
h. Perintah print("\nDEQUE") digunakan untuk memberikan jarak baris (baris kosong) sekaligus menampilkan judul bagian output untuk struktur deque.
i. Perintah print("Data depan (front):", antrian_deque[0]) digunakan untuk menampilkan elemen pertama pada deque, yang menunjukkan bagian depan antrean.
j. Perintah print("Data tengah:", antrian_deque[1]) digunakan untuk menampilkan elemen yang berada di posisi tengah pada deque.
k. Perintah print("Data belakang (rear):", antrian_deque[-1]) digunakan untuk menampilkan elemen terakhir pada deque, yaitu bagian belakang antrean.

2. Mengakses Data dalam Queue Pada Modul Queue
   Pada struktur queue yang menggunakan modul queue, data tidak dapat diakses atau dilihat secara langsung menggunakan indeks. Hal ini berbeda dengan list dan deque yang memungkinkan pengguna untuk melihat isi data tanpa mengubahnya.

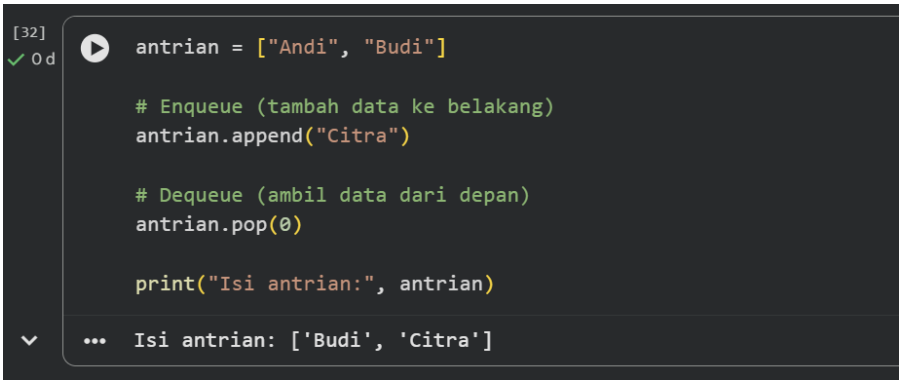
E. Operasi Dasar pada Queue

Operasi dasar pada queue digunakan untuk mengelola data dalam struktur antrean. Operasi ini mencakup proses menambahkan data ke dalam antrean dan mengambil data dari antrean sesuai dengan prinsip First In First Out (FIFO). Secara umum, terdapat dua operasi utama dalam queue, yaitu:

1. Enqueue → menambahkan data ke bagian belakang antrean
2. Dequeue → mengambil data dari bagian depan antrean

Penerapan kedua operasi tersebut dapat dilakukan dengan beberapa cara.

1. Operasi Queue Menggunakan List
   Pada list, operasi queue dilakukan dengan memanfaatkan fungsi bawaan Python. Perhatikan gambar berikut.



Gambar 4.7. Operasi Queue Menggunakan List

Berikut merupakan penjelasan kode dari Gambar 4.7.

a. append() digunakan untuk menambahkan data ke bagian belakang antrean
b. pop(0) digunakan untuk mengambil data dari bagian depan antrean

2. Operasi Queue Menggunakan Deque

Untuk memahami penerapan operasi dasar queue menggunakan deque, perhatikan gambar berikut.

Gambar 4.8. Operasi Queue Menggunakan Deque

Berikut merupakan penjelasan kode dari Gambar 4.8.

a. append() menambahkan data ke belakang antrean
b. popleft() mengambil data dari bagian depan antrean

3. Operasi Queue Menggunakan Modul Queue
   Untuk memahami penerapan operasi dasar queue menggunakan modul queue, perhatikan gambar berikut.

Gambar 4.9. Operasi Queue Menggunakan Modul Queue

Berikut merupakan penjelasan kode dari Gambar 4.9.

a. put() digunakan untuk menambahkan data ke dalam queue
b. get() digunakan untuk mengambil data dari bagian depan antrean

BAB V STRUKTUR DATA STACK

A. Pengertian Stack

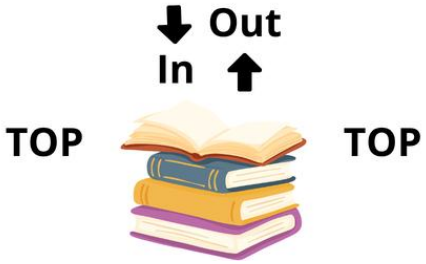
Stack merupakan salah satu struktur data yang digunakan untuk menyimpan sekumpulan data secara berurutan dengan prinsip Last In First Out (LIFO). Prinsip ini berarti data yang terakhir dimasukkan ke dalam stack akan menjadi data pertama yang diambil kembali. Konsep stack dapat dianalogikan seperti tumpukan benda dalam kehidupan sehari-hari, misalnya tumpukan buku atau piring. Ketika sebuah benda ditambahkan, benda tersebut akan diletakkan di bagian paling atas. Sebaliknya, ketika ingin mengambil benda, yang diambil terlebih dahulu adalah benda yang berada di posisi paling atas.

Dalam struktur stack, hanya terdapat satu bagian yang digunakan untuk menambah dan mengambil data, yaitu bagian atas yang disebut sebagai top. Setiap data yang masuk akan ditempatkan di posisi top, dan setiap data yang keluar juga diambil dari posisi tersebut. Dengan mekanisme ini, stack digunakan pada berbagai kondisi yang memerlukan pengolahan

data secara berurutan dari yang terakhir masuk, seperti pada fitur undo dalam aplikasi dan riwayat aktivitas.

B. Cara Kerja Stack

Dalam stack, seluruh proses pengelolaan data dilakukan melalui satu titik, yaitu bagian atas yang disebut sebagai top. Setiap data yang masuk ke dalam stack akan ditempatkan di posisi top, sehingga data tersebut menjadi elemen paling atas. Perhatikan Gambar 5.1.



Gambar 5.1. Cara Kerja Stack

Proses penambahan data ini dikenal sebagai push. Sebaliknya, ketika data akan diambil, pengambilan selalu dilakukan dari posisi top. Data yang berada di paling atas akan keluar terlebih dahulu, kemudian posisi top akan berpindah ke elemen di bawahnya. Proses ini dikenal sebagai pop. Untuk memperjelas alur kerja stack, perhatikan urutan berikut:

1. Kondisi awal: ["Andi", "Budi"]
2. Setelah ditambahkan data baru (push "Citra"): ["Andi", "Budi", "Citra"], posisi top berada pada "Citra"
3. Setelah satu data diambil (pop): ["Andi", "Budi"], data "Citra" keluar terlebih dahulu

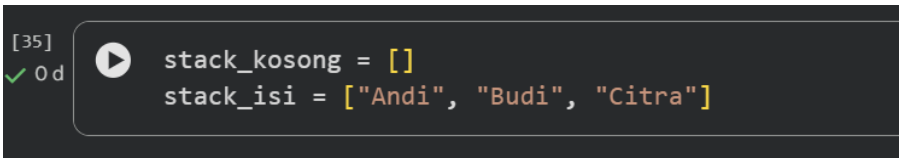
Dari ilustrasi tersebut, terlihat bahwa:

1. Penambahan data selalu dilakukan di bagian atas
2. Pengambilan data juga dilakukan dari bagian atas
3. Data terakhir yang masuk akan menjadi data pertama yang keluar

C. Membuat Stack dengan Python

Terdapat beberapa cara yang dapat digunakan untuk membuat stack di Python, di antaranya menggunakan list dan deque. Kedua pendekatan ini mampu menyimpan data secara berurutan dan mendukung konsep stack.

1. Membuat Stack Menggunakan List
   Salah satu cara paling sederhana untuk merepresentasikan stack adalah dengan menggunakan list. Perhatikan gambar berikut.

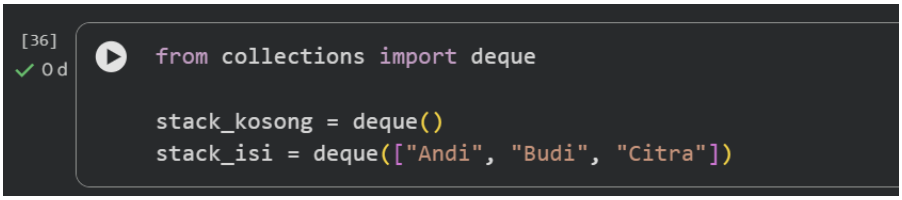


Gambar 5.2. Membuat Stack Menggunakan List

Berikut merupakan penjelasan kode dari Gambar 5.2.

a. Baris kode stack_kosong = [] digunakan untuk membuat sebuah stack yang masih kosong. Pada kondisi ini, belum terdapat data di dalam stack, namun struktur sudah siap digunakan untuk menyimpan data.
b. Baris kode stack_isi = ["Andi", "Budi", "Citra"] digunakan untuk membuat stack yang sudah berisi beberapa data awal. Data tersebut disusun secara berurutan sesuai dengan urutan penulisan.
c. Pada struktur stack_isi, elemen terakhir yaitu "Citra" berada pada posisi paling atas (top) dalam stack, sehingga elemen tersebut akan menjadi data pertama yang diambil apabila dilakukan proses pengambilan data.

2. Membuat Stack Menggunakan Deque
   Selain list, stack juga dapat dibuat menggunakan deque dari modul collections. Perhatikan gambar berikut.



Gambar 5.3. Membuat Stack Menggunakan Deque

Berikut merupakan penjelasan kode dari Gambar 5.3.

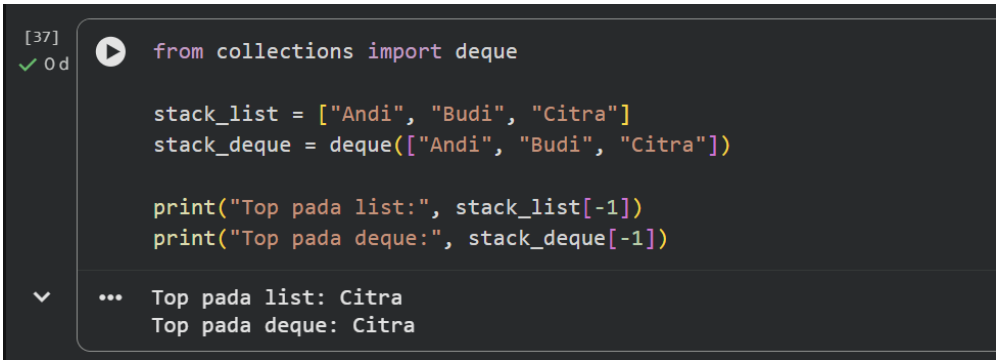
a. Baris kode from collections import deque digunakan untuk mengambil struktur data deque dari modul collections yang

tersedia dalam Python, sehingga dapat digunakan untuk merepresentasikan stack.

b. Baris kode stack_kosong = deque() digunakan untuk membuat sebuah stack yang masih kosong menggunakan deque. Pada kondisi ini, belum terdapat data di dalam stack, namun struktur sudah siap digunakan untuk menyimpan data.
c. Baris kode stack_isi = deque(["Andi", "Budi", "Citra"]) digunakan untuk membuat stack yang sudah berisi beberapa data awal. Data tersebut disusun secara berurutan sesuai dengan urutan penulisan.
d. Pada struktur stack_isi, elemen terakhir yaitu "Citra" berada pada posisi paling atas (top) dalam stack, sehingga elemen tersebut akan menjadi data pertama yang diambil apabila dilakukan proses pengambilan data.

D. Mengakses Data dalam Stack

Setelah stack berhasil dibuat, langkah berikutnya adalah memahami bagaimana cara mengakses data yang terdapat di dalamnya. Berbeda dengan list yang dapat diakses secara bebas menggunakan indeks, pada stack akses data difokuskan pada bagian tertentu sesuai dengan konsep tumpukan. Dalam struktur stack, data yang paling penting untuk diakses adalah data yang berada di bagian atas (top). Hal ini dikarenakan seluruh proses pengambilan data dilakukan dari posisi tersebut. Ketika stack dibuat menggunakan list atau deque, maka data pada bagian atas (top) dapat diakses menggunakan indeks terakhir. Perhatikan gambar berikut.



Gambar 5.4. Mengakses Data dalam Stack

Berikut merupakan penjelasan kode dari Gambar 5.4.

1. Baris kode from collections import deque digunakan untuk mengambil struktur data deque dari modul collections, sehingga dapat digunakan sebagai salah satu cara untuk merepresentasikan stack dalam Python.
2. Variabel stack_list dibuat menggunakan list yang berisi tiga data, yaitu "Andi", "Budi", dan "Citra". Data tersebut tersusun secara berurutan, di mana elemen terakhir merupakan bagian atas (top) dari stack.
3. Variabel stack_deque dibuat menggunakan deque dengan isi data yang sama seperti list. Hal ini bertujuan untuk menunjukkan bahwa kedua struktur dapat digunakan untuk menyimpan data stack dengan susunan yang serupa.
4. Perintah print("Top pada list:", stack_list[-1]) digunakan untuk menampilkan elemen terakhir pada list. Indeks -1 menunjukkan posisi paling akhir, sehingga data yang ditampilkan merupakan bagian atas (top) dari stack.
5. Perintah print("Top pada deque:", stack_deque[-1]) digunakan untuk menampilkan elemen terakhir pada deque. Sama seperti list, indeks -1 menunjukkan posisi paling akhir, yaitu bagian atas (top) dari stack.

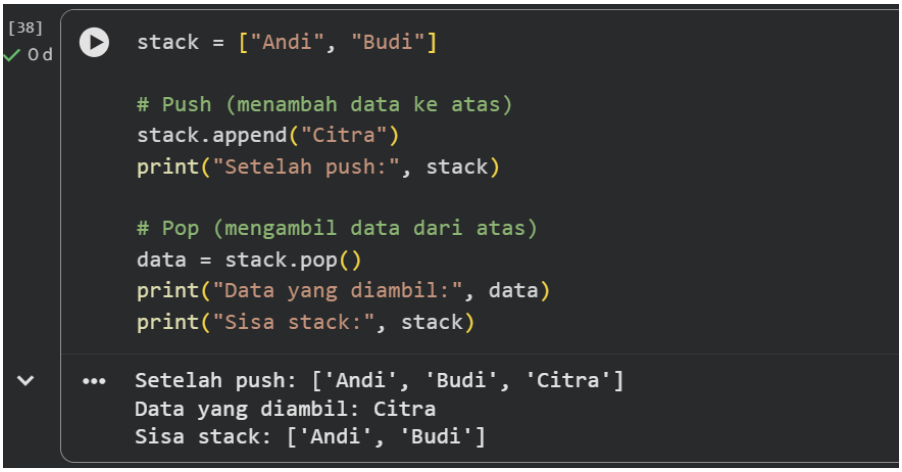
E. Operasi Dasar Pada Stack

Operasi dasar pada stack digunakan untuk mengelola data dalam struktur tumpukan. Operasi ini dilakukan melalui satu titik, yaitu bagian atas (top), sesuai dengan prinsip Last In First Out (LIFO). Terdapat dua operasi utama dalam stack, yaitu:

1. Push → menambahkan data ke bagian atas stack
2. Pop → mengambil data dari bagian atas stack

Berikut merupakan operasi dasar pada stack dengan menggunakan list dan deque.

1. Operasi Stack Menggunakan List
   Pada list, operasi stack dilakukan dengan memanfaatkan fungsi bawaan yang tersedia dalam Python. Melalui fungsi tersebut, data dapat ditambahkan ke bagian atas dan diambil kembali sesuai dengan prinsip stack. Perhatikan gambar berikut.



Gambar 5.5. Operasi Stack Menggunakan List

Berikut merupakan penjelasan kode dari Gambar 5.5.

a. Baris kode stack = ["Andi", "Budi"] digunakan untuk membuat sebuah stack yang berisi dua data awal, yaitu "Andi" dan "Budi". Pada kondisi ini, elemen terakhir yaitu "Budi" berada di bagian atas (top) dari stack.
b. Perintah stack.append("Citra") digunakan untuk menambahkan data baru ke dalam stack, yaitu "Citra". Data yang ditambahkan akan ditempatkan di bagian atas (top) sesuai dengan konsep push.
c. Perintah print("Setelah push:", stack) digunakan untuk menampilkan kondisi stack setelah penambahan data, sehingga terlihat bahwa "Citra" berada di posisi terakhir.
d. Perintah data = stack.pop() digunakan untuk mengambil sekaligus menghapus elemen terakhir pada stack. Data yang diambil merupakan elemen yang berada di bagian atas (top), yaitu "Citra", sesuai dengan konsep pop.
e. Perintah print("Data yang diambil:", data) digunakan untuk menampilkan data yang telah diambil dari stack.
f. Perintah print("Sisa stack:", stack) digunakan untuk menampilkan kondisi stack setelah proses pop, sehingga terlihat bahwa elemen terakhir telah dihapus dan stack kembali berisi "Andi" dan "Budi".

2. Operasi Stack Menggunakan Deque

Pada deque, operasi stack juga dilakukan dengan metode yang serupa. Perhatikan gambar berikut.

```
[39]
✓ 0 d
from collections import deque

stack = deque(["Andi", "Budi"])

# Push
stack.append("Citra")
print("Setelah push:", list(stack))

# Pop
data = stack.pop()
print("Data yang diambil:", data)
print("Sisa stack:", list(stack))

Setelah push: ['Andi', 'Budi', 'Citra']
Data yang diambil: Citra
Sisa stack: ['Andi', 'Budi']
```

Gambar 5.6. Operasi Stack Menggunakan Deque

Berikut merupakan penjelasan kode dari Gambar 5.6.

a. Baris kode from collections import deque digunakan untuk mengambil struktur data deque dari modul collections, sehingga dapat digunakan untuk merepresentasikan stack dalam Python.
b. Baris kode stack = deque(["Andi", "Budi"]) digunakan untuk membuat stack yang sudah berisi dua data awal, yaitu "Andi" dan "Budi". Pada kondisi ini, elemen terakhir yaitu "Budi" berada di bagian atas (top) dari stack.
c. Perintah stack.append("Citra") digunakan untuk menambahkan data baru ke dalam stack, yaitu "Citra". Data yang ditambahkan akan ditempatkan di bagian atas (top) sesuai dengan konsep push.
d. Perintah print("Setelah push:", list(stack)) digunakan untuk menampilkan kondisi stack setelah penambahan data. Fungsi list() digunakan agar data pada deque ditampilkan dalam bentuk yang lebih mudah dibaca.

e. Perintah data = stack.pop() digunakan untuk mengambil sekaligus menghapus elemen terakhir pada stack. Data yang diambil merupakan elemen yang berada di bagian atas (top), yaitu "Citra", sesuai dengan konsep pop.
f. Perintah print("Data yang diambil:", data) digunakan untuk menampilkan data yang telah diambil dari stack.
g. Perintah print("Sisa stack:", list(stack)) digunakan untuk menampilkan kondisi stack setelah proses pop, sehingga terlihat bahwa elemen terakhir telah dihapus dan stack kembali berisi "Andi" dan "Budi".

BAB VI Integrasi List, Stack, dan Queue untuk Sistem Antrean Layanan TU

Dalam sistem antrean layanan TU, berbagai struktur data dapat digabungkan untuk menangani kebutuhan yang berbeda secara efisien:

1. Queue untuk Mengatur Antrean Murid
   Queue (antrian) digunakan untuk menyimpan murid sesuai urutan kedatangan.
   a. Operasi enqueue menambahkan murid ke belakang antrean.
   b. Operasi dequeue menghapus murid dari depan antrean saat dilayani.
   c. Operasi front memungkinkan petugas melihat siapa yang akan dilayani berikutnya.
      Dengan queue, proses layanan menjadi tertib dan adil, sesuai prinsip first-in, first-out (FIFO).
2. List untuk Menyimpan dan Mengelola Data Murid
   List digunakan untuk menyimpan data lengkap murid, termasuk nama dan jenis layanan. List memudahkan operasi tambah (tambah data di posisi tertentu), update (ubah data murid), dan delete (hapus data murid dari posisi manapun). Dengan list, petugas bisa mengedit atau menghapus permintaan layanan tanpa mengganggu urutan murid yang lain.
3. Stack untuk Undo Aktivitas
   Stack digunakan untuk menyimpan riwayat aktivitas petugas, misalnya membatalkan aktivitas pelayanan.

a. Operasi pop memungkinkan membatalkan aktivitas terakhir (undo), sehingga data bisa dikembalikan ke kondisi sebelumnya.

Dengan stack, sistem menyediakan mekanisme koreksi cepat jika terjadi kesalahan pelayanan.

Dengan menggabungkan ketiga struktur data ini, sistem layanan TU menjadi lebih terstruktur, mudah diubah, dan aman dari kesalahan input, sekaligus memperlihatkan bagaimana konsep struktur data dapat diterapkan secara praktis dalam pengelolaan antrean dan catatan layanan. Queue memastikan antrean murid terkelola secara FIFO. List menyimpan data murid yang fleksibel, memungkinkan pengeditan atau penghapusan di posisi manapun. Stack melengkapi sistem dengan kemampuan undo. Implementasi dari integrasi List, Stack, dan Queue ini dapat dilihat pada kode berikut, yang memungkinkan sistem layanan TU untuk menambah, melayani, mengedit, menghapus, dan membatalkan layanan terakhir. Gambar 6.1. merupakan contoh kode untuk integrasi list, queue, dan stack pada sistem antrean TU.

```
[40]
✓ 0d
# ANTREAN TU

# 1. PERSIAPAN LIST (Tempat menyimpan data)
antrean = []
riwayat_undo = []

# --- FUNGSI UNTUK MENGELOLA DATA ---

def tambah_data(data):
    antrean.append(data)
    print(f"[TAMBAH] {data} berhasil masuk antrean.")

def edit_data(nomor_urutan, data_baru):
    index = nomor_urutan - 1 # Mengubah urutan manusia (1,2,3) ke urutan komputer (0,1,2)
    if 0 <= index < len(antrean):
        data_lama = antrean[index]
        antrean[index] = data_baru
        print(f"[EDIT] Data nomor {nomor_urutan} diganti: {data_lama} -> {data_baru}")
    else:
        print("[Gagal] Nomor urutan tidak ada!")

def hapus_data(nomor_urutan):
    index = nomor_urutan - 1
    if 0 <= index < len(antrean):
        data_dibuang = antrean.pop(index)
        print(f"[HAPUS] {data_dibuang} telah keluar dari antrean.")
    else:
        print("[Gagal] Nomor urutan tidak ditemukan!")

def layani_pertama():
    if len(antrean) > 0:
        diproses = antrean.pop(0) # Prinsip FIFO (First-In, First-Out)
        riwayat_undo.append(diproses)
        print(f"[PROSES] Melayani: {diproses}")
    else:
        print("[KOSONG] Antrean sudah habis.")
```

```
    def batal_layanan():
        if len(riwayat_undo) > 0:
            data_balik = riwayat_undo.pop() # Prinsip LIFO (Last-In, First-Out)
            antrean.insert(0, data_balik)
            print(f"[UNDO] Pelayanan {data_balik} dibatalkan, kembali ke posisi 1.")

    def cek_antrean():
        print(f"\n--- DAFTAR ANTREAN SEKARANG: {antrean} ---")

    # BAGIAN MENJALANKAN FUNGSI SISTEM

    print(">>> MEMULAI SISTEM <<<")

    # 1. Masukkan data
    tambah_data("Budi - Legalisir Ijazah")
    tambah_data("Hani - Surat Magang")
    tambah_data("Ari - Surat Cuti")
    cek_antrean()

    # 2. Edit data (Contoh: Ganti Ari jadi Ari Kurniawan)
    # Format: edit_data(nomor_urutan, "nama_baru")
    edit_data(2, "Ari Kurniawan - Surat Cuti")
    cek_antrean()

    # 3. Hapus data (Contoh: Ari Kurniawan batal/keluar antrean)
    hapus_data(2)
    cek_antrean()

    # 4. Layani yang pertama datang (FIFO)
    layani_pertama()
    cek_antrean()

    # 5. Batalkan pelayanan terakhir (UNDO - LIFO)
    batal_layanan()
    cek_antrean()

    print("\n>>> SISTEM SELESAI <<<")
```

Gambar 6.1. Contoh Integrasi Kode untuk Sistem Antrean Layanan TU

```
>>> MEMULAI SISTEM <<<
[TAMBAH] Budi - Legalisir Ijazah berhasil masuk antrean.
[TAMBAH] Hani - Surat Magang berhasil masuk antrean.
[TAMBAH] Ari - Surat Cuti berhasil masuk antrean.

--- DAFTAR ANTREAN SEKARANG: ['Budi - Legalisir Ijazah', 'Hani - Surat Magang', 'Ari - Surat Cuti'] ---
[EDIT] Data nomor 2 diganti: Hani - Surat Magang -> Ari Kurniawan - Surat Cuti

--- DAFTAR ANTREAN SEKARANG: ['Budi - Legalisir Ijazah', 'Ari Kurniawan - Surat Cuti', 'Ari - Surat Cuti'] ---
[HAPUS] Ari Kurniawan - Surat Cuti telah keluar dari antrean.

--- DAFTAR ANTREAN SEKARANG: ['Budi - Legalisir Ijazah', 'Ari - Surat Cuti'] ---
[PROSES] Melayani: Budi - Legalisir Ijazah

--- DAFTAR ANTREAN SEKARANG: ['Ari - Surat Cuti'] ---
[UNDO] Pelayanan Budi - Legalisir Ijazah dibatalkan, kembali ke posisi 1.

--- DAFTAR ANTREAN SEKARANG: ['Budi - Legalisir Ijazah', 'Ari - Surat Cuti'] ---

>>> SISTEM SELESAI <<<
```

Gambar 6.2. Output Kode Sistem Antrean Layanan TU

Berikut merupakan penjelasan dari Gambar 6.1.

1. Inisialisasi Data (Tempat Penyimpanan)
   Di sini kita menyiapkan dua "wadah" utama menggunakan tipe data List:
   a. antrean = []: Digunakan untuk menyimpan daftar orang yang sedang menunggu. Ini mewakili Queue.
   b. riwayat_undo = []: Digunakan untuk mencatat siapa saja yang sudah selesai dilayani. Ini mewakili Stack (untuk keperluan pembatalan).
2. Fungsi Utama (Logika Program)
   a. Menambah Antrean (tambah_data)
      1) Logika: Menggunakan .append().
      2) Penjelasan: Menambahkan data ke urutan paling belakang (akhir list). Sama seperti orang yang baru datang ke TU, mereka harus berdiri di barisan paling belakang.
   b. Melayani Antrean (layani_pertama)
      1) Konsep: FIFO (First-In, First-Out).
      2) Logika: Menggunakan .pop(0).
      3) Penjelasan: Petugas akan memanggil orang yang berada di barisan paling depan (indeks 0). Data yang diambil kemudian dipindahkan ke riwayat_undo agar jika ada kesalahan, kita masih punya catatannya.
   c. Membatalkan Pelayanan (batal_layanan)
      1) Konsep: LIFO (Last-In, First-Out).
      2) Logika: Menggunakan .pop() (mengambil yang terakhir) dan .insert(0, ...) (mengembalikan ke depan).
      3) Penjelasan: Jika petugas salah panggil, fitur Undo ini mengambil data terakhir dari riwayat, lalu memasukkannya kembali ke posisi nomor 1 di antrean.
   d. Manipulasi Data (edit_data & hapus_data)
      Logika: Akses berbasis Index.
      Penjelasan: Karena komputer menghitung dari 0, sedangkan manusia dari 1, maka ada baris index = nomor_urutan - 1.
3. Alur Simulasi (Step-by-Step)
   Dalam bagian praktek, berikut hal yang terjadi:

a. Input: Budi, Hani, dan Ari masuk. Antrean: ['Budi', 'Hani', 'Ari'].
b. Edit: edit_data(2, ...) mengubah urutan ke-2 (Hani) menjadi Ari Kurniawan.
c. Hapus: hapus_data(2) menghapus orang di urutan ke-2. Sekarang sisa Budi dan Ari (Ari bergeser maju).
d. Proses (FIFO): layani_pertama() mengambil Budi (urutan pertama). Sisa antrean: ['Ari'].
e. Undo (LIFO): batal_layanan() mengambil Budi dari riwayat dan menaruhnya kembali ke depan. Antrean kembali menjadi: ['Budi', 'Ari'].

kode pada Gambar 6.1 juga dapat digunakan sebagai acuan dalam membuat simulasi sistem layanan antrean lainnya.