ALGORITMA & STRUKTUR DATA

dengan

Bahasa Pemrograman Python

Anggri Yulio Pernanda | Rahayu Trisetyowati Untari |Edwar Rosman

ISBN 978-623-5299-29-7

Algoritma dan Struktur Data dengan Bahasa Pemrograman Python

Anggri Yulio Pernanda

Rahayu Trisetyowati Untari

Edwar Rosman

PENERBIT
CV. HAQI PARADISE
MEDIATAMA

Algoritma dan Struktur Data dengan Bahasa Pemrograman Python

Anggri Yulio Pernanda, Rahayu Trisetyowati Untari, Edwar Rosman

ISBN: 978-623-5299-29-7

Editor:
Hariz

Foto:
Rahayu Trisetyowati Untari

Desain Sampul :
Edwar Rosman

Ilustrasi Dalam:
Cv. Haqi Paradise Mediatama

Tata Layout:
Trisno

Penerbit:
Cv. Haqi Paradise Mediatama

Kantor Pusat:
Jl. Bundo Kanduang No 1 Padang *Phonecell*/Telp: 085365372924/ (0751) 7053731. Email: hrzm2f @gmail.com
Cetakan Pertama, 2024

Kata Pengantar

Dengan rendah hati dan penuh rasa syukur, kami ingin menyampaikan ucapan terima kasih yang tulus kepada Tuhan Yang Maha Esa atas berkah-Nya yang melimpah, sehingga buku berjudul "Algoritma dan Struktur Data dengan Python" ini dapat kami hadirkan untuk dinikmati oleh berbagai kalangan pembaca.

Buku ini telah dirancang dengan tujuan untuk memberikan pemahaman yang kokoh tentang konsep-konsep dasar dalam ilmu komputer, terutama terkait dengan algoritma dan struktur data, dengan menggunakan bahasa pemrograman Python yang populer dan mudah dipahami.

Dalam era yang semakin kompleks dan terhubung, pengetahuan yang solid tentang algoritma dan struktur data menjadi semakin penting. Buku ini akan membimbing pembaca melalui berbagai topik, mulai dari konsep dasar seperti list, tuple, dan set, hingga algoritma pencarian dan pengurutan yang lebih kompleks seperti binary search dan merge sort.

Salah satu keunggulan Python adalah kemudahan sintaksisnya dan fleksibilitas yang tinggi, sehingga kami memilihnya sebagai bahasa pemrograman utama dalam buku ini. Dengan demikian, pembaca tidak hanya akan memahami konsep-konsep teoritis, tetapi juga dapat menerapkannya langsung dalam kode Python yang nyata.

Terakhir, penulis mengucapkan terima kasih kepada semua pembaca yang telah memilih buku ini sebagai panduan mereka dalam mempelajari algoritma dan struktur data dengan Python. Semoga buku ini memberikan pengalaman pembelajaran yang bermanfaat dan memuaskan bagi Anda semua.

Padang, Juni 2024

Penulis.

Daftar Isi

Daftar Gambar

Daftar Tabel

1 Pengenalan Algoritma

1.1 Dasar Teori

Algoritma adalah langkah-langkah terstruktur atau prosedur logis yang dirancang untuk menyelesaikan suatu masalah atau melakukan tugas tertentu. Dalam dunia komputasi, algoritma berperan sebagai panduan atau resep yang merinci cara melakukan suatu pekerjaan secara sistematis. Setiap langkah dalam algoritma harus jelas, dapat dilaksanakan, dan menghasilkan output yang benar.

Beberapa aspek kunci dari algoritma meliputi:

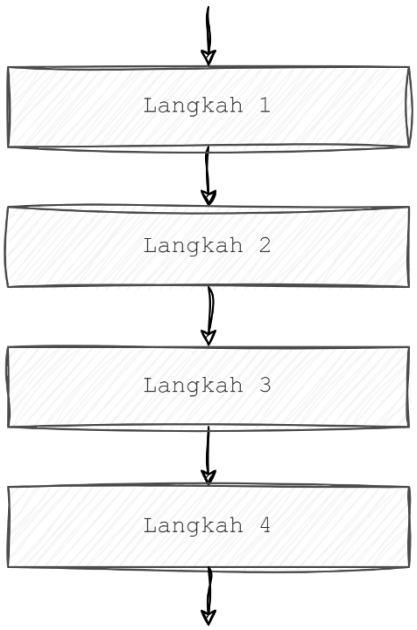
- Input: Algoritma menerima input, yaitu data atau informasi yang diperlukan untuk memulai proses.
- Proses: Algoritma melibatkan serangkaian langkah-langkah yang diterapkan pada input untuk menghasilkan output. Setiap langkah harus bersifat konkret dan dapat dilaksanakan.
- Output: Setelah proses selesai, algoritma menghasilkan output atau solusi yang merupakan hasil akhir dari eksekusi langkah-langkah tersebut.
- Terminasi: Algoritma harus berakhir setelah sejumlah langkah tertentu, dan tidak boleh terus berjalan tanpa batas waktu. Ini dikenal sebagai sifat terminasi.
- Efisien: Algoritma idealnya dirancang untuk mencapai tujuannya dengan cara yang paling efisien mungkin, meminimalkan penggunaan sumber daya seperti waktu atau memori.

1.2 Struktur Dasar Algoritma

Struktur dasar dari suatu algoritma dapat dibagi menjadi tiga komponen utama yang membentuk kerangka kerja untuk merancang dan memahami algoritma, yaitu:

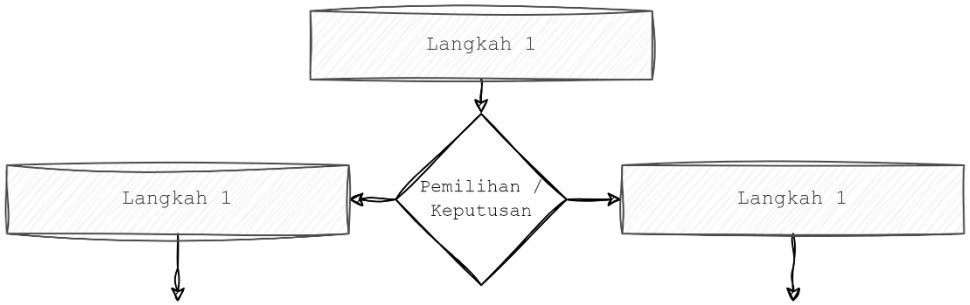
1. Urutan (Sequential)

   Struktur urutan menggambarkan langkah-langkah yang dijalankan secara berurutan, satu per satu. Setiap langkah dieksekusi setelah langkah sebelumnya selesai. Ini mencerminkan aliran eksekusi dari atas ke bawah.



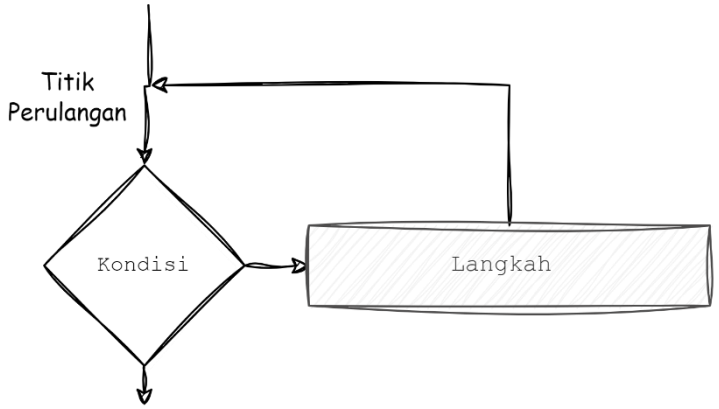
2. Pemilihan (Selection/Decision):

   Struktur pemilihan digunakan untuk membuat keputusan berdasarkan kondisi tertentu. Pada bagian ini, algoritma dapat beralih ke langkah-langkah tertentu tergantung pada apakah kondisi yang ditentukan benar atau salah.



3. Perulangan (Iteration/Looping):

   Struktur perulangan digunakan untuk mengulang langkah-langkah tertentu selama kondisi tertentu masih terpenuhi. Ini memungkinkan pengulangan tugas atau operasi yang sama.



1.3 Kenapa Perlu Mempelajari Algoritma

Belajar algoritma memiliki sejumlah alasan yang sangat penting, terutama jika kita tertarik dalam bidang ilmu komputer, teknologi informasi, atau pengembangan perangkat lunak. Berikut adalah beberapa alasan mengapa belajar algoritma sangat bermanfaat:

- Algoritma merupakan dasar dari pemrograman. Setiap program komputer, baik yang sederhana maupun kompleks, pasti menggunakan algoritma.
- Algoritma membantu untuk memecahkan masalah secara efektif. Algoritma yang efisien dapat menyelesaikan masalah dengan cepat dan menggunakan sumber daya yang sedikit.
- Algoritma dapat diterapkan dalam berbagai bidang. Algoritma tidak hanya digunakan dalam pemrograman, tetapi juga dalam berbagai bidang lainnya, seperti matematika, sains, dan bisnis.

Selain itu pemahaman tentang algoritma secara tidak langsung juga akan meningkatkan kemampuan diri seorang yang, seperti:

- Meningkatkan kemampuan berpikir logis dan sistematis. Algoritma adalah urutan langkah-langkah yang logis dan sistematis untuk menyelesaikan masalah. Oleh karena itu, mempelajari algoritma dapat membantu untuk meningkatkan kemampuan berpikir logis dan sistematis kita.
- Mengembangkan keterampilan pemecahan masalah. Algoritma dapat membantu untuk menemukan solusi yang efektif untuk masalah yang kompleks. Oleh karena itu, mempelajari algoritma dapat membantu untuk mengembangkan keterampilan pemecahan masalah kita.
- Meningkatkan kemampuan berpikir kritis. Algoritma dapat membantu untuk menilai kelayakan dari solusi yang berbeda. Oleh karena itu, mempelajari algoritma dapat membantu untuk meningkatkan kemampuan berpikir kritis kita.
- Meningkatkan kreativitas. Algoritma dapat digunakan untuk menghasilkan solusi yang kreatif untuk masalah yang kompleks. Oleh

karena itu, mempelajari algoritma dapat membantu untuk meningkatkan kreativitas kita.

1.4 Notasi Algoritma

Notasi algoritma adalah cara untuk menuliskan algoritma dengan menggunakan bahasa yang mudah dibaca dan dipahami oleh manusia. Notasi algoritma tidak mengacu pada bahasa pemrograman tertentu, sehingga dapat digunakan untuk menjelaskan algoritma apa pun. Secara umum terdapat 3 jenis notasi algoritma, yaitu:

1. **Notasi deskriptif** adalah notasi algoritma yang menggunakan bahasa alami untuk menjelaskan langkah-langkah algoritma. Notasi deskriptif cocok untuk algoritma yang sederhana dan mudah dipahami.
2. **Pseudocode** adalah notasi algoritma yang menggunakan bahasa yang menyerupai bahasa pemrograman untuk menjelaskan langkah-langkah algoritma. Pseudocode lebih formal daripada notasi deskriptif, sehingga lebih mudah untuk diterjemahkan ke dalam bahasa pemrograman tertentu.
3. **Flowchart** adalah notasi algoritma yang menggunakan diagram untuk menjelaskan langkah-langkah algoritma. Flowchart lebih visual daripada notasi deskriptif dan pseudocode, sehingga lebih mudah untuk dipahami.

Contoh kasus:

Anda akan membuat sebuah perangkat lunak sederhana yang akan menghitung luas sebuah persegi panjang, sebelum Membuat program tersebut anda diminta untuk Membuat algoritma menggunakan notasi deskriptif, pseudocode dan flowchart.

Penyelesaian:

Notasi Deskriptif

1. Mulai
2. Input panjang dan lebar dari persegi panjang
3. Hitung luas persegi panjang dengan rumus: luas = panjang x lebar
4. Simpan hasil perhitungan luas dalam variabel hasil
5. Tampilkan hasil luas persegi panjang
6. Selesai

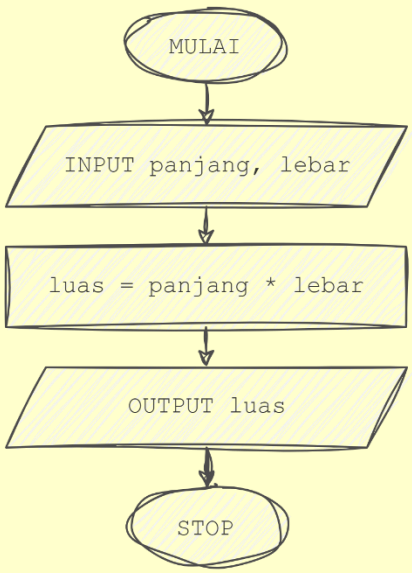
Pseudocode

```
ALGORITMA MenghitungLuasPersegiPanjang
MULAI
   INPUT "Masukkan panjang persegi panjang:", panjang
   INPUT "Masukkan lebar persegi panjang:", lebar

   luas = panjang * lebar

   OUTPUT "Luas persegi panjang adalah:", luas
SELESAI
```

Flowchart



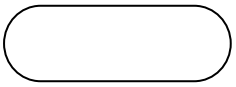
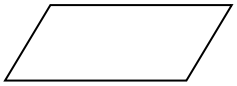
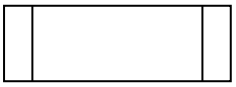
Gambar 1 Flowchart Menghitung Luas Persegi Panjang

2 lowchart & Pseudocode

2.1 Flowchart

Flowchart adalah representasi grafis dari serangkaian langkah-langkah atau prosedur yang digambarkan dengan menggunakan simbol-simbol grafis. Flowchart digunakan untuk menggambarkan alur kerja suatu proses atau algoritma dengan cara yang mudah dimengerti. Dengan kata lain, flowchart merupakan bentuk visual dari notasi deskriptif dalam algoritma.

Simbol-simbol pada flowchart menggambarkan langkah-langkah, keputusan, atau operasi khusus dalam suatu proses.

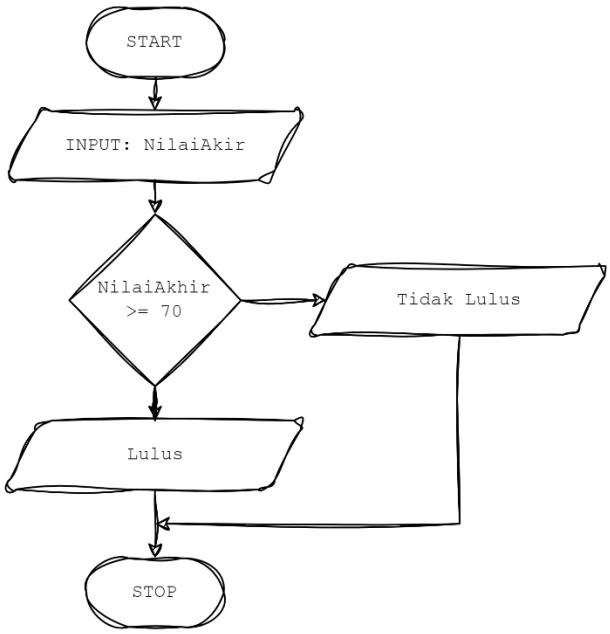
Simbol	Keterangan
	Terminal: digunakan untuk menunjukkan awal dan akhir satu set proses yang berhubungan dengan komputer
	Input / Output: digunakan untuk menunjukkan operas input / output
	Pemrosesan komputer: digunakan untuk menunjukkan pemrosesan apa pun yang dilakukan oleh sistem komputer
	*Predefined processing*: digunakan untuk menunjukkan proses apa pun yang tidak didefinisikan secara khusus dalam diagram alur
	Input / Output Dokumen: digunakan ketika input berasal dari dokumen dan output masuk ke dokumen.
	Keputusan: digunakan untuk menunjukkan titik mana pun dalam proses di mana keputusan harus dibuat untuk menentukan tindakan lebih lanjut
	On-page connector: digunakan untuk menghubungkan bagian-bagian diagram alur yang dilanjutkan pada halaman yang sama

	Off-page connector: digunakan untuk menghubungkan bagian-bagian diagram alur pada halaman terpisah
	Garis aliran: digunakan untuk menghubungkan simbol

Tabel 1 Simbol Flowchart

Flowchart (diagram alir) dapat digunakan untuk menunjukkan urutan langkah-langkah untuk melakukan pekerjaan apa pun. Secara sederhana flowchart menggambarkan bagaimana suatu operasi menerima masukan, melakukan operasi aritmatika pada masukan, dan menampilkan keluaran kepada pengguna. Berikut adalah beberapa contoh diagram alir:

Contoh 1. Menentukan status kelulusan mahasiswa

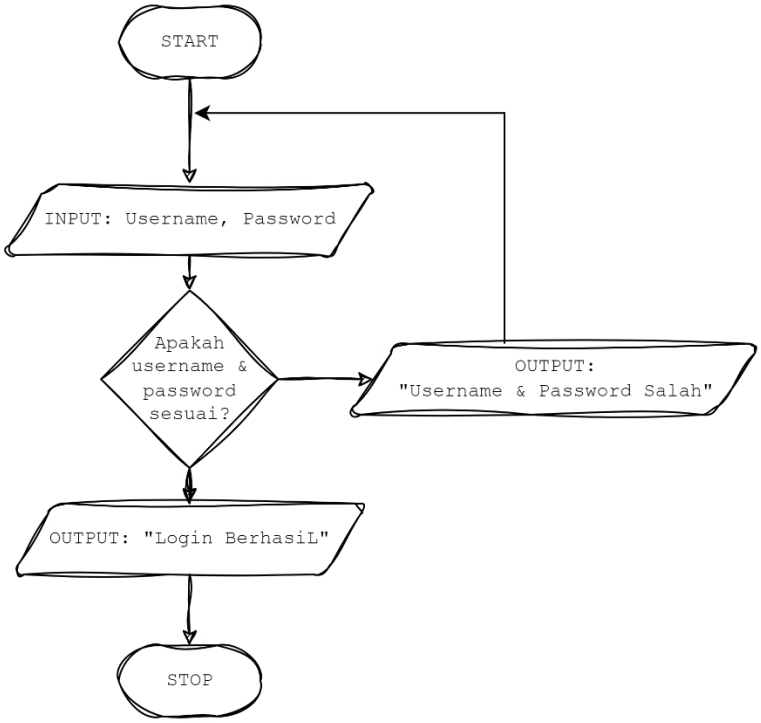


Gambar 2 Flowchart Status Kelulusan Mahasiswa

Pada gambar diatas bisa dilihat bahwa input dari proses pada diagram alir berupa variable **NilaiAkhir,** berdasarkan variable tersebut dilakukan proses

decision (Keputusan) jika nilai besar atau sama dengan 70 maka mahasiswa tersebut lulus jika kurang maka mahasiswa tersebut tidak lulus.

Contoh 2. Proses login pada sebuah system



Gambar 3 Diagram Alir Proses Login

Diagram diatas merupakan proses login pada sebuah system, Dimana pengguna diminta untuk memasukkan username dan password, kemudian sistem akan melakukan pemeriksaan apakah username dan password tersebut sesuai atau tidak. Jika ditemukan bahwa password tersebut tidak sesuai, maka pengguna diminta untuk memasukkan kembali username dan password.

2.2 Pseudocode

Pseudocode adalah cara untuk mendeskripsikan langkah-langkah suatu algoritma atau prosedur menggunakan bahasa semi-formal yang mirip dengan bahasa pemrograman, tetapi lebih ringan dan fleksibel. Dalam pseudocode, kita menggunakan bahasa manusia yang mudah dipahami untuk menyusun langkah-langkah suatu tugas tanpa terikat dengan sintaksis atau aturan bahasa pemrograman tertentu.

Pseudocode tidak memiliki aturan baku yang ketat, tetapi ada beberapa pedoman umum yang dapat membantu agar pseudocode lebih mudah dimengerti dan dapat diinterpretasikan dengan baik. Berikut adalah beberapa aturan penulisan pseudocode:

1. Gunakan Bahasa yang Jelas dan Sederhana agar pseudocode dapat dimengerti oleh berbagai pembaca, termasuk mereka yang mungkin tidak memiliki latar belakang pemrograman yang kuat.
2. Identifikasi Variabel dengan jelas dan sediakan deskripsi singkat tentang peran atau isi dari masing-masing variabel.
3. Gunakan Notasi Standar seperti IF, ELSE, FOR, WHILE, DO, END, dan lainnya untuk menunjukkan struktur kontrol dan perulangan.
4. Indentasi untuk Menunjukkan Struktur Blok dalam pseudocode, seperti yang umumnya dilakukan dalam pemrograman.
5. Hindari Detil Implementasi atau sintaksis bahasa pemrograman tertentu. Fokuskan pada konsep dan logika algoritma.
6. Jelaskan Setiap Langkah atau pernyataan dalam pseudocode untuk membantu pemahaman.
7. Gunakan Komentar: atau catatan untuk memberikan penjelasan tambahan atau konteks jika diperlukan.
8. Gunakan notasi aritmetika yang umum seperti +, -, *, / untuk operasi matematika.

9. Jelaskan langkah-langkah input dan output dengan jelas, misalnya INPUT dan OUTPUT.
10. Hindari terlalu rinci dalam deskripsi langkah-langkah. Pseudocode seharusnya memberikan gambaran umum tanpa terlalu mendetail pada level implementasi.

Berikut adalah contoh sederhana dalam penulisan pseudocode.

```
START
    INPUT "Masukkan nilai A:", A
    INPUT "Masukkan nilai B:", B

    IF A > B THEN
        OUTPUT "Nilai A lebih besar dari B"
    ELSE
        OUTPUT "Nilai B lebih besar atau sama dengan A"
    ENDIF

    FOR i FROM 1 TO 10
        OUTPUT "Perulangan ke-", i
    ENDFOR

    WHILE A > 0
        OUTPUT "Nilai A masih positif"
        A = A - 1
    ENDWHILE

    OUTPUT "Selesai"
END
```

Untuk lebih jelasnya kita akan melakukan penulisan pseudocode untuk contoh kasus diagram alir pada Gambar 2

```
START
    INPUT "Masukkan Nilai Akhir:", NilaiAKhir

    IF NilaiAkhir >= 70 THEN
        OUTPUT "LULUS"
    ELSE
        OUTPUT "TIDAK LULUs"
    ENDIF
END
```

3 Bahasa Pemrograman Python

3.1 Pengenalan Python

Python adalah bahasa pemrograman tingkat tinggi, interpretatif, dan serbaguna yang dirancang untuk keterbacaan kode dan produktivitas. Diciptakan oleh Guido van Rossum dan pertama kali dirilis pada tahun 1991, Python telah menjadi salah satu bahasa pemrograman yang paling populer di dunia. Berikut adalah beberapa karakteristik utama dari Python:

1. Sintaksis yang Bersih dan Mudah Dipahami:
   Python dirancang dengan sintaksis yang bersih dan mudah dipahami, memungkinkan pengembang untuk mengekspresikan ide dengan jumlah baris kode yang relatif sedikit.
2. Bahasa Pemrograman Tingkat Tinggi:
   Python adalah bahasa pemrograman tingkat tinggi, yang berarti pengembang dapat fokus pada logika program tanpa harus terlalu khawatir tentang detail tingkat rendah.
3. Interpretatif dan Platform Agnostik:
   Python diinterpretasikan, yang berarti kode dapat dieksekusi baris per baris tanpa memerlukan kompilasi sebelumnya. Python juga bersifat platform agnostik, artinya dapat berjalan di berbagai sistem operasi seperti Windows, macOS, dan Linux.
4. Dukungan untuk Berbagai Paradigma Pemrograman:
   Python mendukung paradigma pemrograman yang beragam, termasuk pemrograman prosedural, pemrograman berorientasi objek, dan pemrograman fungsional.
5. Mudah untuk Dipelajari:

Python dianggap sebagai salah satu bahasa pemrograman yang mudah dipelajari, menjadikannya pilihan yang baik bagi pemula dan pengembang berpengalaman.

6. Libraries dan Frameworks yang Kuat:

   Python memiliki ekosistem yang kaya dengan banyak perpustakaan dan kerangka kerja (framework) yang mendukung berbagai bidang seperti pengembangan web (Django, Flask), ilmu data (NumPy, Pandas), kecerdasan buatan (TensorFlow, PyTorch), dan banyak lagi.

7. Komunitas yang Besar dan Aktif:

   Python memiliki komunitas pengembang yang besar dan aktif, yang berarti terdapat banyak sumber daya, dokumentasi, dan dukungan di seluruh dunia.

8. Pemeliharaan Kode yang Mudah:

   Keterbacaan kode adalah salah satu prinsip desain Python. Dengan menggunakan indentasi sebagai bagian integral sintaksisnya, Python mendorong pemeliharaan kode yang bersih dan mudah dimengerti.

9. Dukungan untuk Skriping dan Automasi:

   Python sering digunakan untuk skriping (scripting) dan otomasi tugas-tugas rutin karena kemampuannya yang cepat dan mudah untuk diimplementasikan.

10. Open Source:

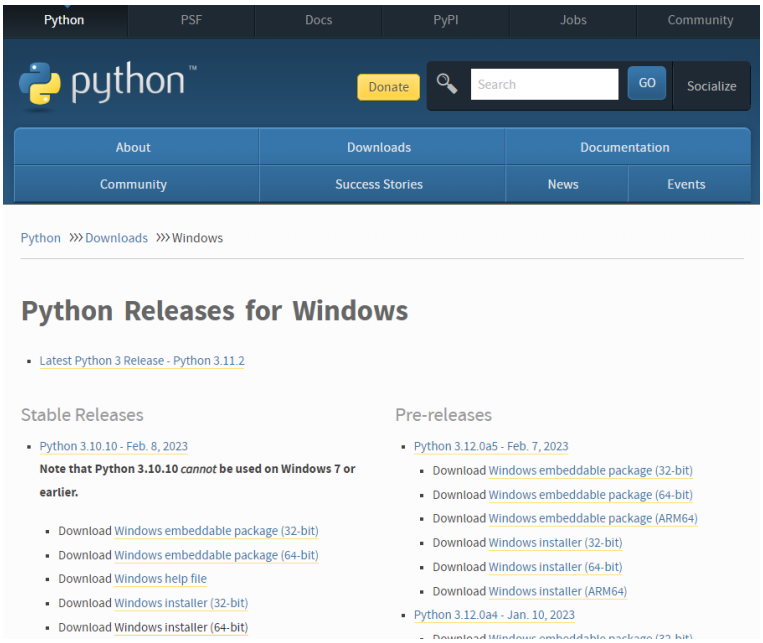
    Python bersifat open source, artinya kode sumbernya dapat diakses, dimodifikasi, dan didistribusikan secara bebas.

3.2 Instalasi Python

1. Unduh paket instalasi Python

Buka halaman resmi Python untuk mengunduh python versi Windows (https://www.python.org/downloads/windows), temukan rilis Python 3 yang stabil (panduan ini menggunakan Python versi 3.10.10.).

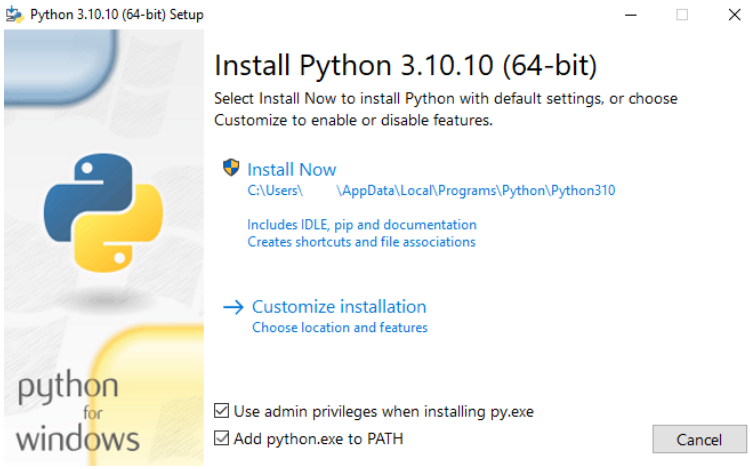
Klik tautan yang sesuai untuk sistem Anda untuk mengunduh file yang dapat dieksekusi: penginstal Windows (64-bit) atau penginstal Windows (32-bit).



Gambar 4 Situs resmi Python

2. Jalankan paket instalasi yang telah diunduh

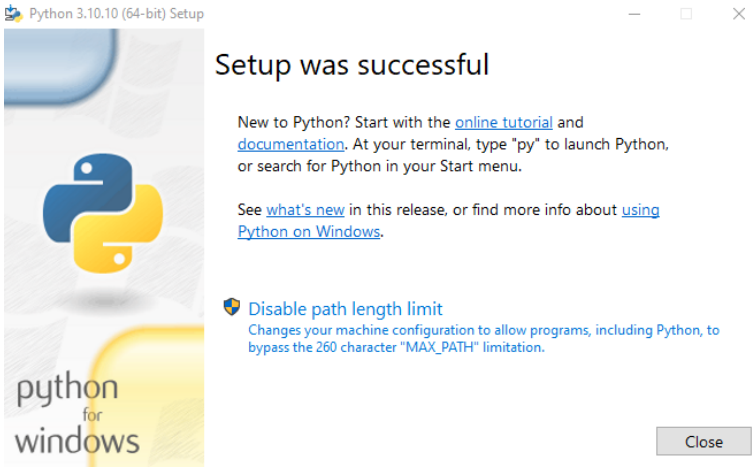
   Setelah penginstal diunduh, klik dua kali file .exe, misalnya python-3.10.10-amd64.exe, untuk menjalankan penginstal Python. Centang **Install launcher for all users** , yang memungkinkan semua pengguna komputer mengakses aplikasi Python. Pilih kotak centang Add python.exe to PATH, agar kita dapat menjalankan Python dari baris perintah (cmd).



Gambar 5 Proses instalasi Python

Jika Anda baru mempelajari Python dan Anda ingin menginstalnya dengan fitur default, maka klik **Install Now**

3. Lanjutkan proses instalasi secara default. Seteleh proses instalasi selesai maka akan muncul jendela **Setup was successful**



Gambar 6 Proses akhir instalasi python

4. Verifikasi Instalasi Python

Anda dapat memverifikasi apakah instalasi Python berhasil melalui Command Line (cmd) atau melalui aplikasi Integrated Development Environment (IDLE).

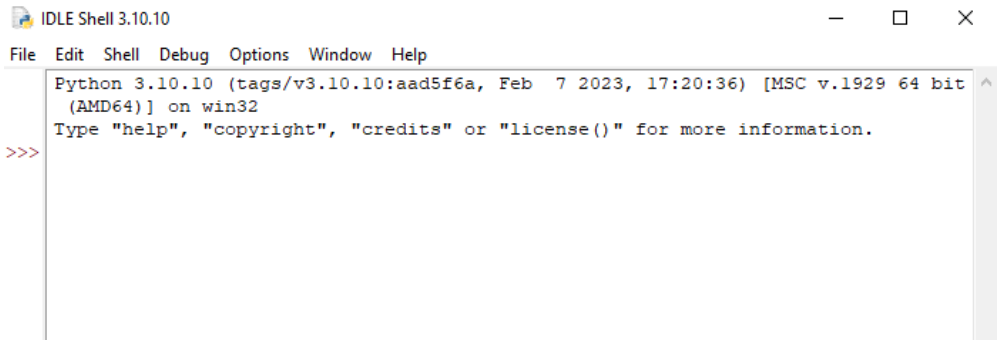
Buka menu **Start** dan ketikan **cmd** di bilah pencarian. Pilih **Command Prompt**. Lalu masukkan perintah berikut di prompt perintah:

```
python --version
```

Jika semua proses instalasi berhasil dilakukan maka akan muncul versi Python pada jendela Command Prompt

```
Python 3.10.10
```

Selain itu kita juga dapat memeriksa versi Python dengan membuka aplikasi IDLE. Buka menu Start dan masukkan python di bilah pencarian lalu klik aplikasi IDLE, misalnya IDLE (Python 3.10 64-bit).



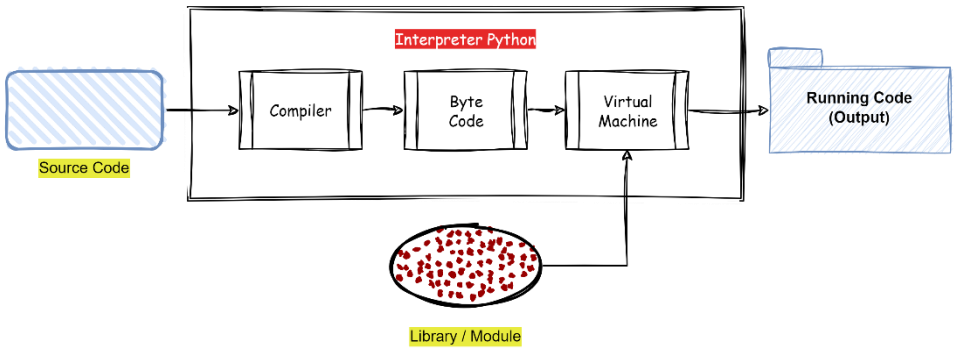
Gambar 7 Integrated Development and Learning Environment (IDLE)

3.3 Mekanisme Kerja Python

Python bekerja dengan membaca kode Python yang ditulis oleh programmer. Kode Python ini kemudian dikompilasi menjadi kode mesin yang dapat dijalankan oleh komputer. Proses kompilasi kode Python dilakukan oleh

interpreter Python. Interpreter Python adalah perangkat lunak yang membaca kode Python dan menerjemahkannya ke dalam kode mesin.

Kode mesin ini kemudian dapat dijalankan oleh komputer. Komputer akan menjalankan kode mesin ini langkah demi langkah untuk menghasilkan hasil yang diinginkan. Untuk lebih jelasnya bisa dilihat pada gambar berikut.



Gambar 8 Mekanisme kerja Python

3.4 Gaya Penulisan Kode Program [Pythonic]

Gaya penulisan Pythonic mengacu pada konvensi dan praktik terbaik dalam menulis kode Python agar sesuai dengan filosofi dan panduan pemrograman Python. Tujuan dari gaya penulisan Pythonic adalah untuk membuat kode lebih mudah dibaca, mudah dimengerti, dan lebih konsisten. Panduan utama untuk gaya penulisan Pythonic dapat ditemukan dalam "PEP 8" (Python Enhancement Proposal 8), yang merupakan panduan resmi untuk gaya penulisan Python. Berikut adalah beberapa aspek kunci dari gaya penulisan Pythonic:

1. Indentasi

   Gunakan indentasi empat spasi untuk setiap tingkat indentasi. Ini adalah konvensi standar dalam Python.

2. Spasi dan Pemisahan

   Hindari penggunaan spasi ekstra di akhir baris atau di dalam kurung kurawal. Gunakan satu spasi sebelum dan sesudah operator, kecuali dalam parameter fungsi.

3. Panjang Baris

   Batasi panjang baris kode hingga 79 karakter, kecuali jika memang diperlukan untuk kejelasan.

4. Penamaan Variabel dan Fungsi

   Gunakan huruf kecil dan pisahkan kata-kata dengan garis bawah untuk nama variabel dan fungsi (snake_case). Hindari penggunaan nama variabel yang terlalu umum seperti 'l' atau 'O'.

5. Komentar

   Gunakan komentar dengan bijak untuk menjelaskan kode yang kompleks atau sulit dimengerti. Hindari komentar yang jelas dan tidak memberikan nilai tambah.

6. Import

   Pisahkan impor sesuai dengan kategori dan gunakan baris kosong antara grup impor. Hindari penggunaan impor umum seperti import *.

7. Dokumentasi

   Sertakan dokumentasi string (docstring) yang jelas untuk fungsi, modul, dan kelas. Gunakan format docstring yang sesuai dengan PEP 257.

8. Manipulasi String

   Gunakan metode string seperti join() daripada penggabungan string menggunakan operator +.

9. List Comprehension

   Gunakan list comprehension untuk membuat dan memanipulasi list dengan cara yang ringkas dan mudah dibaca.

10. Exception Handling

    Hindari penggunaan **except:** tanpa spesifikasi jenis exception. Gunakan blok **try-except** dengan spesifik exception yang hendak ditangani.

11. Iterasi

    Gunakan fungsi bawaan Python seperti **enumerate()** untuk mendapatkan indeks dan nilai dalam loop.

12. Boolean Evaluations

    Gunakan ekspresi boolean yang sederhana dan jelas. Hindari perbandingan yang berlebihan.

13. Menggunakan 'is' dan 'is not':

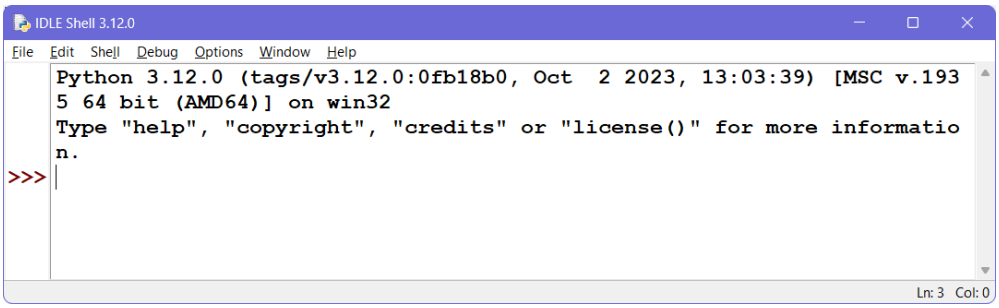
    Gunakan is dan is not untuk memeriksa identitas objek (kecocokan objek), bukan == atau != yang digunakan untuk memeriksa kesamaan nilai.

Penerapan gaya penulisan Pythonic tidak hanya membuat kode lebih mudah dipahami oleh pembaca manusia tetapi juga membantu memastikan bahwa kode Anda konsisten dengan praktik terbaik di komunitas Python. Mengikuti panduan PEP 8 adalah langkah awal yang baik untuk mengembangkan kode Pythonic.

3.5 Menulis dan Menjalankan Program Python

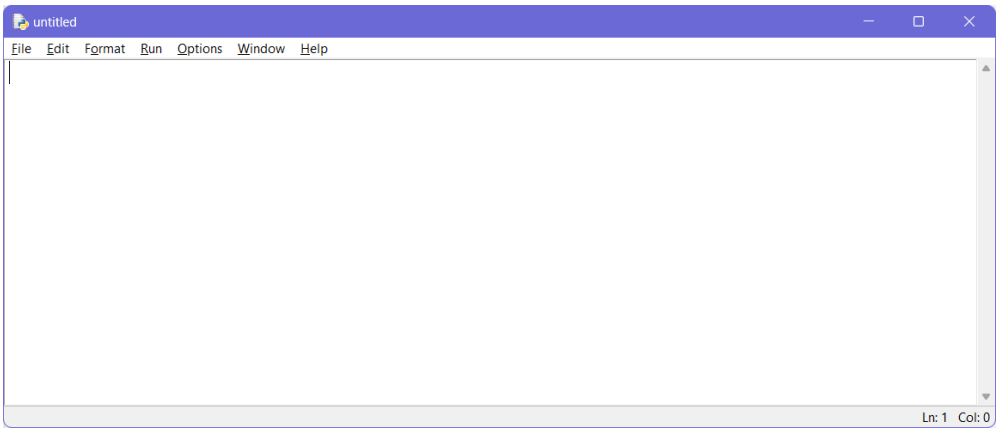
Setelah anda selesai melakukan instalasi, secara default paket instalasi tersebut juga akan menginstall sebuah aplikasi Gambar 7 Integrated Development and Learning Environment (IDLE). Aplikasi ini bisa kita manfaatkan sebagai code editor python. Aplikasi ini bisa ditemukan pada menu **Start >> IDLE**

Buka aplikasi tersebut untuk hingga muncul tampilan IDLE Shell seperti gambar berikut.



Gambar 9 Tampilan Awal IDLE Shell

Untuk Membuat file python baru, Klik Menu **File >> New File**



Gambar 10 Membuat file baru menggunakan IDLE

Gambar diatas merupakan editor yang akan digunakan untuk Membuat kode program Python. Untuk menjalankan program yang telah ditulis nantinya cukup dengan memilih menu **Run >> Run Module** atau dengan menekan tombol **F5.**

4 Tipe Data, Variabel dan Operator

4.1 Tipe Data

Tipe data adalah karakteristik yang menentukan jenis nilai yang dapat disimpan dan dioperasikan dalam suatu program. Python memiliki beberapa tipe data dasar, antara lain:

Tipe Data	Penulisan	Deskripsi
Integer	int	Mewakili bilangan bulat. Contoh: x = 5
Float	float	Mewakili bilangan pecahan atau desimal. Contoh: y = 3.14
String	str	Mewakili urutan karakter. Contoh: nama = "John"
Boolean	bool	Mewakili nilai kebenaran, yaitu True atau False. Contoh: status = True
List	list	Mewakili urutan nilai yang dapat diubah. Contoh: angka = [1, 2, 3]
Tuple	tuple	Mewakili urutan nilai yang tidak dapat diubah. Contoh: koordinat = (4, 5)
Set	set	Mewakili himpunan nilai unik. Contoh: huruf_unik = {'a', 'b', 'c'}
Dictionary	dict	Mewakili pasangan kunci-nilai. Contoh: data = {'nama': 'John', 'umur': 25}

Tabel 2 Type Data Python

4.2 Variabel

Variabel adalah nama yang diberikan untuk menyimpan nilai atau data dalam program. Dalam Python, variabel tidak perlu dideklarasikan secara eksplisit dan

tipe datanya akan ditentukan secara otomatis saat nilai diberikan. Contoh penggunaan variabel:

```
# Contoh penggunaan variabel
angka = 10
desimal = 3.14
teks = "Halo, dunia!"
benar = True
```

4.3 Operator

Operator adalah simbol atau tanda khusus yang digunakan untuk melakukan operasi pada operand atau nilai. Python mendukung berbagai jenis operator, termasuk:

1. Operator Aritmetika:

   Digunakan untuk melakukan operasi matematika seperti penambahan, pengurangan, perkalian, pembagian, dan sebagainya.

   ```
   a = 5
   b = 2

   tambah = a + b  # Penambahan
   kurang = a - b  # Pengurangan
   kali = a * b    # Perkalian
   bagi = a / b    # Pembagian
   sisa = a % b    # Sisa pembagian
   pangkat = a ** b  # Pangkat
   ```

2. Operator Perbandingan

   Digunakan untuk membandingkan nilai, menghasilkan nilai kebenaran (True atau False).

   ```
   x = 5
   y = 10

   sama_dengan = x == y     # Sama dengan
   ```

```
tidak_sama = x != y       # Tidak sama dengan
lebih_besar = x > y       # Lebih besar
kurang_besar = x < y      # Kurang besar
lebih_besar_sama = x >= y  # Lebih besar atau sama dengan
kurang_besar_sama = x <= y # Kurang besar atau sama dengan
```

3. Operator Logika

   Digunakan untuk melakukan operasi logika seperti **and**, **or**, dan **not**.

```
p = True
q = False

logika_and = p and q  # Logika AND
logika_or = p or q    # Logika OR
logika_not = not p    # Logika NOT
```

4. Operator Penugasan

   Digunakan untuk menetapkan nilai ke variabel.

```
x = 10
y = 5

x += y  # Sama dengan x = x + y
```

5. Operator Keanggotan

   Digunakan untuk memeriksa apakah nilai tertentu ada dalam struktur data seperti list atau string.

```
angka = [1, 2, 3]

cek_angka = 2 in angka  # Apakah 2 ada dalam list?
```

6. Operator Identitas

   Digunakan untuk memeriksa identitas objek, apakah dua objek merujuk ke lokasi memori yang sama.

```
a = [1, 2, 3]
b = [1, 2, 3]
```

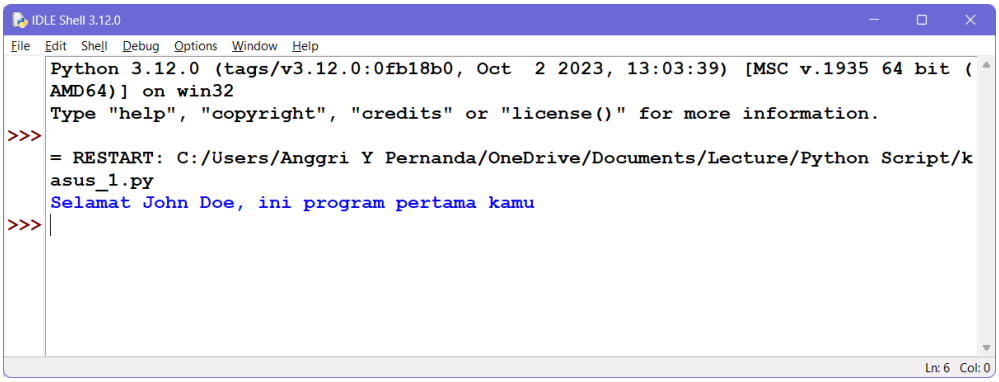
```
identitas = a is b  # Apakah a dan b merujuk ke objek yang sama?
```

4.4 Contoh Program

Contoh 1. Menampilkan teks

```
# Membuat variable nama
nama = "John Doe"
# Menampilkan teks
print("Selamat %s, ini program pertama kamu" %(nama))
```

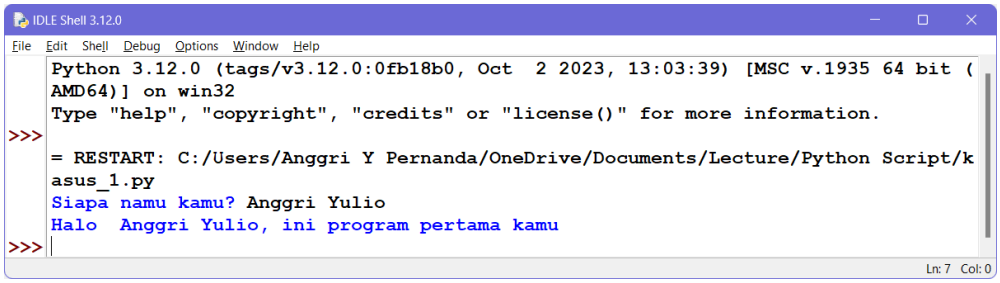
Hasil:



Contoh 2. Menampilkan teks dengan input

```
# Membuat variable nama
nama = input("Siapa namu kamu?")
# Menampilkan teks
print("Halo %s, ini program pertama kamu" %(nama))
```

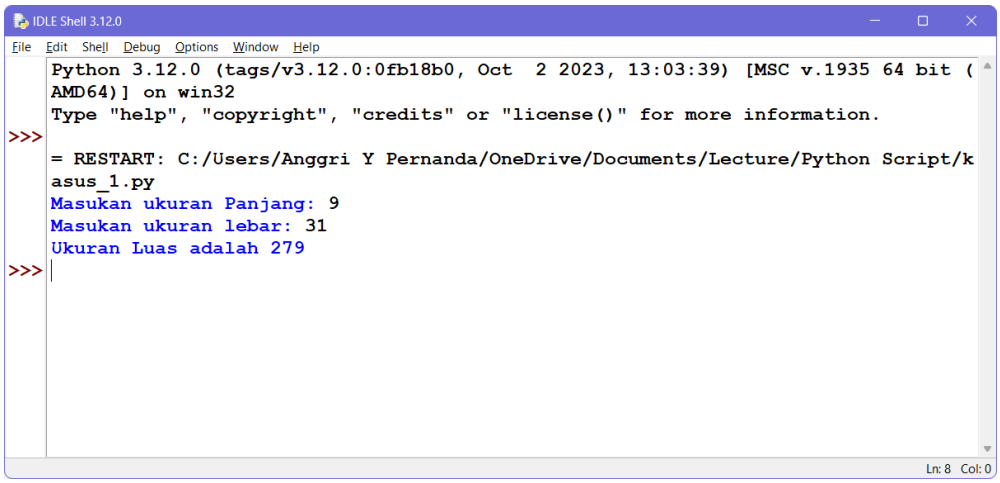
Hasil:



Contoh 3. Membuat program menghitung luas persegi Panjang

```
# Membuat variable Panjang dan lebar
panjang = input("Masukan ukuran Panjang: ")
lebar = input("Masukan ukuran lebar: ")
luas = panjang * lebar
# Menampilkan hasil
print("Ukuran Luas adalah %i" %(luas))
```

Hasil:



5 List, Dictionary, Tuple dan Set

5.1 List

List adalah struktur data yang digunakan untuk menyimpan sekumpulan elemen dalam satu variabel. Elemen-elemen ini dapat memiliki tipe data yang berbeda dan dapat diakses melalui indeks, yang dimulai dari 0. List merupakan salah satu tipe data bawaan Python yang paling fleksibel dan sering digunakan.

Keunggulan List:

- Fleksibilitas: List dapat menyimpan elemen dengan tipe data yang berbeda.
- Kemudahan Akses: Elemen dapat diakses, ditambahkan, atau dihapus dengan mudah menggunakan indeks.
- Metode Bawaan: Python menyediakan banyak metode bawaan untuk manipulasi list.

1. Membuat dan Mengakses List

Membuat List

List dapat dibuat dengan menggunakan tanda kurung siku [].

```
# Membuat list kosong
list_kosong = []

# Membuat list berisi angka
angka = [1, 2, 3, 4, 5]

# Membuat list berisi string
hewan = ['kucing', 'anjing', 'kelinci']

# Membuat list campuran
campuran = [1, 'apel', 3.14, True]
```

2. Mengakses Elemen List

Elemen list dapat diakses menggunakan indeks.

```
# Akses elemen pertama
print(angka[0])  # Output: 1

# Akses elemen kedua
print(hewan[1])  # Output: anjing
```

3. Memodifikasi List

Mengubah Elemen List

Elemen list dapat diubah dengan memberikan nilai baru pada indeks tertentu.

```
angka[2] = 10
print(angka)  # Output: [1, 2, 10, 4, 5]
```

4. Menambahkan Elemen ke List

Elemen baru dapat ditambahkan ke list menggunakan metode append().

```
angka.append(6)
print(angka)  # Output: [1, 2, 10, 4, 5, 6]
```

5. Menghapus Elemen dari List

Elemen dapat dihapus dari list menggunakan metode remove() atau pop().

```
# Menggunakan remove() untuk menghapus elemen pertama yang
ditemukan
angka.remove(10)
print(angka)  # Output: [1, 2, 4, 5, 6]
```

```
# Menggunakan pop() untuk menghapus elemen pada indeks
tertentu
angka.pop(2)
print(angka)  # Output: [1, 2, 5, 6]
```

5.2 Dictionary (Kamus)

Dictionary atau yang sering disebut kamus adalah struktur data yang memungkinkan penyimpanan pasangan nilai-kunci (key-value pairs). Setiap kunci dalam kamus harus unik dan terhubung dengan nilai tertentu. Dalam Python, dictionary didefinisikan dengan menggunakan tanda kurung kurawal {}, dan setiap pasangan nilai-kunci dipisahkan oleh tanda titik dua :.

Keunggulan Dictionary:

- Cepat untuk Pencarian: Dictionary menyediakan pencarian cepat berdasarkan kunci.
- Fleksibel: Dapat menyimpan berbagai jenis data, termasuk list, tuple, atau bahkan dictionary lain.
- Mudah Dibaca: Representasi data yang mirip dengan bahasa manusia memudahkan pemahaman dan penggunaan.

1. Membuat dan Mengakses Dictionary

Membuat Dictionary

Dictionary dapat dibuat dengan menyertakan pasangan nilai-kunci dalam tanda kurung kurawal {}.

```
# Membuat dictionary kosong
kosong = {}
```

```
# Membuat dictionary dengan pasangan nilai-kunci
siswa = {'nama': 'Andi', 'umur': 20, 'kelas': 'XII'}
```

2. Mengakses Elemen Dictionary

Elemen dictionary dapat diakses dengan menyebutkan kuncinya di dalam tanda kurung siku [].

```
# Mengakses nilai dengan kunci 'nama'
print(siswa['nama'])  # Output: Andi

# Mengakses nilai dengan kunci 'umur'
print(siswa['umur'])  # Output: 20
```

3. Memodifikasi dan Menambahkan Elemen Dictionary

Memodifikasi Nilai

Nilai dari kunci tertentu dalam dictionary dapat diubah dengan cara yang serupa dengan mengaksesnya.

```
# Mengubah nilai kunci 'kelas'
siswa['kelas'] = 'XI'
print(siswa)  # Output: {'nama': 'Andi', 'umur': 20, 'kelas':
'XI'}
```

Menambahkan Elemen Baru

Elemen baru dapat ditambahkan ke dictionary dengan menetapkan nilai untuk kunci yang belum ada.

```
# Menambahkan pasangan nilai-kunci baru
siswa['nilai'] = 90
print(siswa)  # Output: {'nama': 'Andi', 'umur': 20, 'kelas':
'XI', 'nilai': 90}
```

4. Menghapus Elemen Dictionary

Elemen dictionary dapat dihapus menggunakan perintah del atau metode pop().

Menghapus Elemen Berdasarkan Kunci

```
# Menghapus kunci 'nilai' dan nilainya
del siswa['nilai']
print(siswa)  # Output: {'nama': 'Andi', 'umur': 20, 'kelas':
'XI'}
```

Menghapus Elemen dan Mengembalikan Nilainya

```
# Menghapus dan mengembalikan nilai kunci 'umur'
umur = siswa.pop('umur')
print(umur) # Output: 20
print(siswa)  # Output: {'nama': 'Andi', 'kelas': 'XI'}
```

5.3 Tuple

Tuple adalah struktur data dalam Python yang mirip dengan list, namun bersifat immutable, artinya, setelah tuple dibuat, elemennya tidak dapat diubah. Tuple didefinisikan dengan menggunakan tanda kurung biasa () dan elemen-elemennya dipisahkan oleh koma.

Keunggulan Tuple:

- Immutable: Elemen-elemen dalam tuple tidak dapat diubah setelah tuple dibuat. Hal ini membuat tuple cocok untuk data yang tidak boleh diubah.
- Efisien: Karena sifatnya yang immutable, tuple lebih efisien dalam penggunaan memori daripada list.
- Digunakan sebagai Key dalam Dictionary: Karena sifatnya yang tidak berubah, tuple dapat digunakan sebagai kunci dalam dictionary.

1. Membuat Tuple

Tuple dapat dibuat dengan menyertakan elemen-elemennya dalam tanda kurung ().

```
# Membuat tuple kosong
kosong = ()

# Membuat tuple dengan beberapa elemen
bulan = ('Januari', 'Februari', 'Maret', 'April', 'Mei')
```

2. Mengakses Elemen Tuple

Elemen tuple dapat diakses dengan menggunakan indeks, mirip dengan cara mengakses elemen dalam list.

```
# Mengakses elemen dengan indeks
print(bulan[0])  # Output: Januari
print(bulan[2])  # Output: Maret
```

3. Mengubah Tuple

Karena sifatnya yang immutable, tuple tidak dapat diubah setelah dibuat. Namun, tuple dapat digabungkan atau diiris untuk membuat tuple baru.

```
# Menggabungkan dua tuple
bulan_lain = ('Juni', 'Juli', 'Agustus')
bulan_gabungan = bulan + bulan_lain
print(bulan_gabungan)  # Output: ('Januari', 'Februari',
'Maret', 'April', 'Mei', 'Juni', 'Juli', 'Agustus')

# Mengiris tuple untuk membuat tuple baru
semester_1 = bulan[:3]
print(semester_1)  # Output: ('Januari', 'Februari', 'Maret')
```

4. Menghapus Tuple

Karena sifatnya yang immutable, tidak ada cara langsung untuk menghapus elemen dari tuple. Namun, tuple itu sendiri dapat dihapus menggunakan perintah del.

```
# Menghapus tuple
del bulan
```

5.4 Set

Set adalah struktur data dalam Python yang digunakan untuk menyimpan kumpulan elemen unik tanpa urutan tertentu. Set didefinisikan dengan menggunakan tanda kurung kurawal {} atau fungsi set().

Keunggulan Set:

- Elemen Unik: Set hanya menyimpan elemen yang unik, sehingga tidak ada duplikasi dalam set.
- Operasi Matematika: Set mendukung operasi matematika seperti gabungan, irisan, dan perbedaan.

1. Membuat dan Mengakses Set

Set dapat dibuat dengan menyertakan elemen-elemennya dalam tanda kurung kurawal {}.

```
# Membuat set kosong
kosong = set()

# Membuat set dengan beberapa elemen
huruf = {'a', 'b', 'c', 'd', 'e'}
```

Jika ingin membuat set dari sebuah list, dapat menggunakan fungsi set().

```
# Membuat set dari list
angka = set([1, 2, 3, 4, 5])
```

2. Mengakses Elemen Set

Karena set tidak memiliki urutan tertentu, tidak ada cara langsung untuk mengakses elemen-elemen dalam set.

3. Menambahkan dan Menghapus Elemen

Menambahkan Elemen

Elemen baru dapat ditambahkan ke dalam set menggunakan metode add().

```
# Menambahkan elemen ke dalam set
huruf.add('f')
print(huruf)  # Output: {'a', 'b', 'c', 'd', 'e', 'f'}
```

Menghapus Elemen

Untuk menghapus elemen dari set, dapat menggunakan metode remove() atau discard().

```
# Menghapus elemen dari set
huruf.remove('e')
print(huruf)  # Output: {'a', 'b', 'c', 'd', 'f'}

huruf.discard('z')  # Jika elemen tidak ada dalam set,
discard() tidak akan menimbulkan error
```

4. Operasi pada Set

Gabungan Set

Operasi gabungan pada set dilakukan dengan menggunakan operator | atau metode union().

```
# Gabungan dua set
huruf_vokal = {'a', 'e', 'i', 'o', 'u'}
gabungan = huruf | huruf_vokal
print(gabungan)  # Output: {'a', 'b', 'c', 'd', 'e', 'f',
'i', 'o', 'u'}
```

Irisan Set

Operasi irisan pada set dilakukan dengan menggunakan operator & atau metode intersection().

```
# Irisan dua set
huruf_bersama = huruf & huruf_vokal
print(huruf_bersama)  # Output: {'a', 'e'}
```

Perbedaan Set

Operasi perbedaan pada set dilakukan dengan menggunakan operator - atau metode difference().

```
# Perbedaan dua set
huruf_tidak_vokal = huruf - huruf_vokal
print(huruf_tidak_vokal)  # Output: {'c', 'd', 'b', 'f'}
```

5.5 Kesimpulan

1. **List**
   - Karakteristik: List adalah struktur data yang paling umum digunakan dalam Python. Ini adalah kumpulan elemen yang dapat diubah (mutable) dan diurutkan.
   - Keunggulan: Fleksibilitas dalam menambah, menghapus, dan mengakses elemen. Cocok untuk menyimpan data yang membutuhkan urutan.
   - Kelemahan: Penelusuran elemen dalam list besar bisa menjadi lambat. Perubahan besar pada list bisa memengaruhi kinerja.
   - Contoh Penggunaan: Penyimpanan data terurut seperti daftar kontak, hasil dari pengolahan data, dan elemen-elemen yang memerlukan pengindeksan.
2. **Dict (Dictionary):**
   - Karakteristik: Dict adalah struktur data yang menggunakan pasangan kunci-nilai (key-value pairs) untuk menyimpan data. Tidak berurutan dan mutable.
   - Keunggulan: Akses cepat ke nilai berdasarkan kunci. Cocok untuk menyimpan data yang memerlukan pencarian cepat berdasarkan kunci.
   - Kelemahan: Tidak ada urutan tertentu dalam penyimpanan data. Membutuhkan lebih banyak memori daripada list.

- Contoh Penggunaan: Pengaturan konfigurasi, menyimpan data terstruktur seperti data pengguna dengan atribut, dan pemetaan antara dua set data.

3. **Tuple:**
    - Karakteristik: Tuple adalah kumpulan elemen yang tidak dapat diubah (immutable) dan berurutan.
    - Keunggulan: Efisien dalam penggunaan memori. Elemen-elemennya tidak dapat diubah, memberikan keamanan data.
    - Kelemahan: Tidak dapat dimodifikasi setelah pembuatan. Tidak mendukung operasi seperti penambahan atau penghapusan elemen.
    - Contoh Penggunaan: Penggunaan sebagai kunci dalam dict (jika kunci perlu tetap tidak berubah), pengembalian nilai dari fungsi yang memerlukan beberapa nilai yang berurutan, dan penggunaan dalam proses yang memerlukan ketidakubahannya.
4. **Set:**
    - Karakteristik: Set adalah kumpulan elemen unik tanpa urutan tertentu.
    - Keunggulan: Memiliki operasi matematika seperti gabungan, irisan, dan perbedaan yang berguna dalam pengolahan data. Hanya menyimpan elemen unik.
    - Kelemahan: Tidak ada urutan tertentu dalam penyimpanan data. Tidak bisa memiliki elemen duplikat.
    - Contoh Penggunaan: Menghilangkan duplikat dari data, melakukan operasi logika antara dua set data, dan penggunaan dalam kasus di mana keunikan elemen adalah kebutuhan utama.

Kesimpulan:

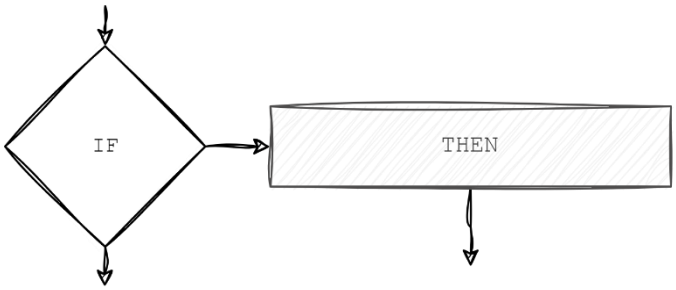
Pemilihan struktur data tergantung pada kebutuhan spesifik aplikasi. Dari uraian diatas dapat kita tarik beberapa kesimpulan.

1. List cocok untuk pengolahan data berurutan yang memerlukan fleksibilitas.
2. Dict ideal untuk pencarian cepat berdasarkan kunci.
3. Tuple digunakan ketika elemen tidak boleh berubah setelah pembuatan.
4. Set berguna untuk operasi matematika dan menghilangkan duplikat.

6 Percabangan [Selection/Decision]

6.1 Konsep Dasar

Konsep percabangan dalam pemrograman adalah kemampuan program untuk membuat keputusan dan mengambil jalur eksekusi yang berbeda berdasarkan kondisi tertentu. Berikut adalah contoh sederhana diagram alir percabangan.



Gambar 11 Struktur Diagram Alir Percabangan

6.2 Struktur Percabangan

Dalam Bahasa pemrograman Python, konsep percabangan dapat diimplementasikan menggunakan pernyataan if, elif (else if), dan else. Berikut adalah penjelasan tentang konsep percabangan dalam Python:

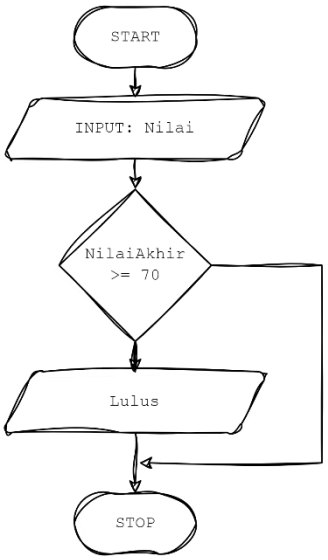
1. Pernyataan 'if'

Pernyataan if digunakan untuk mengevaluasi suatu kondisi dan menjalankan blok kode tertentu jika kondisi tersebut benar (True). Berikut merupakan struktur pernyataan 'if':

```
if kondisi:
# Blok kode yang dijalankan jika kondisi benar
```

Contoh kasus:

Jika seorang mahasiswa mendapat nilai diatas 70 maka ia dianggap lulus.



Gambar 12 Diagram Alir Penilaian Siswa

```
nilai = int(input("Berapa nilai anda?"))
if nilai> 70 :
      print("Selamat anda lulus")
```

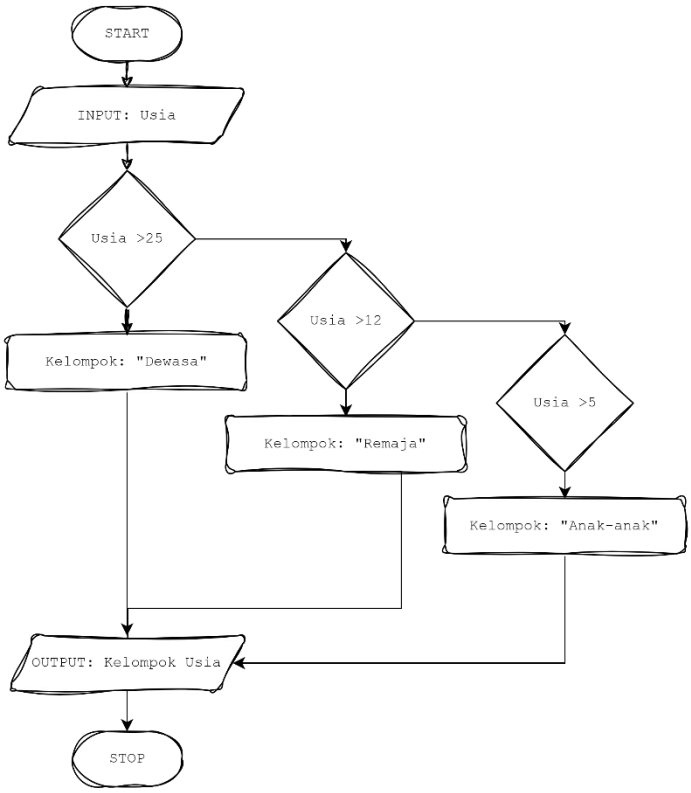
2. Pernyataan "elif"

Pernyataan elif (else if) digunakan untuk menambahkan kondisi tambahan setelah if. Jika kondisi pada if tidak terpenuhi, maka elif akan dievaluasi. Berikut adalah struktur dasar penyataan elif:

```
if kondisi_1:
    # Blok kode yang dijalankan jika kondisi_1 benar
elif kondisi_2:
    # Blok kode yang dijalankan jika kondisi_2 benar
```

penyataan elif dapat ditambahkan sebanyak mungkin, sesuai dengan jumlah kondisi yang diinginkan.

Contoh kasus: Pengelompokan usia



Gambar 13 Diagram Alir Pengelompokan Usia

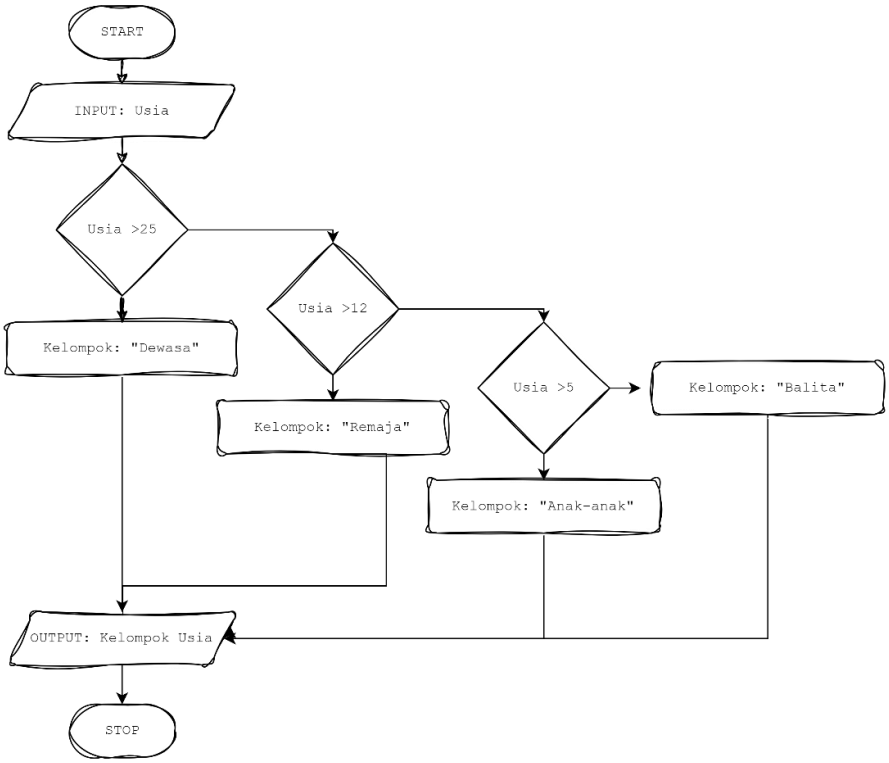
```
usia = int(input("Berapa usia anda?"))
if usia > 25:
      print("Kelompok Usia Dewasa")
elif usia > 12:
      print("Kelompok Usia Remaja")
elif usia > 5:
      print("Kelompok Usia Anak-anak")
```

3. Pernyataan “else”

Pernyataan else digunakan untuk menentukan blok kode yang akan dieksekusi jika semua kondisi sebelumnya (if dan elif) tidak terpenuhi. Struktur dasar pernyataan else adalah sebagai berikut.

```
if kondisi_1:
    # Blok kode yang dijalankan jika kondisi_1 benar
elif kondisi_2:
    # Blok kode yang dijalankan jika kondisi_2 benar
else:
    # Blok kode yang dijalankan jika semua kondisi di atas
tidak terpenuhi
```

Contoh kasus: Pengelompokan usia



Gambar 14 Lanjutan Diagram Alir Pengelompokan Usia

```
usia = int(input(“Berapa usia anda?”))
if usia > 25:
      print(“Kelompok Usia Dewasa”)
elif usia > 12:
      print(“Kelompok Usia Remaja”)
elif usia > 5:
      print(“Kelompok Usia Anak-anak”)
else:
      print(“Kelompok Usia Balita”)
```

Penjelasan:

1. **usia = int(input("Berapa usia anda?"))**: Baris ini meminta pengguna memasukkan usia mereka. Fungsi **input()** digunakan untuk menerima input dari pengguna dalam bentuk string, dan **int()** mengonversi string tersebut menjadi bilangan bulat.

2. **if usia > 25:**: Ini adalah pernyataan kondisional pertama yang mengecek apakah usia lebih dari 25. Jika kondisi ini benar, maka blok kode di bawahnya akan dieksekusi.

3. **print("Kelompok Usia Dewasa")**: Jika usia lebih dari 25, pernyataan **print** ini akan menampilkan "Kelompok Usia Dewasa".

4. **elif usia > 12:**: Jika kondisi pada **if** tidak terpenuhi, pernyataan **elif** (else if) ini akan mengecek apakah usia lebih dari 12. Jika benar, blok kode di bawahnya akan dieksekusi.

5. **print("Kelompok Usia Remaja")**: Jika usia lebih dari 12 tetapi tidak lebih dari 25, pernyataan **print** ini akan menampilkan "Kelompok Usia Remaja".

6. **elif usia > 5:**: Jika kondisi pada **elif** sebelumnya tidak terpenuhi, pernyataan **elif** ini akan mengecek apakah usia lebih dari 5. Jika benar, blok kode di bawahnya akan dieksekusi.

7. **print("Kelompok Usia Anak-anak")**: Jika usia lebih dari 5 tetapi tidak lebih dari 12, pernyataan **print** ini akan menampilkan "Kelompok Usia Anak-anak".

8. **else:**: Ini adalah blok terakhir yang akan dieksekusi jika tidak ada kondisi sebelumnya yang terpenuhi.

9. **print("Kelompok Usia Balita")**: Jika usia tidak memenuhi kondisi pada **if** atau **elif** sebelumnya, pernyataan **print** ini akan menampilkan "Kelompok Usia Balita".

6.3 Contoh Kasus

Anda diminta Membuat program sederhana untuk memberikan nilai akhir kepada mahasiswanya berdasarkan nilai ujian akhir. Mahasiswa akan mendapatkan nilai berdasarkan rentang tertentu. Program harus menentukan kategori nilai sesuai dengan skala berikut:

90 - 100: A

80 - 89: B

70 - 79: C

60 - 69: D

Kurang dari 60: Tidak Lulus

Pertanyaan:

1. Buatlah program Python untuk menentukan kategori nilai mahasiswa berdasarkan nilai yang diberikan.
2. Modifikasi program Anda untuk menambahkan kondisi tambahan. Jika mahasiswa mendapatkan nilai 100, tambahkan pesan ke dalam output yang menyatakan bahwa mahasiswa mendapatkan nilai sempurna.
3. Bagaimana Anda akan menangani situasi di mana input nilai yang diberikan tidak berada dalam rentang 0-100? Tambahkan validasi input pada program Anda untuk menangani situasi ini.
4. Modifikasi program Anda untuk menampilkan pesan khusus jika mahasiswa mendapatkan nilai D atau Tidak Lulus. Jika mahasiswa mendapatkan nilai D, tambahkan pesan bahwa mahasiswa perlu meningkatkan kinerjanya. Jika mahasiswa Tidak Lulus, berikan saran bahwa mahasiswa perlu melakukan perbaikan untuk mencapai kelulusan.

Jawaban:

```
nilai = int(input("Nilai yang diperoleh: "))
if nilai < 0 or nilai > 100:
    print ("Input nilai tidak valid")
elif nilai == 100:
    print ("Nilai: A, Anda mendapatkan nilai sempurna")
elif nilai >= 90:
    print ("Nilai: A")
elif 80 <= nilai < 90:
    print ("Nilai B")
elif 70 <= nilai < 79:
    print ("Nilai C")
elif 60 <= nilai < 69:
    print ("Nilai D, anda perlu belajar lebih giat")
else:
    print("Tidak Lulus, anda perlu melakukan remedial")
```

7 Perulangan / Looping

7.1 Konsep Dasar

Konsep perulangan dalam pemrograman memungkinkan sebuah blok kode dieksekusi berulang kali selama kondisi tertentu terpenuhi. Dengan kata lain, perulangan memungkinkan pengulangan suatu tugas atau serangkaian tugas tanpa harus menulis ulang kode berkali-kali. Dalam Python, terdapat dua jenis perulangan utama: perulangan **for** dan perulangan **while**. Berikut contoh struktur perulangan.

7.2 Perulangan *for*

Perulangan for adalah struktur pengulangan yang digunakan untuk menjalankan serangkaian pernyataan atau blok kode secara berulang, dimana iterasi perulangan dilakukan berdasarkan elemen-elemen dalam suatu urutan (seperti list, tuple, string, atau range). Pengulangan for biasanya digunakan ketika kita mengetahui berapa kali iterasi akan dilakukan. Keunggulan perulangan **for**:

- Kemudahan Penggunaan: Perulangan for sangat mudah digunakan dan lebih sering digunakan jika jumlah iterasi sudah diketahui.
- Menghindari Perulangan Tak Terbatas: Dengan menggunakan urutan atau fungsi range(), kita dapat menghindari perulangan tak terbatas dan memastikan iterasi sesuai dengan panjang urutan.
- Penggunaan Variabel Iterator: Variabel iterator dalam perulangan for menyimpan nilai setiap elemen selama iterasi, memudahkan akses dan pengolahan.

Berikut struktur umum perulangan **for**:

```
for variabel in urutan:
    # Blok kode yang diulang
    pernyataan1
```

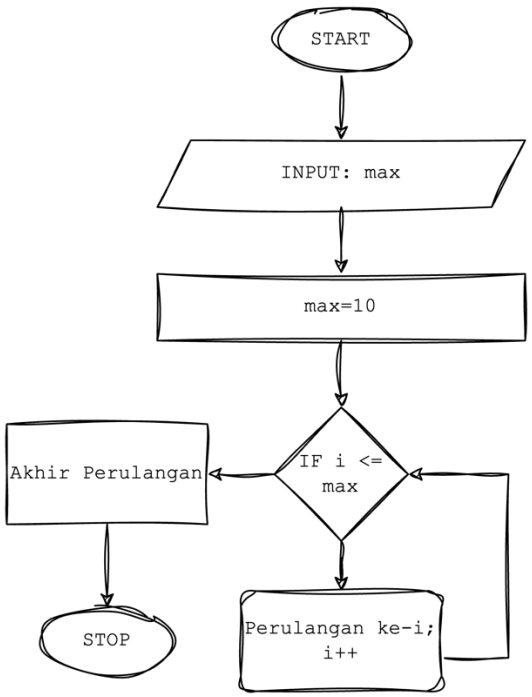
```
    pernyataan2
    # ...
```

Penjelasan:

variabel: Variabel yang digunakan untuk menyimpan nilai setiap elemen dalam urutan selama setiap iterasi.

urutan: Urutan (sequence) dari elemen-elemen yang akan diiterasi.

Berikut contoh flowchart perulangan for Dimana perulangan akan tetap dilanjutkan hingga nilai i sama dengan 10.



Gambar 15 Diagram alir perulangan For

Contoh penggunaan perulangan for:

1. Penggunaan Dasar for dengan List

Menampilkan elemen-elemen dalam sebuah list menggunakan for loop.

```
buah = ['apel', 'pisang', 'jeruk']
for item in buah:
    print(item)
```

2. Penggunaan for dengan range()

Menampilkan angka dari 1 hingga 10 menggunakan fungsi range().

```
for i in range(1, 11):
    print(i)
```

3. Menghindari Perulangan Tak Terbatas

Dengan for loop dan range(), jumlah iterasi sudah ditentukan sehingga menghindari perulangan tak terbatas.

```
for i in range(5):  # Iterasi dilakukan sebanyak 5 kali
    print(i)
```

4. Variabel Iterator

Contoh penggunaan variabel iterator dalam for loop untuk mengakses dan memproses setiap elemen.

```
angka = [10, 20, 30, 40, 50]
total = 0
for num in angka:
    total += num
print(f'Total: {total}')
```

5. Perulangan dengan String

Mengiterasi setiap karakter dalam sebuah string.

```
kata = 'Python'
for huruf in kata:
    print(huruf)
```

6. Perulangan Bersarang

Contoh for loop bersarang untuk menghasilkan kombinasi dari dua list.

```
warna = ['merah', 'hijau', 'biru']
bentuk = ['lingkaran', 'persegi', 'segitiga']
for w in warna:
    for b in bentuk:
        print(f'{w} {b}')
```

7. List Comprehension

Contoh penggunaan list comprehension untuk membuat list baru dengan perulangan for.

```
angka = [1, 2, 3, 4, 5]
kuadrat = [x**2 for x in angka]
print(kuadrat)  # Output: [1, 4, 9, 16, 25]
```

7.3 Perulangan *while*

Perulangan while adalah struktur pengulangan yang digunakan untuk menjalankan serangkaian pernyataan atau blok kode secara berulang selama kondisi yang diberikan bernilai benar (True). Iterasi perulangan akan terus

berlanjut sampai kondisi tersebut bernilai salah (False). Perulangan while berguna ketika jumlah iterasi tidak diketahui sebelumnya dan bergantung pada suatu kondisi dinamis.

Keunggulan Perulangan while:

- Fleksibilitas Kondisi: while loop dapat digunakan untuk perulangan yang bergantung pada kondisi dinamis yang mungkin berubah selama proses eksekusi.
- Pengulangan Tak Terbatas: Dengan while loop, kita bisa membuat perulangan yang berjalan tanpa batas hingga kondisi tertentu terpenuhi.
- Penggunaan Kontrol yang Lebih Bebas: Kita bisa lebih mudah mengontrol kapan dan bagaimana perulangan berhenti dengan menggunakan pernyataan seperti break dan continue.

Contoh Kode:

1. Penggunaan Dasar while

Menggunakan while loop untuk mencetak angka dari 1 hingga 10.

```
i = 1
while i <= 10:
    print(i)
    i += 1
```

2. Menggunakan while dengan Kondisi Dinamis

Menggunakan while loop untuk mencari nilai dalam list sampai ditemukan.

```
angka = [10, 20, 30, 40, 50]
index = 0
target = 30
while index < len(angka):
```

```
    if angka[index] == target:
        print(f'{target} ditemukan pada indeks {index}')
        break
    index += 1
```

3. Menghindari Perulangan Tak Terbatas

Pastikan kondisi dalam while loop berubah agar tidak terjadi perulangan tak terbatas.

```
# Contoh perulangan tak terbatas yang harus dihindari
# i = 1
# while i <= 10:
#     print(i)
#     # i tidak diubah, perulangan akan terus berjalan

# Perulangan dengan kondisi yang berubah
i = 1
while i <= 10:
    print(i)
    i += 1
```

4. Menggunakan break dalam while

Menghentikan perulangan ketika suatu kondisi terpenuhi.

```
i = 1
while True:
    print(i)
    i += 1
    if i > 10:
        break
```

5. Menggunakan continue dalam while

Melewati iterasi saat kondisi tertentu terpenuhi.

```
i = 0
while i < 10:
    i += 1
    if i % 2 == 0:
        continue
    print(i)  # Hanya mencetak angka ganjil
```

6. Perulangan Tak Terbatas dengan while

Perulangan tak terbatas yang berguna untuk server atau program yang harus terus berjalan hingga dihentikan secara eksplisit.

```
while True:
     print("Program berjalan...")
     # Program ini akan terus berjalan hingga dihentikan
```

7. Menggunakan while untuk Validasi Input

Menggunakan while untuk meminta input pengguna hingga mendapatkan input yang valid.

```
while True:
    input_angka = input("Masukkan angka (1-10): ")
    if input_angka.isdigit():
        angka = int(input_angka)
        if 1 <= angka <= 10:
            print(f'Anda memasukkan angka yang valid:
{angka}')
            break
    print("Input tidak valid. Coba lagi.")
```

7.4 Latihan Mandiri

1. Menampilkan Angka dalam Urutan Terbalik

   Tulis sebuah for loop yang mencetak angka dari 10 hingga 1 dalam urutan terbalik.

2. Menghitung Faktor

   Diberikan sebuah angka n, tulis sebuah program yang menggunakan while loop untuk menghitung faktor dari n.

3. Membalik String

   Tulis sebuah program yang meminta pengguna untuk memasukkan sebuah string dan menggunakan for loop untuk membalik string tersebut.

4. Jumlah Deret Angka

   Diberikan sebuah angka n, tulis sebuah program yang menggunakan while loop untuk menghitung jumlah dari deret angka 1 hingga n.

8 Procedure dan Function

8.1 Dasar Teori

Procedure adalah serangkaian instruksi atau tindakan yang dieksekusi dalam urutan tertentu. Mereka biasanya digunakan untuk melakukan tugas tertentu atau operasi tanpa mengembalikan nilai apa pun. Contohnya mungkin mencetak teks ke layar atau memperbarui nilai variabel.

Function adalah blok kode yang dirancang untuk melakukan tugas tertentu dan mengembalikan nilai sebagai hasilnya. Mereka menerima argumen sebagai input, melakukan operasi berdasarkan argumen tersebut, dan kemudian mengembalikan nilai hasil. Dalam banyak bahasa pemrograman, fungsi dapat dipanggil dan digunakan berkali-kali di berbagai bagian dari program.

Perbedaan utama antara keduanya adalah bahwa fungsi mengembalikan nilai, sedangkan prosedur tidak. Dalam beberapa bahasa pemrograman, seperti Python, istilah "fungsi" digunakan untuk merujuk pada kedua jenis, sementara dalam bahasa lain, seperti PL/SQL, mereka dibedakan.

8.2 Procedure

Sebuah procedure adalah sebuah blok kode yang melakukan tindakan tertentu atau serangkaian tindakan dalam suatu program. Mereka tidak mengembalikan nilai, tetapi mungkin mengubah nilai variabel atau melakukan tugas-tugas lain seperti mencetak output. Biasanya, prosedur digunakan untuk mengelompokkan kode yang berfungsi bersama untuk melakukan tugas tertentu agar dapat digunakan kembali tanpa harus menulis ulang kode tersebut.

Berikut adalah contoh sederhana dari sebuah prosedur dalam Python:

```
def greet():
    print("Hello, welcome to our program!")

# Panggil prosedur greet()
greet()
```

Dalam contoh di atas, greet() adalah sebuah prosedur yang mencetak pesan sambutan ke layar. Ini adalah contoh yang sangat sederhana, tetapi prosedur dapat menjadi jauh lebih kompleks, melakukan banyak tindakan, dan menerima argumen untuk mengubah perilakunya.

Dibawah adalah lihat contoh yang lebih kompleks di mana prosedur menerima argumen:

```
def calculate_total_price(quantity, price_per_unit):
    total_price = quantity * price_per_unit
    print("Total price:", total_price)

# Panggil prosedur calculate_total_price() dengan argumen
yang sesuai
calculate_total_price(5, 10)
```

Dalam contoh ini, prosedur calculate_total_price() mengambil dua argumen, quantity dan price_per_unit, dan menghitung total harga dengan mengalikan keduanya. Hasilnya kemudian dicetak ke layar.

Prosedur sangat membantu dalam mengatur kode dan membuatnya lebih mudah dipahami dan dikelola. Mereka memungkinkan kita untuk membagi program menjadi bagian-bagian yang lebih kecil dan lebih fokus, sehingga meningkatkan keterbacaan dan keberlanjutan kode.

8.3 Function

Function adalah blok kode yang dirancang untuk melakukan tugas tertentu. Mereka mirip dengan prosedur, tetapi perbedaannya adalah bahwa fungsi mengembalikan nilai setelah menyelesaikan tugasnya, sementara prosedur tidak.

Fungsi sangat berguna ketika Anda ingin melakukan operasi tertentu yang mungkin perlu Anda lakukan berkali-kali dalam program Anda. Anda dapat menulis fungsi sekali dan memanggilnya di banyak tempat tanpa harus menulis ulang kode yang sama berulang kali.

Berikut adalah contoh sederhana dari sebuah fungsi dalam Python:

```
def add(a, b):
    return a + b

# Panggil fungsi add()
result = add(3, 5)
print("Result:", result)
```

Dalam contoh di atas, add() adalah sebuah fungsi yang mengambil dua argumen a dan b, dan mengembalikan hasil penjumlahan dari kedua argumen tersebut. Nilai yang dikembalikan kemudian disimpan dalam variabel result dan dicetak ke layar.

Fungsi dapat melakukan tugas yang lebih kompleks daripada contoh di atas. Mereka dapat mengambil sejumlah argumen, melakukan operasi matematika, mengakses data, memanggil fungsi lain, dan melakukan hampir semua tugas yang diperlukan dalam program Anda.

Berikut contoh yang lebih kompleks:

```
def calculate_discounted_price(price, discount_rate):
    discounted_price = price - (price * discount_rate / 100)
    return discounted_price

# Panggil fungsi calculate_discounted_price()
original_price = 100
discounted_price = calculate_discounted_price(original_price,
10)
print("Discounted price:", discounted_price)
```

Dalam contoh ini, fungsi calculate_discounted_price() mengambil harga asli dan tingkat diskon, kemudian menghitung harga yang didiskon berdasarkan input tersebut.

Fungsi memungkinkan kita untuk membuat kode yang lebih modular, mudah dipelihara, dan mudah diuji. Dengan menggunakan fungsi, kita dapat memecah program menjadi bagian-bagian yang lebih kecil, masing-masing menangani tugas tertentu, sehingga meningkatkan keterbacaan dan skalabilitas kode.

9 Algoritma Pencarian

Algoritma pencarian adalah metode untuk menemukan suatu nilai tertentu dalam kumpulan data. Tujuannya adalah untuk menentukan apakah elemen tertentu ada dalam kumpulan data, dan jika iya, di mana posisi elemen tersebut berada.

9.1 Linear Search (Pencarian Linear):

Linear Search adalah algoritma sederhana yang digunakan untuk mencari elemen tertentu dalam kumpulan data dengan memeriksa setiap elemen satu per satu dari awal hingga akhir kumpulan data. Ini bekerja dengan cara sebagai berikut:

1. Mulai dari elemen pertama kumpulan data.
2. Periksa setiap elemen satu per satu.
3. Jika elemen yang dicari ditemukan, kembalikan indeksnya.
4. Jika tidak, lanjutkan ke elemen berikutnya.
5. Ulangi proses di atas sampai seluruh kumpulan data diperiksa.

Keuntungan dari Linear Search adalah itu sederhana dan mudah dipahami. Namun, kelemahannya adalah kompleksitasnya O(n), di mana n adalah jumlah elemen dalam kumpulan data. Ini berarti waktu pencarian meningkat secara linier seiring dengan peningkatan jumlah elemen dalam kumpulan data.

Contoh penerapan pada python:

```
def linear_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i  # Mengembalikan indeks jika elemen
ditemukan
    return -1  # Mengembalikan -1 jika elemen tidak ditemukan

# Contoh penggunaan
```

```
arr = [10, 20, 30, 40, 50]
target = 30
result = linear_search(arr, target)
if result != -1:
    print(f"Elemen {target} ditemukan pada indeks {result}.")
else:
    print(f"Elemen {target} tidak ditemukan dalam array.")
```

Penjelasan:

- def linear_search(arr, target):: Ini adalah deklarasi dari fungsi linear_search yang menerima dua parameter: arr, yang merupakan array yang akan dicari, dan target, yang merupakan elemen yang ingin dicari dalam array.
- for i in range(len(arr)):: Loop for digunakan untuk mengiterasi melalui setiap elemen dalam array arr.
- if arr[i] == target:: Ini adalah kondisi yang memeriksa apakah elemen saat ini diindeks i sama dengan target.
- return i: Jika elemen yang dicari ditemukan, fungsi akan mengembalikan indeks di mana elemen tersebut ditemukan.
- return -1: Jika elemen tidak ditemukan setelah loop selesai dieksekusi, fungsi akan mengembalikan -1.

9.2 Binary Search (Pencarian Biner)

Binary Search adalah algoritma pencarian yang hanya dapat digunakan pada kumpulan data yang telah diurutkan secara terurut. Ini bekerja dengan cara sebagai berikut:

1. Bandingkan elemen yang dicari dengan elemen tengah dari kumpulan data.
2. Jika elemen yang dicari sama dengan elemen tengah, kembalikan indeksnya.

3. Jika elemen yang dicari lebih kecil dari elemen tengah, cari di setengah kiri kumpulan data.
4. Jika elemen yang dicari lebih besar dari elemen tengah, cari di setengah kanan kumpulan data.
5. Ulangi proses di atas pada setengah kumpulan data yang relevan hingga elemen yang dicari ditemukan atau tidak ada lagi elemen yang tersisa untuk diperiksa.

Keuntungan dari Binary Search adalah kompleksitasnya O(log n), di mana n adalah jumlah elemen dalam kumpulan data yang diurutkan. Ini membuatnya sangat efisien untuk kumpulan data yang besar. Namun, kelemahannya adalah kumpulan data harus diurutkan terlebih dahulu, dan algoritma ini tidak efektif untuk kumpulan data yang tidak terurut.

Contoh penerapan denghan Python:

```
def binary_search(arr, target):
    low = 0
    high = len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid  # Mengembalikan indeks jika elemen
ditemukan
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1  # Mengembalikan -1 jika elemen tidak ditemukan

# Pastikan array sudah diurutkan
arr = [10, 20, 30, 40, 50]
target = 30
```

```
result = binary_search(arr, target)
if result != -1:
    print(f"Elemen {target} ditemukan pada indeks {result}.")
else:
    print(f"Elemen {target} tidak ditemukan dalam array.")
```

Penjelasan:

- def binary_search(arr, target):: Ini adalah deklarasi dari fungsi binary_search yang menerima dua parameter: arr, yang merupakan array yang akan dicari (harus diurutkan terlebih dahulu), dan target, yang merupakan elemen yang ingin dicari dalam array.
- low = 0 dan high = len(arr) - 1: Inisialisasi dua pointer, low dan high, yang akan menunjukkan rentang pencarian dalam array.
- while low <= high:: Loop while akan terus berjalan selama low tidak melewati high.
- mid = (low + high) // 2: Menghitung indeks tengah dari rentang pencarian.
- if arr[mid] == target:: Ini adalah kondisi yang memeriksa apakah elemen di indeks tengah mid sama dengan target.
- elif arr[mid] < target: dan else:: Kondisi ini memperbarui nilai low atau high berdasarkan apakah target lebih besar atau lebih kecil dari elemen di indeks tengah.
- return mid: Jika elemen yang dicari ditemukan, fungsi akan mengembalikan indeks di mana elemen tersebut ditemukan.
- return -1: Jika elemen tidak ditemukan setelah loop selesai dieksekusi, fungsi akan mengembalikan -1.

Pemilihan antara Linear Search dan Binary Search tergantung pada sifat data yang akan dicari, apakah sudah diurutkan atau tidak, serta kebutuhan akan kinerja dan efisiensi.

Daftar Pustaka

Abdel-Hafeez, S., & Gordon-Ross, A. (2017). An Efficient O ($ N $) Comparison-Free Sorting Algorithm. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 25(6), 1930-1942.

Boyer, R. S., & Moore, J. S. (1977). A fast string searching algorithm. Communications of the ACM, 20(10), 762-772.

Hopcroft, J. E., Ullman, J. D., & Aho, A. V. (1983). Data structures and algorithms (Vol. 175). Boston, MA, USA:: Addison-wesley.

Lutz, M. (2013). Learning python: Powerful object-oriented programming. " O'Reilly Media, Inc.".

PENERBIT
CV. HAQI PARADISE
MEDIATAMA

ISBN 978-623-5299-29-7
9 786235 299297