MODUL AJAR

ALGORITMA DAN PEMROGRAMAN

Oleh :

GEMA KHARISMAJATI, S.Kom.,M.Kom.

PROGRAM STUDI SISTEM INFORMASI

FAKULTAS SAINS DAN TEKNOLOGI

UNIVERSITAS PGRI YOGYAKARTA

2024

HALAMAN PENGESAHAN

1. Judul Modul Ajar : Algoritma & Pemrograman
2. Pelaksana
   a. Nama Lengkap : Gema Kharismajati, S.Kom.,M.Kom.
   b. Jenis Kelamin : Laki-laki
   c. Pangkat/Golongan : Penata Muda Tk I/IIIb
   d. NIP/NIS : 199601142024011006
   e. Program Studi/Fakultas : Sistem Informasi/Fakultas Sains & Teknologi
   f. No HP/Email : 082226359766/gemakharismajati@upy.ac.id

Yogyakarta, 01 Agustus 2024

Mengetahui,	Pelaksana/Penulis
Kaprodi Sistem Informasi	
Ferra Arik Tridalestari, ST.,MT.	Gema Kharismajati, S.Kom.,M.Kom.
NIS. 19860115 202110 2 005	NIS. 19960114 202401 1 006

Mengetahui,

Ketua Lembaga Pengembang Pendidikan

Selly Rahmawati, M.Pd.

NIS. 19870723 201302 2 002

KATA PENGANTAR

Modul ajar Algoritma dan Pemrograman ini dirancang untuk mendukung proses kegiatan belajar mengajar, sehingga memudahkan para pembaca dalam memahami konsep-konsep algoritma dan pemrograman, khususnya bagi mahasiswa Program Studi Sistem Informasi di Universitas PGRI Yogyakarta. Modul ajar ini akan memberikan informasi secara lengkap mengenai struktur data dasar, tipe data, operator, kontrol alur, dan teknik dasar penyusunan algoritma. Dengan adanya modul ini, diharapkan mahasiswa dapat mengaplikasikan algoritma dengan membuat program sederhana menggunakan bahasa pemrograman yang dipelajari.

Penulis memilih bahasa pemrograman Python dalam modul ini karena kemudahan belajarnya, sifat cross-platform, dukungan multi-device, dan terutama, kemudahannya dalam implementasi algoritma. Semoga modul ajar Algoritma dan Pemrograman ini dapat membantu pembaca dalam menyelesaikan masalah dengan metode pemecahan berdasarkan konsep algoritma yang telah dipelajari.

Penulis menyadari bahwa dalam penyusunan modul ini masih terdapat kekurangan dan jauh dari kesempurnaan. Oleh karena itu, kritik, saran, serta masukan yang membangun sangat diharapkan demi perbaikan dan kesempurnaan modul ajar ini di masa mendatang. Mengingat tidak ada sesuatu yang sempurna tanpa adanya masukan yang konstruktif. Akhir kata penulis ucapkan terimakasih, harapan penulis semoga Modul Ajar Algoritma dan Pemrograman ini dapat bermanfaat bagi semua pihak terutama mahasiswa Prodi Sistem Informasi Universitas PGRI Yogyakarta.

Yogyakarta, Agustus 2024

Penulis

Dafttar Isi

PENDAHULUAN

1. Deskripsi Mata Kuliah

A. Deskripsi Umum

Mata kuliah Algoritma dan Pemrograman merupakan salah satu mata kuliah dasar yang wajib diambil oleh mahasiswa program studi Sistem Informasi di Fakultas Sains dan Teknologi, Universitas PGRI Yogyakarta. Mata kuliah ini dirancang untuk memberikan pemahaman fundamental tentang konsep algoritma dan dasar-dasar pemrograman kepada mahasiswa. Melalui mata kuliah ini, mahasiswa akan mempelajari berbagai teknik dalam merancang, menganalisis, dan mengimplementasikan algoritma menggunakan bahasa pemrograman, khususnya Python.

Pada awal perkuliahan, mahasiswa akan diperkenalkan dengan konsep dasar algoritma dan pemrograman. Pemahaman ini mencakup struktur data dasar seperti array dan linked list, serta kontrol aliran program termasuk percabangan dan perulangan. Mahasiswa akan belajar menyusun algoritma untuk menyelesaikan berbagai masalah komputasi dan memahami bagaimana algoritma tersebut dapat diimplementasikan dalam bentuk program yang efisien dan efektif.

Selama perkuliahan, berbagai teknik pemrograman akan diajarkan, mulai dari penggunaan fungsi dan prosedur hingga konsep rekursi. Mahasiswa juga akan mempelajari teknik sorting dan searching yang penting dalam pengolahan data. Implementasi teknik-teknik ini dilakukan menggunakan bahasa pemrograman Python, yang terkenal dengan sintaksisnya yang sederhana dan kemampuannya yang kuat dalam pemrosesan data.

Selain itu, mata kuliah ini juga mencakup pengantar pemrograman berorientasi objek (OOP), di mana mahasiswa akan mempelajari konsep dasar OOP seperti kelas, objek, dan metode. Pemahaman ini penting untuk pengembangan aplikasi yang lebih kompleks dan terstruktur. Implementasi konsep OOP dalam Python akan membantu mahasiswa memahami bagaimana cara kerja pemrograman berorientasi objek dan aplikasinya dalam pengembangan perangkat lunak.

Penekanan juga diberikan pada kemampuan mahasiswa untuk menguji dan melakukan debugging pada program yang mereka buat. Ini termasuk teknik untuk mendeteksi dan memperbaiki kesalahan dalam kode, serta memastikan bahwa program berfungsi sesuai dengan yang diharapkan. Kemampuan ini sangat penting untuk menghasilkan perangkat lunak yang andal dan bebas dari bug.

Mahasiswa juga akan diajarkan tentang pentingnya dokumentasi kode program. Dokumentasi yang baik akan memudahkan pengembangan dan pemeliharaan kode di masa depan serta membantu tim pengembang lain untuk memahami dan menggunakan kode yang telah ditulis. Mata kuliah ini mengajarkan teknik-teknik dokumentasi yang efektif untuk memastikan bahwa setiap bagian kode terdokumentasi dengan jelas dan lengkap.

Di akhir perkuliahan, mahasiswa akan diuji melalui proyek akhir yang mengharuskan mereka untuk menerapkan semua konsep yang telah dipelajari selama semester. Proyek ini akan menguji kemampuan mereka dalam merancang, mengimplementasikan, menguji, dan mendokumentasikan solusi pemrograman yang kompleks. Presentasi proyek akhir juga akan menjadi bagian dari penilaian, di mana mahasiswa harus mampu menjelaskan solusi yang mereka kembangkan dan menjawab pertanyaan dari dosen dan rekan-rekan mereka.

Secara keseluruhan, mata kuliah Algoritma dan Pemrograman bertujuan untuk membekali mahasiswa dengan pemahaman yang kuat tentang dasar-dasar pemrograman dan kemampuan untuk mengembangkan algoritma yang efektif dan efisien. Melalui pendekatan praktikum dan teori yang seimbang, mahasiswa diharapkan mampu menerapkan pengetahuan yang mereka peroleh dalam berbagai konteks pemrograman dan siap menghadapi tantangan dalam dunia teknologi informasi yang terus berkembang.

B. Deskripsi Khusus

Mata kuliah ini mencakup beberapa topik utama yang diajarkan selama satu semester, yang meliputi:

a Pengenalan Algoritma dan Pemrograman: Mahasiswa akan belajar tentang definisi algoritma, struktur dasar algoritma, dan pengenalan dasar pemrograman menggunakan Python.

b Struktur Data Dasar: Pembahasan tentang array dan linked list, termasuk cara mengakses, menyimpan, dan memanipulasi data dalam struktur ini.

c Kontrol Aliran Program: Teknik-teknik percabangan (if-else) dan perulangan (for, while) untuk mengontrol aliran eksekusi program.

d Fungsi dan Prosedur: Penggunaan fungsi dan prosedur untuk modularisasi program, termasuk parameter, return values, dan scope.

e Rekursi: Konsep dan aplikasi rekursi dalam pemrograman, serta bagaimana cara kerjanya dibandingkan dengan iterasi.

f Sorting dan Searching: Teknik-teknik pengurutan dan pencarian data, termasuk bubble sort, selection sort, insertion sort, dan binary search.

g Pengantar Pemrograman Berorientasi Objek (OOP): Konsep dasar OOP seperti kelas, objek, pewarisan, dan polimorfisme.

h Debugging dan Testing: Teknik-teknik untuk menguji dan memperbaiki kesalahan dalam program, serta pentingnya testing dalam pengembangan perangkat lunak.

i Proyek Akhir: Implementasi proyek akhir yang mencakup seluruh konsep yang telah dipelajari, dengan fokus pada pengembangan solusi pemrograman yang kompleks dan terstruktur.

Mata kuliah ini memberikan landasan yang kuat bagi mahasiswa untuk melanjutkan ke mata kuliah pemrograman yang lebih lanjut dan siap menghadapi tantangan di bidang teknologi informasi. Dengan pemahaman yang mendalam tentang algoritma dan dasar-dasar pemrograman, mahasiswa diharapkan mampu

menjadi profesional yang kompeten dan inovatif dalam mengembangkan solusi teknologi.

2. Prasyarat Mata Kuliah

Untuk dapat mengikuti mata kuliah Algoritma dan Pemrograman, mahasiswa diharapkan telah memiliki pengetahuan dasar dalam penggunaan komputer dan memahami konsep dasar matematika, termasuk aljabar dan logika dasar. Prasyarat ini akan membantu mahasiswa dalam memahami materi yang diajarkan dan mengikuti praktik pemrograman dengan lebih mudah.

3. Rencana Pembelajaran

Pertemuan	Topik	Sub Topik
1-2	Pengantar Algoritma dan Pemrograman	Pengenalan Python
3-4	Tipe Data dan Variabel	Operator, dan Ekspresi
5-6	Struktur Kontrol	Percabangan dan Pengulangan
7-8	Fungsi dan Modul	
9-10	Struktur Data Dasar	List, Tuple, Set, dan Dictionary
11-12	Pemrograman Berbasis Objek	
13-14	Pengujian dan Debugging	
15	Review dan Persiapan Ujian	
16	Ujian Akhir Semester	

4. Petunjuk Penggunaan Modul Ajar

4.1. Untuk Mahasiswa

a. Baca Secara Berurutan: Ikuti urutan bab yang telah disusun untuk memahami konsep secara bertahap.
b. Praktik Mandiri: Lakukan latihan pemrograman yang diberikan untuk memperdalam pemahaman.
c. Diskusi dan Tanya Jawab: Manfaatkan sesi diskusi di kelas dan forum online untuk mengajukan pertanyaan.
d. Rujuk Referensi Tambahan: Gunakan referensi tambahan yang disarankan untuk memperluas wawasan.
e. Gunakan tes formatif dan gunakan kunci jawaban untuk mengecek pemahaman

4.2. Untuk Dosen

a. Panduan Mengajar: Gunakan buku ajar sebagai panduan utama dalam menyampaikan materi di kelas.
b. Konteks Praktis: Sertakan contoh kasus praktis yang relevan untuk memperkuat konsep yang diajarkan.
c. Evaluasi dan Umpan Balik: Berikan tugas dan kuis secara berkala untuk mengukur pemahaman mahasiswa dan berikan umpan balik konstruktif.

d. Diskusi Kelompok: Dorong mahasiswa untuk bekerja dalam kelompok untuk menyelesaikan tugas yang kompleks.

5. Capaian Pembelajaran

CPMK	Deskripsi
CPMK1	Mahasiswa mampu menjelaskan konsep dasar algoritma dan pemrograman termasuk struktur data dasar, tipe data, operator, dan kontrol alur.
CPMK2	Mahasiswa mampu mengidentifikasi, merumuskan, dan menyelesaikan masalah menggunakan pendekatan algoritmis dengan efisien dan efektif.
CPMK3	Mahasiswa mampu mengimplementasikan algoritma dalam bahasa pemrograman Python.
CPMK4	Mahasiswa mampu menguji dan melakukan debugging pada program untuk memastikan program berfungsi sesuai dengan yang diharapkan.
CPMK5	Mahasiswa mampu mendokumentasikan kode program dengan baik sehingga dapat dipahami dan digunakan oleh pihak lain.
CPMK6	Mahasiswa mampu bekerja dalam tim untuk mengembangkan solusi pemrograman yang kompleks serta mampu membagi tugas dan tanggung jawab secara efektif.

6. Cek Kemampuan Awal Mahasiswa

Untuk mengevaluasi kemampuan awal mahasiswa, berikut adalah beberapa pertanyaan yang dapat digunakan:

a. Apa yang Anda ketahui tentang algoritma? Berikan contoh sederhana.
b. Jelaskan perbedaan antara variabel dan tipe data.
c. Apa yang Anda ketahui tentang struktur kontrol dalam pemrograman? Sebutkan beberapa contohnya.
d. Apakah Anda pernah menulis program komputer sebelumnya? Jika ya, dalam bahasa pemrograman apa?
e. Bagaimana Anda mendefinisikan masalah dan menemukan solusi dalam kehidupan sehari-hari?

Pertanyaan-pertanyaan diatas, akan membantu mengidentifikasi pemahaman awal mahasiswa tentang konsep dasar yang akan dibahas dalam mata kuliah Algoritma dan Pemrograman. Tujuan utamanya sebagai berikut :

a. Mengukur pemahaman awal tentang konsep dasar algoritma.
b. Menilai pemahaman tentang dasar-dasar pemrograman.
c. Mengidentifikasi pengetahuan tentang kontrol alur dalam pemrograman.
d. Mengetahui pengalaman sebelumnya dalam pemrograman dan bahasa pemrograman yang digunakan.
e. Mengevaluasi kemampuan berpikir logis dan sistematis dalam pemecahan masalah.

Modul 1: Pengenalan Algoritma dan Pemrograman

Modul ini dirancang sebagai pengantar untuk mempelajari algoritma dan pemrograman, dengan fokus pada penggunaan bahasa Python. Di dalamnya, mahasiswa akan dibimbing untuk memahami konsep dasar algoritma, definisi pemrograman, dan bagaimana memulai pengembangan program dengan Python. Modul ini memiliki relevansi yang signifikan sebagai landasan untuk mendalami konsep-konsep pemrograman yang lebih kompleks dan pengembangan perangkat lunak yang lebih canggih. Dengan mengikuti modul ini, mahasiswa diharapkan mampu memahami dasar-dasar algoritma dan pemrograman, mengenal fitur utama bahasa Python serta lingkungan pengembangannya, dan akhirnya mampu menulis serta menjalankan program-program sederhana dalam Python. Capaian pembelajaran ini merupakan langkah awal yang esensial dalam membangun keterampilan pemrograman yang lebih lanjut.

A. Penyajian materi

1. Pengertian Algoritma

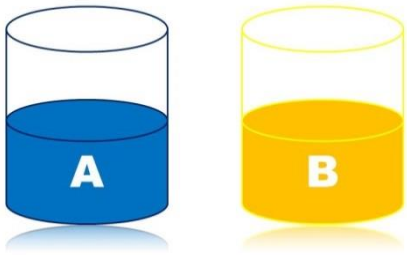
Algoritma adalah urutan langkah-langkah logis dan sistematis yang dirancang untuk menyelesaikan suatu masalah atau mencapai tujuan tertentu. Algoritma digunakan untuk memberikan solusi yang terstruktur terhadap permasalahan yang ada. Dalam konteks pemrograman, algoritma merupakan langkah awal yang harus ditulis sebelum memulai penulisan kode program. Biasanya, masalah yang dapat dipecahkan dengan pemrograman komputer berkaitan dengan perhitungan matematis.

Algoritma adalah inti dari ilmu komputer dan konsep ini juga dapat diterapkan dalam berbagai bidang lainnya. Meskipun sering dikaitkan dengan ilmu komputer, algoritma juga hadir dalam kehidupan sehari-hari. Sebagai contoh, resep pembuatan kue atau masakan adalah bentuk algoritma dalam dunia kuliner. Setiap resep memiliki serangkaian langkah yang harus diikuti untuk menghasilkan masakan yang diinginkan. Jika langkah-langkah dalam resep tersebut tidak disusun secara logis, hasil masakan mungkin tidak akan memuaskan.

Ketika mengikuti resep, langkah pertama adalah membaca instruksi dengan seksama dan melaksanakan setiap langkah sesuai dengan urutan yang ditentukan. Proses pembuatan masakan ini dapat melibatkan berbagai alat atau pemroses, seperti manusia, robot, komputer, atau perangkat elektronik lainnya. Pemroses ini bertugas untuk menjalankan algoritma secara teratur dan sesuai dengan tujuan yang diinginkan.

Untuk lebih memahami konsep algoritma, pertimbangkan contoh sederhana berikut: pertukaran isi antara dua gelas, Gelas A dan Gelas B. Gelas A berisi larutan berwarna biru, sementara Gelas B berisi larutan berwarna kuning (lihat Gambar 1.1). Tujuan dari algoritma ini adalah untuk menukar isi kedua gelas sehingga Gelas A yang awalnya berisi larutan biru kini berisi larutan kuning dari Gelas B.

Ilustrasi permasalahan dapat dilihat dibawah ini :



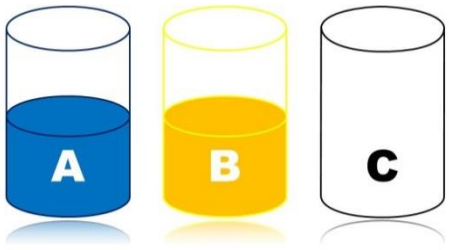
Gambar 1.1 Penukaran isi gelas isinya Gelas A dan Gelas B

Untuk menyelesaikan masalah pertukaran isi antara dua gelas dengan benar, kita memerlukan gelas tambahan yang disebut Gelas C sebagai tempat penyimpanan sementara. Berikut adalah langkah-langkah algoritma yang perlu diikuti:

a. Persiapan Gelas Cadangan: Siapkan Gelas C yang kosong.
b. Pindahkan Larutan dari Gelas A: Tuangkan larutan dari Gelas A ke Gelas C.
c. Pindahkan Larutan dari Gelas B: Tuangkan larutan dari Gelas B ke Gelas A.
d. Pindahkan Larutan dari Gelas C: Tuangkan larutan dari Gelas C ke Gelas B.

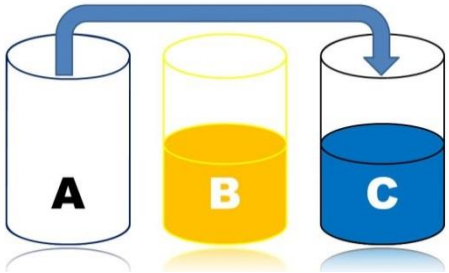
Proses pertukaran ini dapat diilustrasikan dengan urutan gambar berikut:

1. Kondisi Awal: Menunjukkan keadaan sebelum pertukaran dimulai, dengan tambahan Gelas C.



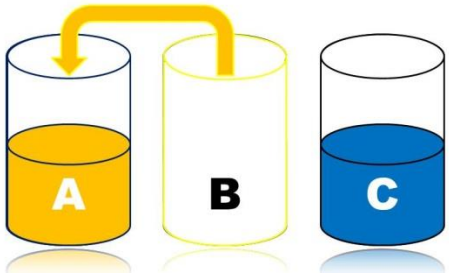
Gambar 1.2 Penukaran isi gelas A,B dan C

2. Langkah Pertama: Larutan dari Gelas A dituang ke Gelas C.



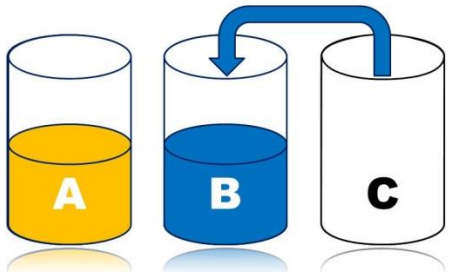
Gambar 1.3 Penukaran isi gelas isinya Gelas A ke Gelas C

3. Langkah Kedua: Larutan dari Gelas B dituang ke Gelas A.



Gambar 1.4 Penukaran isi gelas isinya Gelas B ke Gelas A

4. Langkah Ketiga: Larutan dari Gelas C dituang ke Gelas B.



Gambar 1.5 Penukaran isi gelas isinya Gelas C ke Gelas B

Contoh algoritma ini menunjukkan bahwa pertukaran isi antara dua gelas dapat dilakukan dengan cara yang cukup sederhana. Meskipun proses ini tampak mudah bagi manusia, komputer memerlukan urutan langkah yang sistematis dan logis untuk menyelesaikannya. Inilah inti dari konsep algoritma: serangkaian langkah yang teratur dan logis untuk menyelesaikan suatu masalah dengan cara yang efisien dan benar.

2. Contoh Contoh Algoritma

1. **Resep Masakan:** Algoritma dalam resep masakan menggambarkan langkah-langkah terstruktur untuk menghasilkan makanan tertentu. Proses ini sering kali melibatkan beberapa tahapan berikut:
    - Persiapan Bahan: Mengukur dan menyiapkan bahan-bahan yang diperlukan seperti tepung, gula, telur, dan mentega.
    - Pencampuran Bahan: Mencampur bahan-bahan sesuai urutan yang ditentukan dalam resep. Misalnya, mencampur bahan kering terlebih dahulu sebelum menambahkan bahan basah.
    - Memanaskan Oven: Mengatur suhu oven sesuai dengan instruksi resep. Misalnya, memanaskan oven pada suhu 180°C.

- Memanggang: Menuangkan adonan ke dalam loyang dan memanggangnya dalam oven selama waktu yang ditentukan, misalnya 30 menit.
- Pendinginan dan Penyajian: Setelah matang, mendinginkan kue sebelum menyajikannya.

Setiap langkah dalam resep harus dilakukan dengan urutan yang benar untuk memastikan hasil akhir yang sesuai. Misalnya, jika bahan tidak dicampur dengan benar atau oven tidak dipanaskan terlebih dahulu, kue mungkin tidak berhasil.

2. **Proses Pencarian Data:** Algoritma pencarian linear yang memeriksa setiap elemen dalam daftar satu per satu hingga menemukan elemen yang dicari. Algoritma pencarian data digunakan untuk menemukan elemen tertentu dalam struktur data seperti daftar atau array. Dua contoh metode pencarian adalah:
   a. **Pencarian Linear :** Algoritma ini memeriksa setiap elemen dalam daftar satu per satu dari awal hingga akhir untuk menemukan elemen yang dicari. Misalnya, jika kita mencari angka 7 dalam daftar `[1, 3, 5, 7, 9]`, algoritma ini akan memeriksa setiap elemen hingga menemukan angka 7.

      Langkah-langkah:

      1. Mulai dari elemen pertama.
      2. Bandingkan elemen saat ini dengan elemen yang dicari.
      3. Jika cocok, kembalikan indeks elemen tersebut.
      4. Jika tidak, lanjutkan ke elemen berikutnya.
      5. Jika elemen tidak ditemukan setelah memeriksa seluruh daftar, kembalikan nilai yang menunjukkan bahwa elemen tidak ada.

   b. **Pencarian Biner :** Algoritma ini digunakan pada daftar yang sudah diurutkan dan memanfaatkan teknik pembagian dan penaklukan untuk mengurangi jumlah perbandingan. Ini membagi daftar menjadi dua bagian, membandingkan elemen tengah dengan elemen yang dicari, dan kemudian mencari hanya di setengah bagian yang relevan.

      Langkah-langkah :

      1. Tentukan indeks awal dan akhir dari daftar.
      2. Temukan elemen tengah.

3. Bandingkan elemen tengah dengan elemen yang dicari.
4. Jika cocok, kembalikan indeks elemen tersebut.
5. Jika elemen yang dicari lebih kecil dari elemen tengah, ulangi pencarian pada setengah bagian kiri.
6. Jika elemen yang dicari lebih besar, ulangi pencarian pada setengah bagian kanan.
7. Ulangi langkah-langkah ini hingga elemen ditemukan atau rentang pencarian menjadi kosong.

3. **Algoritma Sortir:** Algoritma pengurutan seperti Bubble Sort atau Merge Sort yang menyusun elemen dalam urutan tertentu. Algoritma sortir digunakan untuk menyusun elemen dalam urutan tertentu, baik dari yang terkecil ke yang terbesar atau sebaliknya. Dua algoritma sortir yang umum adalah:

   a. **Bubble Sort**: Algoritma ini bekerja dengan membandingkan pasangan elemen yang berdekatan dan menukarnya jika urutannya salah. Proses ini diulang hingga tidak ada elemen yang perlu ditukar lagi.

      Langkah-langkah:

      1. Mulai dari elemen pertama dan bandingkan dengan elemen berikutnya.
      2. Jika elemen pertama lebih besar, tukar dengan elemen kedua.
      3. Lanjutkan hingga akhir daftar.
      4. Ulangi langkah-langkah ini untuk seluruh daftar hingga tidak ada lagi penukaran yang diperlukan.

   b. **Merge Sort**: Algoritma ini menggunakan teknik pembagian dan penaklukan dengan membagi daftar menjadi dua bagian, mengurutkan setiap bagian secara terpisah, dan kemudian menggabungkannya kembali dalam urutan yang benar.

      Langkah-langkah:

      1. Bagi daftar menjadi dua bagian hingga setiap bagian hanya memiliki satu elemen.
      2. Gabungkan dua bagian yang sudah diurutkan menjadi satu daftar yang lebih besar.
      3. Ulangi proses penggabungan hingga seluruh daftar terurut.

3. Pengertian Pemrograman

Pemrograman adalah proses menulis, menguji, dan memelihara kode-kode komputer yang memungkinkan perangkat keras dan perangkat lunak untuk melakukan tugas tertentu. Kode ini ditulis dalam bahasa pemrograman, yang merupakan serangkaian instruksi yang dapat dipahami dan dijalankan oleh komputer. Pemrograman merupakan inti dari pengembangan perangkat lunak dan aplikasi, serta merupakan alat utama dalam menciptakan solusi digital yang menyelesaikan masalah atau memenuhi kebutuhan spesifik.

a. Definisi Pemrograman

- Proses Penulisan Kode: Pemrograman melibatkan penulisan kode dalam bahasa pemrograman untuk memberi instruksi kepada komputer mengenai apa yang harus dilakukan. Ini seperti memberikan arahan yang terperinci kepada komputer untuk menjalankan tugas tertentu.

  Contoh: Menulis kode untuk membuat aplikasi yang menghitung suhu tubuh dari data input yang diberikan oleh sensor.

- Pengembangan Perangkat Lunak: Melalui pemrograman, kita dapat membuat perangkat lunak yang mencakup berbagai aplikasi, dari aplikasi desktop hingga perangkat mobile dan web.

  Contoh: Aplikasi pengolah kata seperti Microsoft Word atau aplikasi ponsel seperti WhatsApp.

- Penyelesaian Masalah: Pemrograman adalah alat untuk menyelesaikan masalah dengan membuat solusi berbasis komputer yang efisien dan efektif.

  Contoh: Mengembangkan perangkat lunak yang membantu mengelola inventaris barang dalam sebuah toko.

b. Tujuan Pemrograman

- **Mengotomatisasi Tugas**: Pemrograman memungkinkan kita untuk membuat perangkat lunak yang otomatis menjalankan tugas-tugas tertentu tanpa perlu campur tangan manusia secara langsung. Ini mirip dengan bagaimana mesin cuci otomatis mencuci pakaian tanpa kita harus melakukannya secara manual.

  Contoh: Misalnya, jika Anda bekerja di sebuah perusahaan yang memerlukan laporan bulanan, Anda dapat menulis program yang

secara otomatis mengumpulkan data dan membuat laporan tanpa harus melakukannya secara manual setiap bulan.

- **Pengembangan Perangkat Lunak**: Pemrograman adalah alat utama untuk membuat perangkat lunak yang kita gunakan setiap hari, mulai dari aplikasi ponsel hingga sistem operasi komputer.

  Contoh: Aplikasi seperti WhatsApp, yang Anda gunakan untuk berkomunikasi, adalah hasil dari pemrograman. Tanpa pemrograman, aplikasi ini tidak akan ada.

- **Pemecahan Masalah:** Pemrograman membantu kita menyelesaikan masalah kompleks dengan mengembangkan solusi berbasis komputer. Ini seperti menyusun puzzle yang sangat besar dan rumit di mana komputer membantu menyatukan semua potongan.

  Contoh: Jika Anda ingin menghitung jumlah total pengeluaran Anda selama sebulan, Anda bisa menulis program untuk menjumlahkan angka-angka tersebut dengan cepat dan akurat.

- **Pengembangan Kreativitas:** Pemrograman memungkinkan kita untuk mewujudkan ide-ide kreatif menjadi aplikasi nyata. Ini seperti melukis dengan kode komputer sebagai kuas Anda.

  Contoh: Jika Anda punya ide untuk sebuah game video atau aplikasi pendidikan, Anda bisa menggunakan pemrograman untuk merealisasikan ide tersebut dan melihat hasilnya.

c. Pentingnya Pemrograman

- **Automatisasi :** Pemrograman memungkinkan otomatisasi tugas-tugas rutin, mengurangi kebutuhan akan intervensi manusia, dan meningkatkan efisiensi.

  Contoh: Menggunakan skrip untuk otomatisasi pengolahan data atau pembuatan laporan.

- **Inovasi Teknologi :** Dengan pemrograman, kita dapat menciptakan teknologi baru yang dapat mengubah cara kita hidup dan bekerja.

  Contoh: Teknologi seperti kecerdasan buatan (AI) dan Internet of Things (IoT) yang memerlukan pemrograman untuk berfungsi.

- **Keterampilan Profesional :** Keterampilan pemrograman adalah kemampuan yang sangat dicari di berbagai industri, dari teknologi hingga

kesehatan, karena hampir semua sektor memanfaatkan perangkat lunak dan teknologi komputer.

Contoh: Pengembang perangkat lunak, analis data, dan insinyur perangkat keras semua memerlukan keterampilan pemrograman.

- **Pengembangan Kreatif :** Pemrograman memberikan alat untuk mengekspresikan kreativitas melalui pembuatan aplikasi, game, dan sistem yang inovatif.

  Contoh: Membuat aplikasi permainan atau perangkat lunak kreatif seperti perangkat grafis dan desain.

4. Pengenalan Python

a. Sejarah Python

Python adalah bahasa pemrograman tingkat tinggi yang dikembangkan oleh Guido van Rossum. Bahasa ini pertama kali dirilis pada 20 Februari 1991. Guido van Rossum memulai proyek Python pada akhir 1980-an sebagai penerus dari bahasa pemrograman ABC, yang dikembangkan di Centrum Wiskunde & Informatica (CWI) di Belanda. Python diciptakan dengan tujuan untuk menyederhanakan proses pengembangan perangkat lunak dengan menyediakan bahasa yang bersih dan mudah dipahami.

Nama "Python" diambil dari grup komedi Inggris terkenal, Monty Python. Van Rossum menginginkan nama yang pendek, unik, dan sedikit misterius, yang mudah diingat oleh orang-orang. Hal ini juga mencerminkan filosofi di balik Python, yang tidak hanya bertujuan untuk menjadi bahasa yang kuat tetapi juga menyenangkan untuk digunakan.

Python terus berkembang dan menjadi salah satu bahasa pemrograman yang paling populer di dunia. Pada tahun 2000, Python 2.0 dirilis, membawa sejumlah fitur baru seperti pengumpulan sampah otomatis dan dukungan untuk Unicode. Kemudian, pada tahun 2008, Python 3.0 diperkenalkan sebagai versi yang tidak kompatibel ke belakang dengan Python 2, tetapi membawa peningkatan besar dalam hal konsistensi sintaks dan pengelolaan memori.

b. Fitur Utama Python

- **Sintaks yang mudah dibaca dan ditulis :** Python dirancang dengan fokus pada keterbacaan kode. Sintaks Python menyerupai bahasa Inggris,

membuatnya lebih intuitif dan mudah dipelajari, terutama bagi pemula. Struktur kode yang menggunakan indentasi juga memaksa programmer untuk menulis kode yang rapi dan terstruktur.

- **Interpreted Language:** Python adalah bahasa yang dieksekusi oleh interpreter baris per baris. Artinya, Python tidak perlu dikompilasi sebelum dijalankan, yang memungkinkan pengembang untuk menjalankan kode dengan cepat dan melihat hasilnya segera. Ini sangat membantu dalam proses debugging dan pengembangan secara iteratif.
- **Dukungan Multiplatform:** Python dapat dijalankan di berbagai sistem operasi seperti Windows, macOS, Linux, dan lainnya tanpa perubahan signifikan pada kode. Ini menjadikannya pilihan yang fleksibel bagi pengembang yang perlu membangun aplikasi lintas platform.
- **Pustaka standar yang luas:** Salah satu kekuatan utama Python adalah pustaka standarnya yang sangat luas, yang mencakup modul-modul untuk berbagai tugas seperti pemrosesan string, manipulasi file, komunikasi jaringan, pemrograman sistem, dan banyak lagi. Selain itu, ada ribuan pustaka pihak ketiga yang tersedia, yang memperluas kemampuan Python untuk bidang seperti pengembangan web (Django, Flask), analisis data (Pandas, NumPy), pembelajaran mesin (TensorFlow, scikit-learn), otomatisasi, dan lain-lain.
- **Open Source:** Python adalah perangkat lunak sumber terbuka, yang berarti siapa saja dapat menggunakannya, memodifikasi, dan mendistribusikan kembali bahasa ini tanpa biaya. Komunitas Python yang besar dan aktif secara terus-menerus mengembangkan bahasa ini, memperbaiki bug, dan menambahkan fitur-fitur baru.
- **Dukungan untuk berbagai pradigma pemrograman:** Python mendukung beberapa paradigma pemrograman termasuk pemrograman berorientasi objek, pemrograman fungsional, dan pemrograman prosedural. Fleksibilitas ini memungkinkan pengembang untuk memilih pendekatan yang paling sesuai untuk proyek mereka.
- **Ekosistem yang kuat dan komunitas yang aktif:** Python memiliki ekosistem yang kaya dengan berbagai framework, pustaka, dan alat pengembangan yang mendukung berbagai domain aplikasi. Selain itu,

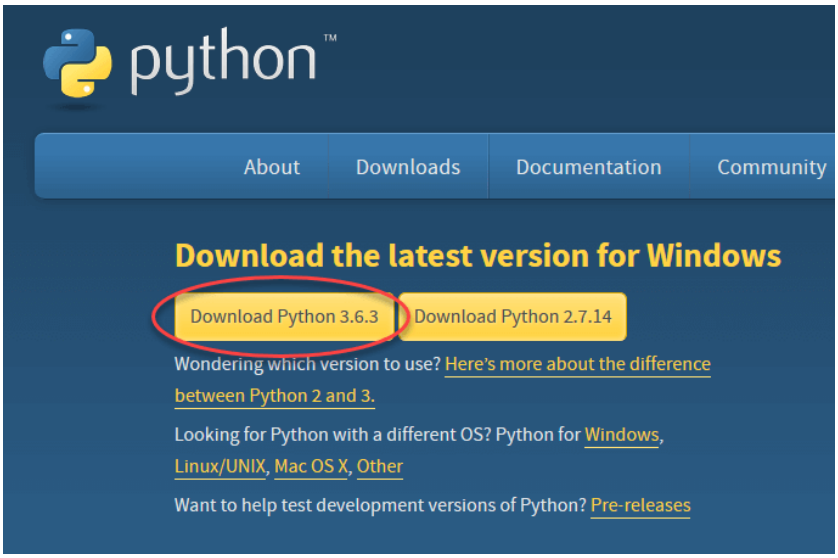
komunitas Python yang besar dan aktif memberikan dukungan, dokumentasi, dan pembaruan yang terus-menerus, menjadikan Python sebagai salah satu bahasa yang paling cepat berkembang.

c. Instalasi dan Pengaturan Lingkungan (PyCharm)

Untuk memulai pemrograman dengan Python, Anda perlu menginstal Python terlebih dahulu dan memilih lingkungan pengembangan yang sesuai. Salah satu Integrated Development Environment (IDE) yang populer digunakan untuk pengembangan Python adalah **PyCharm**.

Langkah 1: Instalasi Python

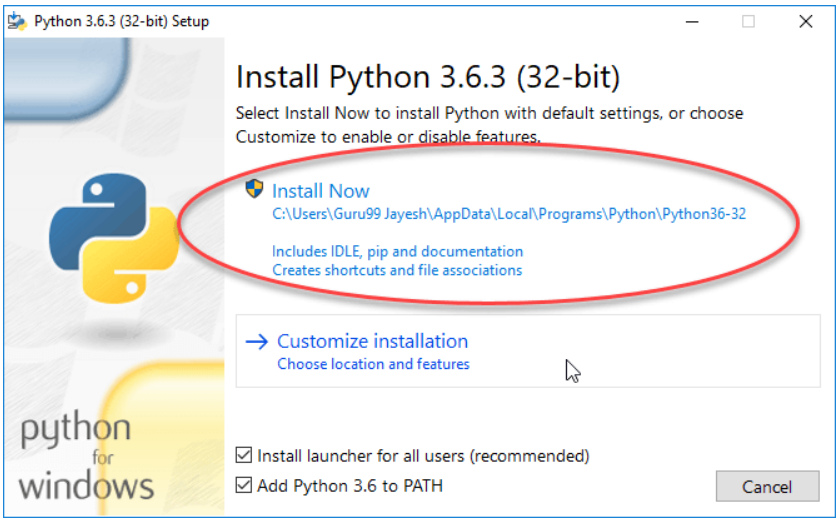
1. **Unduh Python:**
   - Buka peramban web Anda dan kunjungi situs resmi Python di https://www.python.org/.
   - Di halaman utama, Anda akan melihat tombol "Download Python". Situs web secara otomatis mendeteksi sistem operasi Anda dan memberikan tautan unduhan yang sesuai.
   - Pilih versi python lalu Klik tombol tersebut untuk mengunduh penginstal Python



Gambar 1.6 Instalasi Python (1)

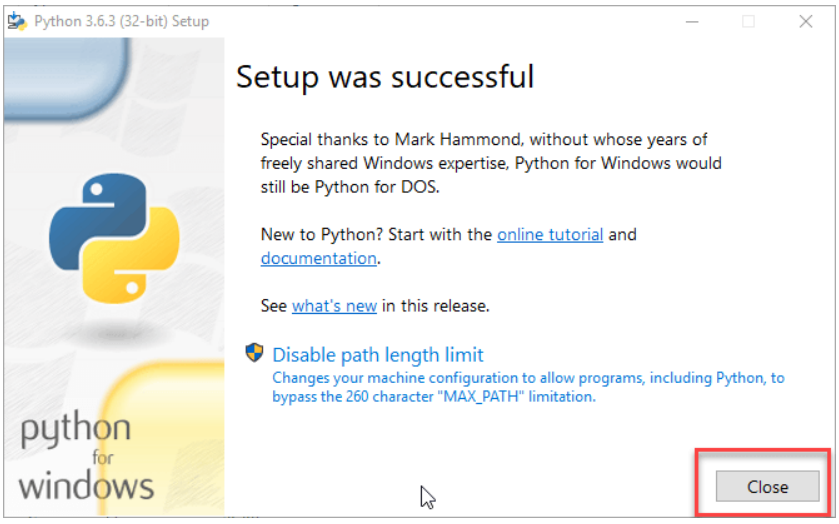
2. **Instalasi Python**
   - Setelah unduhan selesai, cari file penginstal di direktori unduhan Anda dan klik dua kali untuk menjalankan file.exe.

- Saat jendela instalasi muncul, pastikan untuk mencentang kotak “install launcher for all user (recommended)” dan centang kotak yang bertuliskan "Add Python to PATH" di bagian bawah jendela. Ini penting agar Anda dapat menjalankan Python dari baris perintah.
- Klik "Install Now" untuk melanjutkan. Instalasi akan memakan waktu beberapa menit.



Gambar 1.7 Instalasi Python (2)

- Menunggu instalasi python selesai
- Jika sudah selesai, akan muncul layar yang menyatakan bahwa penginstalan telah berakhir atau sukses , lalu klik “close”

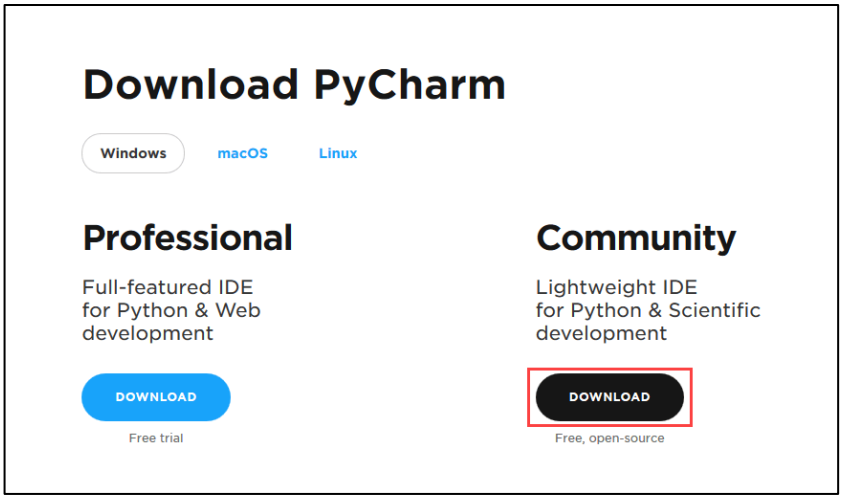


Gambar 1.8 Instalasi Python (3)

Langkah 2: Pengaturan Lingkungan di PyCharm

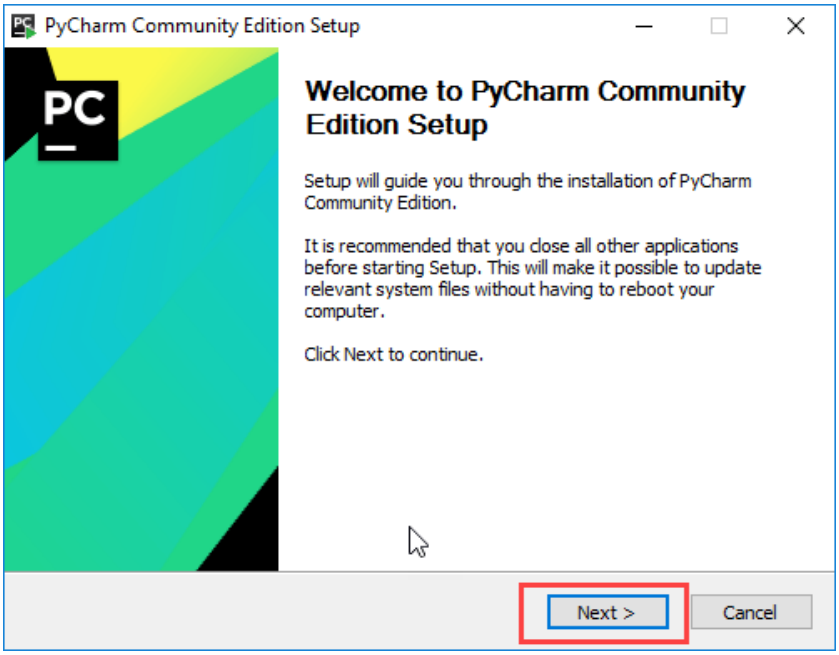
1. Unduh dan Instal PyCharm

- Kunjungi situs resmi PyCharm di https://www.jetbrains.com/pycharm/.
- Di halaman unduhan, Anda akan melihat dua versi PyCharm: Community dan Professional. Versi Community gratis dan cukup untuk sebagian besar kebutuhan pemrograman dasar. Versi Professional berbayar dan menawarkan fitur tambahan yang lebih canggih.
- Klik tombol "Download" di bawah versi yang Anda pilih.



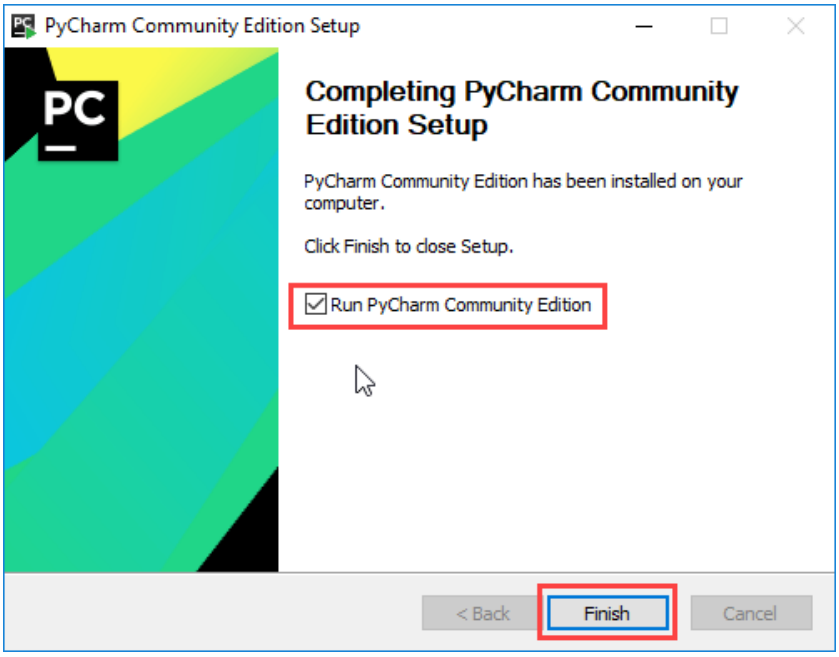
Gambar 1.9 Instalasi PyCharm (1)

- Setelah unduhan selesai, buka file instalasi dan ikuti instruksi untuk menginstal PyCharm di komputer Anda.



Gambar 1.10 Instalasi PyCharm (2)

- Setelah instalasi selesai, akan muncul layar bahwa pycharm sudah terinstal, jika ingin langsung melanjutkan dan menjalankannya, centang pada kotak “Run PyCharm Community Edition” lalu klik finish.



Gambar 1.11 Instalasi PyCharm (3)

2. **Membuat Proyek Baru di PyCharm**
   - Setelah instalasi selesai, buka PyCharm. Pada jendela awal, pilih opsi "New Project" untuk membuat proyek baru.
   - Anda akan diminta untuk memilih lokasi penyimpanan proyek. Tentukan direktori tempat Anda ingin menyimpan proyek Anda.
   - Selanjutnya, pilih Python Interpreter. Jika Python sudah terinstal di sistem Anda, PyCharm biasanya akan mendeteksinya secara otomatis. Jika tidak, Anda dapat menambahkannya secara manual dengan memilih lokasi instalasi Python.
3. **Pengaturan Lingkungan Virtual (Virtual Environment – venv)**
   - Menggunakan virtual environment sangat disarankan untuk mengisolasi dependensi proyek Anda agar tidak bercampur dengan proyek lain.
   - Saat membuat proyek baru, PyCharm akan menawarkan opsi untuk membuat virtual environment. Pilih opsi ini untuk membuat venv.
   - Jika Anda sudah membuat proyek tanpa virtual environment, Anda masih bisa menambahkannya.

Caranya, buka menu **File** -> **Settings** -> **Project**: [**nama proyek**] -> Python Interpreter, lalu pilih opsi untuk membuat virtual environment baru.

- PyCharm akan secara otomatis mengatur lingkungan ini dan Anda bisa langsung mulai menggunakannya.

4. **Memulai Pemrograman**
   - Setelah semua pengaturan selesai, Anda dapat mulai menulis kode Python Anda. Buat file Python baru dengan klik kanan pada direktori proyek, pilih New -> Python File, beri nama file tersebut, dan mulai menulis kode Anda.
   - Untuk menjalankan kode, Anda bisa klik kanan pada file dan pilih "Run", atau menggunakan tombol hijau kecil di pojok kanan atas jendela PyCharm.

d. Struktur Dasar Program Python

Program Python biasanya dimulai dengan menuliskan perintah atau kode yang akan dijalankan. Struktur dasar dari program Python terdiri dari beberapa elemen berikut:

1. **Komentar:** Digunakan untuk menambahkan deskripsi atau catatan dalam kode. Komentar diawali dengan tanda '#'.

```
# Ini adalah komentar
print("Hello, World!")
```

2. **Pernyataan atau Statement:** Python menjalankan perintah berdasarkan pernyataan yang diberikan. Contohnya adalah pernyataan untuk mencetak teks di layar.

```
print("Selamat belajar Python!")
```

3. **Variable dan Tipe Data:** Variabel digunakan untuk menyimpan data, yang bisa berupa berbagai tipe seperti angka, teks, atau daftar.

```
nama = "Andi"
usia = 21
```

4. **Fungsi:** Fungsi adalah blok kode yang dapat digunakan kembali, yang melakukan tugas tertentu. Fungsi dalam Python didefinisikan menggunakan kata kunci '`def`'.

```
def sapa():
    print("Halo, selamat datang!")

sapa()
```

5. **Kondisi dan Pengulangan:** Python mendukung struktur kontrol seperti pernyataan if-else dan loop seperti for dan while.

```
if usia >= 18:
    print("Anda sudah dewasa.")

for i in range(5):
    print(i)
```

6. **Input dan Output:** Python menyediakan cara untuk mengambil input dari pengguna dan menampilkan output.

```
nama = input("Masukkan nama Anda:
")
print("Nama Anda adalah " + nama)
```

B. Latihan/Contoh

Latihan 1: Menulis algoritma untuk aktivitas sehari-hari.

1. **Tujuan:** Latihan ini dirancang untuk membantu mahasiswa memahami konsep dasar algoritma dengan mengaplikasikannya pada aktivitas sehari-hari. Mahasiswa akan belajar bagaimana memecah suatu tugas menjadi langkah-langkah terstruktur yang dapat diikuti oleh komputer atau orang lain.
2. **Instruksi:**

a. **Pilih Aktivitas:** Pilih satu aktivitas sehari-hari yang sederhana dan sering dilakukan, misalnya:
    - Membuat secangkir teh.
    - Mengunci pintu.
    - Mengirim pesan teks.
b. **Tulis Algoritma:** Buat algoritma yang terdiri dari langkah-langkah yang jelas dan berurutan untuk menyelesaikan aktivitas tersebut. Pastikan setiap langkah ditulis dengan detail sehingga bisa diikuti tanpa kebingungan.

3. **Contoh:**

   Algoritma untuk membuat secangkir teh:
   - Ambil teko dan isi dengan air.
   - Panaskan air hingga mendidih.
   - Ambil sebuah cangkir.
   - Masukkan satu kantong teh ke dalam cangkir.
   - Tuangkan air mendidih ke dalam cangkir.
   - Tunggu selama 3-5 menit hingga teh menyeduh.
   - Angkat kantong teh dari cangkir.
   - Tambahkan gula atau susu sesuai selera.
   - Aduk teh dengan sendok.
   - Teh siap untuk diminum.
4. **Diskusi:** Setelah menyusun algoritma, diskusikan dengan teman sekelas atau dosen mengenai bagaimana algoritma tersebut dapat disederhanakan atau ditingkatkan. Apakah ada langkah yang bisa dihilangkan atau dijelaskan lebih baik?
5. **Hasil yang Diharapkan :** Mahasiswa akan dapat memecah suatu tugas menjadi langkah-langkah yang dapat diikuti dan mengerti pentingnya ketepatan dan kejelasan dalam menyusun algoritma.

Latihan 2: Menulis dan menjalankan program "Hello, World!" dalam Python.

1. **Tujuan :** Latihan ini dirancang untuk memperkenalkan mahasiswa pada bahasa pemrograman Python dan lingkungan pengembangannya. Mahasiswa akan belajar bagaimana menulis dan menjalankan program sederhana menggunakan Python.
2. **Instruksi**
   a. **Siapkan Lingkungan Pengembangan:**

- Pastikan Python telah terinstal di komputer.
- Buka PyCharm (atau IDE Python lain yang digunakan).
- Buat proyek baru di PyCharm dengan nama "Latihan_Hello_World".

b. Menulis Program:

- Dalam proyek baru, buat file Python baru dengan nama ‘hello_world.py’.
- Ketik kode berikut ke dalam file tersebut:
  print("Hello, World!")
- Penjelasan Kode:
  - ‘print()’ adalah fungsi bawaan Python yang digunakan untuk menampilkan teks atau output lainnya ke layar.
  - "Hello, World!" adalah string yang akan dicetak oleh program.

c. Menjalankan Program:

- Klik tombol 'Run' di PyCharm atau jalankan file ‘hello_world.py’ melalui terminal.
- Lihat hasilnya di konsol: program akan menampilkan teks ‘Hello, World!.’

d. Modifikasi Program:

- Coba modifikasi teks yang ditampilkan oleh program menjadi nama mahasiswa atau pesan lain.
- Contoh modifikasi:
  print("Selamat datang di dunia pemrograman, [Nama Anda]!")
- Jalankan kembali program untuk melihat perubahan yang terjadi.

e. Diskusi:

- Diskusikan pentingnya menulis program "Hello, World!" sebagai langkah pertama dalam mempelajari bahasa pemrograman baru.
- Bagaimana modifikasi program dapat membantu dalam memahami cara kerja dasar Python?

3. **Hasil yang diharapkan :** Mahasiswa akan familiar dengan menulis dan menjalankan program dasar di Python, serta memahami proses pengembangan perangkat lunak dari ide sederhana hingga eksekusi.

C. Rangkuman

Algoritma adalah langkah-langkah sistematis untuk menyelesaikan masalah, mulai dari mengidentifikasi masalah hingga penyelesaian akhir, dengan contoh seperti resep

masakan dan teknik pencarian data seperti pencarian linear dan biner. Pemrograman, proses penulisan kode untuk mengembangkan perangkat lunak, menggunakan bahasa pemrograman seperti Python, yang terkenal dengan sintaks yang mudah dan pustaka standar yang luas. Python, dikembangkan oleh Guido van Rossum, adalah bahasa yang cocok untuk pemula dan digunakan dalam berbagai aplikasi. Untuk pemrograman dasar, pemahaman algoritma seperti bubble sort dan merge sort serta implementasi kode Python sederhana seperti pertukaran variabel dan pencarian linear sangat penting dalam membangun keterampilan dasar pemrograman.

D. Tugas

Tugas 1 : Menulis

Tuliskan algoritma untuk aktivitas sehari-hari seperti:

1. Menyiapkan sarapan (misalnya membuat sandwich).
2. Mengatur jadwal harian (misalnya, rutinitas pagi).

   *catatan “Tulis algoritma dalam bentuk langkah-langkah terstruktur yang jelas.”

Tugas 2 : Coding

1. **Implementasi Algoritma Pencarian Linear**

   Deskripsi: Buatlah sebuah program Python yang melakukan pencarian linear pada sebuah daftar angka. Program ini harus menerima daftar angka dan nilai yang ingin dicari sebagai input, lalu mengembalikan indeks dari nilai yang dicari jika ditemukan, atau -1 jika tidak ditemukan.

   Petunjuk:

   1. Implementasikan fungsi ‘pencarian_linear(data, target)’ yang menerima dua parameter: ‘data’ (daftar angka) dan ‘target’ (angka yang dicari).
   2. Fungsi harus mengembalikan indeks dari ‘target’ dalam ‘data’ jika ditemukan, atau -1 jika tidak ditemukan.
   3. Tambahkan beberapa contoh pengujian untuk memastikan fungsi bekerja dengan baik.

2. **Implementasi Algoritma Bubble Sort**

   Deskripsi: Buatlah sebuah program Python yang mengurutkan sebuah daftar angka menggunakan algoritma bubble sort. Program ini harus menerima daftar angka sebagai input dan mengembalikan daftar yang sudah terurut.

   Petunjuk:

1. Implementasikan fungsi ‘bubble_sort(data)’ yang menerima satu parameter: ‘data’ (daftar angka).
2. Fungsi harus mengurutkan ‘data’ menggunakan algoritma bubble sort dan mengembalikan daftar yang sudah terurut.
3. Tambahkan beberapa contoh pengujian untuk memastikan fungsi bekerja dengan baik.

E. Pustaka

Kadir, A. (2016). Dasar-Dasar Algoritma. Yogyakarta: Andi Offset.

Lutz, M. (2013). Learning Python. O'Reilly Media.

Mulyono, D. (2018). Pemrograman Komputer dengan C dan Python. Yogyakarta: Graha Ilmu.

Purnomo, H. (2020). Belajar Python untuk Pemula. Bandung: Informatika.

Suryadi, S. (2021). Pemrograman Dasar dengan Python. Bandung: CV. Alfabeta.

Van Rossum, G., & Drake, F. L. (2009). The Python Language Reference Manual. Beaverton: Python Software Foundation.

Modul 2: Array & Linked List

Modul ini dirancang untuk membantu mahasiswa memahami dua struktur data fundamental dalam pemrograman, yaitu Array dan Linked List, dengan penekanan pada konsep, implementasi, dan aplikasinya dalam berbagai masalah komputasi. Modul ini memiliki relevansi yang signifikan, karena pemahaman mendalam tentang Array dan Linked List merupakan dasar yang penting untuk mempelajari struktur data yang lebih kompleks dan teknik pemrograman lanjutan. Dengan mengikuti modul ini, mahasiswa diharapkan mampu menguasai konsep dasar Array dan Linked List, memahami bagaimana kedua struktur data ini bekerja dalam menyimpan dan mengorganisir data, serta mampu menerapkannya dalam pengembangan program-program yang lebih efisien dan efektif. Penguasaan materi ini merupakan langkah awal yang esensial dalam membangun keterampilan pemrograman yang lebih mendalam dan kemampuan analisis algoritma.

A. Penyajian Materi Array

1. Array

Array adalah struktur data yang paling umum dan tersedia di hampir semua bahasa pemrograman. Karena popularitasnya, array menjadi titik awal yang baik untuk mempelajari struktur data dan memahami hubungan antara pemrograman berorientasi objek dan struktur data. Dalam bab ini, kita akan mempelajari array di Java dan membuat kelas array sederhana. Kita juga akan melihat array yang disusun dalam urutan tertentu, di mana data disimpan dalam urutan menaik (atau menurun) berdasarkan kunci. Pengaturan ini memungkinkan kita mencari data dengan cepat menggunakan metode pencarian biner.

Array adalah struktur dasar yang digunakan untuk menyimpan dan mengakses kumpulan data. Array bisa membantu menyelesaikan berbagai masalah dalam ilmu komputer. Sebagian besar bahasa pemrograman menyediakan array sebagai tipe data dasar dan memungkinkan pembuatan array dengan berbagai dimensi. Dalam bab ini, kita akan mulai dengan array satu dimensi, lalu menggunakannya untuk membuat array dua dimensi dan struktur matriks.

Array adalah variabel khusus yang dapat menyimpan beberapa nilai sekaligus. Misalnya, jika kita ingin menyimpan daftar nama sepeda motor, kita bisa menggunakan array untuk menyimpan semua nama dalam satu variabel. Tanpa array, kita harus mendeklarasikan variabel satu per satu, seperti ini:

```
SepedaMotor1 = "Ninja"
SepedaMotor2 = "Mio"
SepedaMotor3 = "Nmax"
```

Namun, jika kita ingin menyimpan dan mengakses banyak data, misalnya 500 jenis sepeda motor di sebuah showroom, akan lebih efisien menggunakan array. Array memungkinkan kita menyimpan semua data dalam satu variabel dan mengaksesnya dengan mudah menggunakan indeks, yaitu posisi elemen dalam array yang dimulai dari 0, 1, 2, dan seterusnya.

2. Pengaksesan Elemen Array

a. Mengakses dan Mengubah Elemen dalam Array

Dalam Python, elemen dalam sebuah array diakses dengan menggunakan indeks. Indeks ini adalah nomor urut dari elemen dalam array, dimulai dari 0

untuk elemen pertama. Misalnya, jika kita ingin mengakses data dari elemen pertama array ‘mobil’, kita bisa menulis:

```
x = mobil[0]
```

Jika ingin mengubah nilai elemen pertama pada array ‘mobil’ menjadi "Honda", kita bisa menulis:

```
mobil[0] = "Honda"
```

b. Menghitung Panjang Array

Untuk mengetahui jumlah elemen dalam array, kita bisa menggunakan fungsi ‘len()’. Fungsi ini mengembalikan jumlah elemen dalam array. Misalnya, untuk menghitung jumlah elemen dalam array ‘mobil’:

```
x = len(mobil)
```

c. Looping Melalui Elemen Array

Untuk melakukan looping melalui semua elemen dalam array, kita bisa menggunakan loop ‘for’. Misalnya, untuk mencetak setiap elemen dalam ‘array mobil’:

```
for x in mobil:
    print(x)
```

d. Menambah Elemen ke dalam Array

Untuk menambahkan elemen baru ke dalam array, kita bisa menggunakan metode ‘append()’. Contohnya, untuk menambahkan "Honda" ke array ‘mobil’:

```
mobil.append("Honda")
```

e. Menghapus Elemen dari Array

Ada beberapa cara untuk menghapus elemen dari array. Salah satunya adalah menggunakan metode ‘pop()’, yang menghapus elemen pada posisi tertentu. Misalnya, untuk menghapus elemen kedua dalam array ‘mobil’:

```
mobil.pop(1)
```

Anda juga bisa menggunakan metode '`remove()`' untuk menghapus elemen berdasarkan nilainya. Misalnya, untuk menghapus elemen dengan nilai "Volvo":

```
mobil.remove("Volvo")
```

Catatan: Metode `remove()` hanya akan menghapus kemunculan pertama dari nilai yang ditentukan.

f. Metode lain untuk manipulasi Array

Python menyediakan berbagai metode bawaan yang dapat digunakan untuk memanipulasi list (array):

- `append()`: Menambahkan elemen di akhir list.
- `clear()`: Menghapus semua elemen dalam list.
- `copy()`: Membuat salinan dari list.
- `count()`: Menghitung jumlah elemen dengan nilai tertentu.
- `index()`: Mengembalikan posisi dari elemen pertama dengan nilai tertentu.
- `insert()`: Menambahkan elemen pada posisi tertentu.
- `pop()`: Menghapus elemen pada posisi tertentu.
- `remove()`: Menghapus elemen pertama dengan nilai tertentu.
- `reverse()`: Membalik urutan elemen dalam list.
- `sort()`: Mengurutkan elemen dalam list.

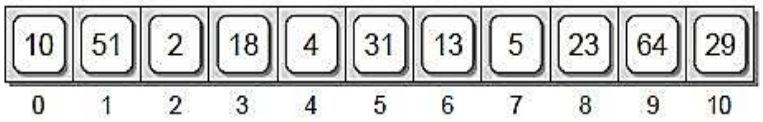
Catatan: Python tidak memiliki dukungan bawaan untuk array seperti di beberapa bahasa pemrograman lainnya, tetapi list di Python dapat digunakan sebagai penggantinya.

3. Struktur Array

Pada tingkat perangkat keras, sebagian besar arsitektur komputer menyediakan mekanisme untuk menyimpan dan mengakses array satu dimensi. Array satu dimensi terdiri dari beberapa elemen yang disimpan bersebelahan dalam memori dan memungkinkan akses langsung ke setiap elemen melalui indeks. Setiap array diidentifikasi dengan satu nama, dan elemen-elemen dalam array dapat diakses menggunakan indeks integer yang menunjukkan posisi elemen dalam array. Ini mirip dengan notasi matematika seperti xix_ixi, di mana beberapa variabel

memiliki nama yang sama tetapi dibedakan oleh indeksnya. Dalam pemrograman, indeks ini biasanya ditempatkan dalam tanda kurung siku setelah nama array, seperti `x[i]`.

Contoh: Struktur array satu dimensi yang terdiri dari 11 elemen bisa digambarkan sebagai berikut.



Gambar Contoh array 1-D yang terdiri dari 11 elemen

a. Perbedaan antara array dan list di python

Meskipun array memiliki struktur yang mirip dengan list di Python, terdapat dua perbedaan utama antara keduanya. Pertama, array memiliki jumlah operasi yang lebih terbatas, yaitu hanya mencakup pembuatan, pembacaan, dan penulisan nilai elemen tertentu. Sebaliknya, list di Python menyediakan berbagai macam operasi untuk memanipulasi isinya. Kedua, ukuran array bersifat tetap setelah dibuat, sementara list dapat bertambah atau berkurang selama program berjalan.

b. Mengapa memilih array

Jika Python sudah menyediakan list yang fleksibel, mengapa masih perlu membahas array? Jawabannya adalah bahwa kedua struktur data ini memiliki kegunaannya masing-masing. Array lebih cocok untuk situasi di mana jumlah elemen diketahui sebelumnya dan tidak berubah, sementara list lebih baik digunakan ketika ukuran urutan data dapat berubah seiring waktu. Sebagai contoh, jika Anda membutuhkan struktur data untuk menyimpan 100.000 elemen, menggunakan list mungkin akan mengalokasikan ruang dua kali lipat dari yang sebenarnya diperlukan, yang bisa menjadi pemborosan memori. Dalam kasus seperti ini, array adalah pilihan yang lebih efisien.

c. Operasi dasar pada array satu dimensi

- Membuat Array: `Array(size)`: Membuat array satu dimensi dengan jumlah elemen yang ditentukan oleh `size`. Setiap elemen diinisialisasi dengan nilai `None`. Ukuran array harus lebih besar dari nol.
- Mengukur Panjang Array: `length()`: Mengembalikan jumlah elemen dalam array.

- Mengakses Nilai Elemen: `getitem(index)`: Mengembalikan nilai yang disimpan dalam array pada posisi yang ditentukan oleh `index`. Indeks harus berada dalam rentang yang valid.
- Memodifikasi Elemen Array: `setitem(index, value)`: Mengubah nilai elemen pada posisi yang ditentukan oleh `index` menjadi `value`. Indeks harus berada dalam rentang yang valid.
- Menghapus Isi Array: `clearing(value)`: Mengosongkan array dengan mengatur semua elemennya ke nilai tertentu.
- Membuat Iterator: `iterator()`: Menghasilkan iterator yang dapat digunakan untuk menjelajahi elemen-elemen dalam array.

Meskipun array sering dianggap sebagai struktur fisik karena diimplementasikan langsung pada tingkat perangkat keras, dalam banyak kasus kita juga membutuhkannya sebagai tipe data abstrak (ADT) dengan lebih banyak operasi tambahan seperti iterator dan pengaturan ukuran array. Dengan menyediakan abstraksi pada tingkat yang lebih tinggi ini, kita bisa lebih fleksibel dalam mengimplementasikan algoritma yang efisien.

4. Implementasi array

Python adalah bahasa pemrograman tingkat tinggi yang ditulis menggunakan bahasa C. C, sebagai bahasa pemrograman yang kuat, menawarkan sintaks yang memungkinkan akses penuh ke fungsionalitas perangkat keras yang mendasarinya. Meskipun C menyediakan kemampuan ini, sintaksnya bisa terasa rumit dibandingkan dengan Python, terutama bagi programmer Python yang belum berpengalaman dengan C.

Banyak tipe data dan kelas di Python sebenarnya diimplementasikan menggunakan tipe data yang sesuai dari bahasa C. Meskipun Python tidak secara langsung menyediakan struktur array sebagai bagian dari bahasanya, ia menawarkan modul seperti `ctypes` dalam Pustaka Standar Python. Modul ini memungkinkan akses ke tipe data dan fungsi dari bahasa C, serta dapat digunakan untuk berinteraksi dengan berbagai koleksi data seperti string, list, tuple, dan kamus di Python.

Namun, modul `ctypes` tidak dirancang untuk penggunaan sehari-hari dalam pengembangan aplikasi Python. Sebaliknya, modul ini lebih banyak

digunakan oleh pengembang modul Python untuk menciptakan modul yang lebih portabel dengan menjembatani antara Python dan C.

Sebagian besar fungsionalitas yang ditawarkan oleh `ctypes` memerlukan pengetahuan tentang bahasa C. Oleh karena itu, teknik untuk mengimplementasikan array menggunakan `ctypes` biasanya tidak digunakan langsung dalam program Python sehari-hari. Sebagai gantinya, kita dapat menggunakan teknik ini dalam kelas array kita sendiri untuk menyediakan fungsionalitas yang diperlukan sesuai dengan konsep Abstract Data Type (ADT) dari array, dengan menyembunyikan detail implementasi di dalam kelas.

Array adalah struktur data yang dapat menyimpan sejumlah item dengan tipe yang sama. Banyak struktur data menggunakan array untuk mengimplementasikan algoritma mereka. Berikut adalah istilah-istilah penting dan operasi dasar yang terkait dengan array:

- **Elemen**: Setiap item yang disimpan dalam array disebut elemen.
- **Indeks**: Setiap lokasi elemen dalam array memiliki indeks numerik yang digunakan untuk mengidentifikasi elemen tersebut.

Operasi Dasar pada Array:

1. Traverse: Mencetak semua elemen dalam array satu per satu.
2. Penyisipan: Menambahkan elemen pada indeks tertentu.
3. Penghapusan: Menghapus elemen pada indeks tertentu.
4. Pencarian: Mencari elemen berdasarkan indeks atau nilai.
5. Perbarui: Memperbarui elemen pada indeks tertentu

B. Penyajian Materi Linked List

1. Linked List

Linked list merupakan salah satu struktur data fundamental dalam bidang ilmu komputer. Struktur ini memungkinkan programmer untuk menyimpan data dengan fleksibilitas tinggi sesuai kebutuhan. Berbeda dengan array, linked list memiliki kemampuan untuk mengalokasikan memori secara dinamis selama program berjalan, sehingga lebih efisien dalam penggunaan sumber daya. Dalam linked list, data disimpan dalam sebuah struktur yang terdiri dari objek-objek yang saling terhubung melalui tautan atau pointer. Setiap elemen dalam linked list

disebut sebagai node, yang berisi dua bagian utama: data yang disimpan dan referensi ke node berikutnya dalam urutan.

Ketika menggunakan array, programmer harus menentukan ukuran array di awal, yang sering kali berakibat pada alokasi memori yang tidak efisien, terutama jika ukuran array yang dibutuhkan sulit diprediksi. Misalnya, alokasi array sebesar 100 elemen mungkin terlalu besar atau terlalu kecil tergantung pada kebutuhan program. Array bersifat statis, sehingga tidak dapat menyesuaikan diri dengan perubahan jumlah data yang disimpan. Sebagai solusinya, linked list menawarkan fleksibilitas dengan memungkinkan penambahan atau pengurangan elemen secara dinamis selama waktu proses.

Linked list dapat dianalogikan dengan rangkaian gerbong kereta api. Mesin kereta mewakili kepala linked list, gerbong-gerbong mewakili data yang disimpan, dan pengait antara gerbong adalah pointer yang menghubungkan satu node ke node berikutnya. Programmer membaca data dalam linked list seperti seorang kondektur yang memeriksa tiket dari satu gerbong ke gerbong berikutnya, mulai dari kepala linked list hingga ke elemen terakhir, yang biasanya ditandai dengan pointer yang menunjuk ke nilai NULL.

a. Efisiensi dan Penggunaan Linked List

Meskipun linked list menawarkan fleksibilitas yang lebih tinggi dibandingkan array, terdapat beberapa kelemahan yang perlu diperhatikan. Salah satu kelemahan utama adalah waktu akses data. Dalam array, data pada indeks ke-n dapat diakses secara langsung, sedangkan pada linked list, untuk mengakses data ke-n, programmer harus melintasi setiap node dari awal hingga mencapai node yang diinginkan, yang memerlukan waktu O(n).

Secara umum, linked list terdiri dari elemen-elemen yang saling terhubung, dan setiap elemen terdiri dari dua bagian: infotype (menyimpan informasi) dan next (menyimpan alamat elemen berikutnya). Sebagai contoh, jika L adalah linked list dan P adalah pointer, maka alamat elemen pertama dalam list L dapat diakses melalui pointer P. Berikut ini adalah beberapa kondisi dan notasi penting dalam linked list:

- List L adalah list kosong jika First(L) = Nil.
- Elemen terakhir dalam linked list dapat dikenali jika Next(Last) = Nil.

Dalam konteks linked list, Nil adalah pengganti dari Null, dan ini biasanya didefinisikan dengan #define Nil Null.

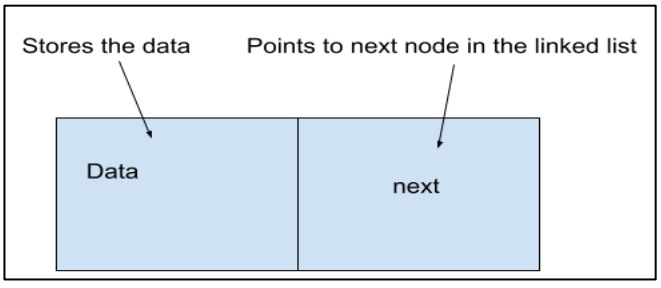
b. Implementasi LIFO dan FIFO dalam Linked List

Dalam implementasi linked list, terdapat dua metode umum yaitu LIFO (Last In First Out) dan FIFO (First In First Out):

- **LIFO (Last In First Out)**: Metode ini mengatur data sehingga elemen yang terakhir dimasukkan akan menjadi elemen yang pertama diakses. Ini mirip dengan cara tumpukan barang, di mana barang yang ditumpuk paling atas akan diambil terlebih dahulu. Dalam linked list, penambahan elemen baru dilakukan pada simpul terakhir, dikenal dengan istilah INSERT.
- **FIFO (First In First Out)**: Metode ini mengatur data sehingga elemen yang pertama kali masuk akan menjadi elemen yang pertama kali diakses. Ini dapat dianalogikan dengan antrean orang membeli tiket, di mana orang yang pertama kali mengantri akan dilayani terlebih dahulu. Dalam linked list dengan metode FIFO, penambahan elemen dilakukan pada simpul yang berada di depan.

2. Node Dalam Linked List

Node adalah elemen dasar dalam struktur data Linked List yang berfungsi sebagai wadah penyimpan data dan memiliki pointer untuk merujuk ke node lain dalam daftar yang saling terhubung. Struktur sederhana dari node ini dapat digambarkan seperti berikut.



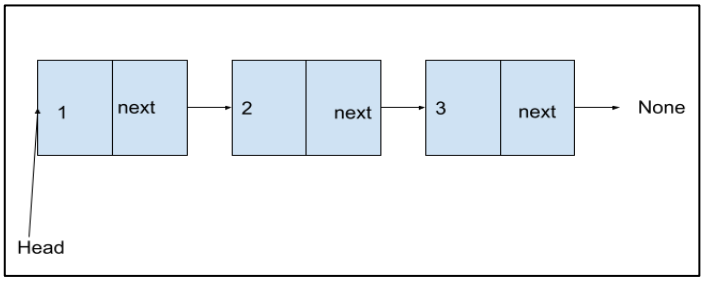
Gambar 1. Struktur Sederhana dari Node

a. Node Linked List dalam Python

Untuk mengimplementasikan node dalam Python, kita dapat membuat kelas '`Node`' yang memiliki dua atribut, yaitu '`data`' untuk menyimpan informasi dan '`next`' yang merupakan pointer ke node berikutnya dalam Linked List.

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
```

Linked List pada dasarnya adalah kumpulan dari beberapa node yang saling terhubung. Ada satu pointer khusus yang disebut **head** yang menunjuk ke node pertama dalam Linked List. Jika Linked List kosong, pointer ini akan menunjuk ke '`None`'. Ilustrasi di bawah ini menunjukkan Linked List yang terdiri dari tiga node.



Gambar 1.2 Ilustrasi Linked List yang terdiri dari tiga node

Dalam Python, Linked List kosong didefinisikan sebagai berikut, di mana pointer **head** menunjuk ke '`None`'.

```
class LinkedList:
    def __init__(self):
        self.head = None
```

b. Menyisipkan Elemen dalam Linked List

Penyisipan elemen dalam Linked List dapat dilakukan pada beberapa posisi: di awal, di akhir, atau di posisi tertentu di antara node-node yang ada. Untuk menyisipkan elemen di awal Linked List, kita pertama-tama membuat node baru dengan data yang diinginkan. Node baru ini kemudian akan diarahkan ke node pertama yang saat ini ditunjuk oleh pointer **head**. Setelah itu, pointer **head** akan di-update untuk menunjuk ke node baru.

```
def insertAtBeginning(self, data):
    new_node = Node(data)
    if self.head is None:
        self.head = new_node
    else:
        new_node.next = self.head
        self.head = new_node
```

Untuk menyisipkan elemen di akhir Linked List, kita perlu menemukan node terakhir, yaitu node yang pointer ‘`next`’-nya menunjuk ke `None`. Setelah ditemukan, kita buat ‘node’ baru dan mengarahkan pointer ‘`next`’ dari node terakhir ke node baru tersebut.

```
def insertAtEnd(self, data):
    new_node = Node(data)
    if self.head is None:
        self.head = new_node
    else:
        current = self.head
        while current.next:
            current = current.next
        current.next = new_node
```

Penyisipan elemen pada posisi tertentu dilakukan dengan menghitung posisi node hingga mencapai posisi yang diinginkan. Setelah itu, node baru dibuat dan diintegrasikan ke dalam Linked List dengan meng-update pointer yang relevan.

```
def insertAtGivenPosition(self, data,
position):
    current = self.head
    count = 1
    while count < position - 1 and current:
        current = current.next
        count += 1
    new_node = Node(data)
    new_node.next = current.next
    current.next = new_node
```

c. Melintasi Linked List

Proses melintasi Linked List dilakukan dengan memulai dari node pertama (yang ditunjuk oleh pointer **head**), mencetak data, dan kemudian bergerak ke node berikutnya hingga mencapai node terakhir (di mana pointer ‘`next`’ adalah ‘`None`’).

```
def traverse(self):
    current = self.head
    while current:
        print(current.data)
        current = current.next
```

d. Menghapus Node

Penghapusan node dalam Linked List dapat dilakukan di awal, di akhir, atau di posisi tertentu di antara node-node. Untuk menghapus node pertama, kita periksa terlebih dahulu apakah Linked List kosong dengan melihat apakah pointer **head** menunjuk ke '`None`'. Jika kosong, exception akan dilempar dengan pesan bahwa Linked List kosong. Jika tidak, node pertama dihapus dengan mengalihkan pointer **head** ke node berikutnya.

```
def delFromBeginning(self):
    try:
        if self.head is None:
            raise Exception("Linked List Kosong")
        else:
            temp = self.head
            self.head = self.head.next
            del temp
    except Exception as e:
        print(str(e))
```

Untuk menghapus node terakhir, kita perlu melintasi Linked List hingga menemukan node yang '`next`'-nya menunjuk ke '`None`'. Node ini kemudian dihapus.

```
def delFromEnd(self):
    try:
        if self.head is None:
            raise Exception("Linked List Kosong")
        else:
            current = self.head
            previous = None
            while current.next:
                previous = current
                current = current.next
            previous.next = None
            del current
    except Exception as e:
        print(str(e))
```

Untuk menghapus node di posisi tertentu, kita melintasi Linked List hingga mencapai node sebelum posisi yang diinginkan. Setelah itu, node pada posisi tersebut dihapus dengan meng-update pointer '`next`'.

```
def delAtPos(self, position):
    try:
        if self.head is None:
            raise Exception("Linked List Kosong")
        else:
            current = self.head
            previous = None
            count = 1
            while current and count < position:
                previous = current
                current = current.next
                count += 1
            previous.next = current.next
            del current
    except Exception as e:
        print(str(e))
```

3. Implementasi Linked List di Python

```
class Node:
    def __init__(self, data=None):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def append(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            return
        last = self.head
        while last.next:
            last = last.next
        last.next = new_node

    def display(self):
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")

    def insert_at_beginning(self, data):
        new_node = Node(data)
        new_node.next = self.head
        self.head = new_node

    def insert_after_node(self, prev_node, data):
        if not prev_node:
            print("Node sebelumnya tidak ada.")
            return
        new_node = Node(data)
        new_node.next = prev_node.next
        prev_node.next = new_node
```

```
    def delete_node(self, key):
        current = self.head

        if current is not None:
            if current.data == key:
                self.head = current.next
                current = None
                return

        while current is not None:
            if current.data == key:
                break
            prev = current
            current = current.next

        if current is None:
            return

        prev.next = current.next
        current = None

# Contoh penggunaan LinkedList
llist = LinkedList()

llist.append(1)
llist.append(2)
llist.append(3)

print("Linked List setelah menambahkan node:")
llist.display()

llist.insert_at_beginning(0)
print("\nLinked List setelah menambahkan node di awal:")
llist.display()

llist.insert_after_node(llist.head.next, 1.5)
print("\nLinked List setelah menambahkan node setelah node
tertentu:")
llist.display()

llist.delete_node(2)
print("\nLinked List setelah menghapus node:"
llist.display()
```

Penjelasan Kode:

1. Node Class: Setiap elemen dalam linked list adalah objek dari kelas Node. Setiap Node memiliki dua atribut: data (untuk menyimpan nilai) dan next (untuk menunjuk ke node berikutnya).
2. LinkedList Class:
   - append(data): Menambahkan node baru di akhir linked list.
   - display(): Menampilkan elemen-elemen dalam linked list.
   - insert_at_beginning(data): Menyisipkan node baru di awal linked list.
   - insert_after_node(prev_node, data): Menyisipkan node baru setelah node tertentu.

- delete_node(key): Menghapus node berdasarkan nilai data (key).

3. Penggunaan:
   - Pertama, linked list dibuat kosong.
   - Kemudian, beberapa node ditambahkan menggunakan append.
   - Node juga bisa ditambahkan di awal linked list dengan insert_at_beginning, atau setelah node tertentu dengan insert_after_node.
   - Node dapat dihapus menggunakan delete_node.

C. Latihan/contoh

Latihan 1: Array Sederhana

Tujuan: Mahasiswa memahami bagaimana cara mendeklarasikan, mengisi, dan mengakses elemen dari sebuah array.

```
# Membuat array berisi 5 angka
angka = [10, 20, 30, 40, 50]

# Mengakses elemen pertama (index 0) dan terakhir (index 4)
print("Elemen pertama:", angka[0])
print("Elemen terakhir:", angka[4])

# Mengubah elemen ketiga (index 2)
angka[2] = 35
print("Array setelah diubah:", angka)

# Menambahkan elemen baru di akhir array
angka.append(60)
print("Array setelah menambahkan elemen baru:", angka)
```

Cara Kerja:

a. Deklarasi Array: `angka = [10, 20, 30, 40, 50]` membuat array yang menyimpan lima angka.
b. Mengakses Elemen: `angka[0]` mengakses elemen pertama, sedangkan `angka[4]` mengakses elemen terakhir.
c. Mengubah Elemen: `angka[2] = 35` mengubah elemen ketiga dari 30 menjadi 35.
d. Menambahkan Elemen: `angka.append(60)` menambahkan angka 60 ke akhir array.

Latihan 2: Linked List Sederhana

Tujuan: Mahasiswa memahami bagaimana cara membuat, menambah, dan menampilkan elemen dalam linked list menggunakan Python.

```python
# Node class untuk Linked List
class Node:
    def __init__(self, data=None):
        self.data = data
        self.next = None

# Linked List class
class LinkedList:
    def __init__(self):
        self.head = None

    # Menambahkan elemen di akhir Linked List
    def append(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node

    # Menampilkan seluruh elemen Linked List
    def display(self):
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")

# Membuat Linked List dan menambahkan beberapa elemen
linked_list = LinkedList()
linked_list.append(10)
linked_list.append(20)
linked_list.append(30)

# Menampilkan Linked List
linked_list.display()
```

Cara Kerja:

a. Node Class: `Node` adalah class yang merepresentasikan satu elemen dari linked list yang memiliki `data` dan `next` untuk menunjuk ke node berikutnya.

b. Linked List Class: `LinkedList` mengelola node-node tersebut. `self.head` menunjuk ke node pertama dalam linked list.

c. Menambah Elemen: `append` menambahkan node baru di akhir linked list.

- Jika linked list kosong (`self.head is None`), node baru akan menjadi head.
- Jika tidak, kita mencari node terakhir (`current.next` bernilai `None`) dan menambahkan node baru di akhir.

d. Menampilkan Elemen: `display` memulai dari head dan mencetak setiap elemen hingga mencapai node terakhir (di mana `current.next` adalah `None`).

D. Rangkuman

Array adalah struktur data dasar yang menyimpan kumpulan elemen dengan tipe yang sama, memungkinkan akses cepat dan efisien menggunakan indeks numerik. Array dapat bersifat satu dimensi atau lebih, seperti array dua dimensi yang

membentuk matriks. Dalam Python, array dapat diakses dan dimodifikasi dengan indeks, dan beberapa metode bawaan seperti append(), pop(), dan remove() memungkinkan manipulasi elemen. Meskipun Python tidak memiliki tipe array khusus, struktur list di Python sering digunakan untuk tujuan ini, dengan perbedaan utama bahwa array memiliki ukuran tetap sedangkan list dapat berubah ukuran. Struktur array di tingkat perangkat keras umumnya berbentuk array satu dimensi dengan elemen yang disimpan secara bersebelahan dalam memori.

Sementara itu, linked list adalah struktur data yang terdiri dari elemen-elemen yang disebut node, di mana setiap node menyimpan data dan pointer ke node berikutnya. Linked list memungkinkan alokasi memori dinamis dan efisien, memfasilitasi penambahan atau pengurangan elemen secara fleksibel. Dalam linked list, ada dua metode umum untuk mengelola elemen: LIFO (Last In First Out) dan FIFO (First In First Out). Node dalam linked list diimplementasikan dengan kelas yang memiliki atribut untuk data dan pointer ke node berikutnya, memungkinkan berbagai operasi seperti penyisipan, penghapusan, dan traversing elemen. Struktur ini memberikan fleksibilitas lebih besar dibandingkan array, meskipun akses data memerlukan waktu lebih lama karena harus melintasi setiap node dari awal hingga akhir.

E. Tugas

1. Jelasakan Menurut Anda Perbedaan Array Dan linked list!
2. Jelaskan 3 Kelebihan Dan Kekurangan Suatu Program Array!
3. Jelaskan bagaian dan operasi-operasi pada liked List!
4. Buatlah contoh array dalam pemrograman Python!
5. Buatlah lisked list dengan pemrograman python!

F. Pustaka

Amanulhaq, A. A. (2021). Implementasi Algoritma Image Hashing dan Hamming Distance untuk Deteksi Kemiripan Gambar. Repository ITS.

Agarwal, B., & Agarwal, B. (2018). Hands-On Data Structures and Algorithms with Python. London: Packt Publishing.

Eriana, E. S., & Zain, A. (2021). Praktikum Algoritma dan Pemrograman. Tangerang Selatan: Unpam Press.

Jodi, U. R. (2020). Algoritma dan Struktur Data. [Publisher Information].

Katahman, M. A., & Fathurrahman, M. (2021). Pembangunan Aplikasi Realiti Terimbuh untuk Pengenalan Struktur Data. Information Technology and Computer Science.

Nasrullah, A. H. (2021). Implementasi Algoritma Decision Tree untuk Klasifikasi Produk Laris. Jurnal Ilmiah Ilmu Komputer I.

Qi, P., Zhang, Y., Bolton, J., & Manning, C. D. (2020). Stanza: A Python Natural Language Processing Toolkit for Many Human Languages. [Conference or Journal Name].

Sianipar, R. H. (2013). Pemrograman & Struktur Data C: Belajar dari Contoh untuk Programmer Pemula Maupun Berpengalaman. Penerbit Informatika.

Thanaki, J. (2017). Python Natural Language Processing. Mumbai: Packt Publishing.

Zein, A. (2018). Pendeteksian Kantuk Secara Real Time Menggunakan Pustaka OpenCV dan Dlib Python. Sainstech: Jurnal Penelitian dan Pengkajian Sains dan Teknologi.

Modul 3: Algoritma Percabangan

Modul ini dirancang sebagai panduan komprehensif untuk memperkenalkan dan mengajarkan konsep algoritma percabangan dalam dunia pemrograman, yang merupakan salah satu dasar penting dalam pengambilan keputusan pada pengembangan perangkat lunak. Modul ini memiliki relevansi tinggi bagi mahasiswa, terutama dalam memahami bagaimana program dapat menentukan jalur eksekusi berdasarkan kondisi tertentu, sehingga memungkinkan pengembangan aplikasi yang lebih dinamis dan responsif terhadap berbagai skenario yang mungkin terjadi. Dengan mengikuti modul ini, mahasiswa diharapkan mampu memahami dan mengimplementasikan berbagai bentuk percabangan, seperti if, if-else, if-elif-else, dan if bersarang, dalam bahasa pemrograman Python. Selain itu, mahasiswa juga akan belajar bagaimana cara membuat program yang dapat menangani kondisi kompleks dengan lebih efisien, serta bagaimana menyusun logika program yang memungkinkan penanganan berbagai kondisi secara efektif.

A. Penyajian Materi

1. Pengertian Algoritma Percabangan

Dalam dunia pemrograman, kita seringkali dihadapkan pada berbagai jenis persoalan yang memerlukan analisis lebih mendalam, terutama ketika persoalan tersebut tidak sesederhana menjalankan serangkaian instruksi secara berurutan. Pada persoalan yang lebih kompleks, seringkali kita harus mempertimbangkan berbagai kemungkinan yang mungkin terjadi. Untuk setiap kemungkinan ini, terdapat kondisi tertentu yang harus dipenuhi, serta tindakan atau instruksi yang harus diambil ketika kondisi tersebut terpenuhi.

Proses ini disebut dengan istilah percabangan atau control flow. Percabangan memungkinkan kita untuk menentukan jalur mana yang akan diambil oleh program berdasarkan kondisi yang telah ditentukan. Dengan adanya percabangan, instruksi tidak lagi dieksekusi secara linier, tetapi bergantung pada kondisi yang ada.

Sebagai contoh, mari kita lihat bagaimana kita bisa menentukan apakah suatu bilangan termasuk bilangan genap atau bilangan ganjil. Algoritma untuk menentukan hal tersebut bisa dijelaskan sebagai berikut:

1. Mulai.
2. Masukkan suatu bilangan, misalnya bilangan Y.
3. Jika bilangan Y habis dibagi dua, lanjutkan ke perintah keempat. Jika tidak, lanjutkan ke perintah kelima.
4. Tulis “Y adalah bilangan genap”. Lanjutkan ke perintah keenam.
5. Tulis “Y adalah bilangan ganjil”.
6. Selesai.

Pada algoritma di atas, kita dapat melihat bahwa setelah perintah ketiga, terdapat dua kemungkinan jalur yang dapat diambil oleh program. Jika bilangan Y habis dibagi dua, maka program akan menjalankan perintah keempat, kemudian langsung melompat ke perintah keenam tanpa menjalankan perintah kelima. Sebaliknya, jika bilangan Y tidak habis dibagi dua, program akan langsung melompat ke perintah kelima dan tidak menjalankan perintah keempat. Pada akhirnya, kedua jalur tersebut akan bertemu kembali di perintah keenam, yang menandakan bahwa proses algoritma telah selesai.

Dalam bahasa pemrograman Python, konsep percabangan diimplementasikan menggunakan beberapa fungsi kondisi. Python memiliki beberapa bentuk percabangan yang memungkinkan kita untuk mengekspresikan berbagai kondisi dengan cara yang fleksibel dan mudah dipahami. Fungsi-fungsi percabangan dalam Python antara lain:

1. ‘if’ : Digunakan untuk menjalankan suatu blok kode jika kondisi tertentu terpenuhi.
2. ‘if, else’ : Digunakan ketika kita ingin mengeksekusi blok kode alternatif jika kondisi dalam if tidak terpenuhi.
3. ‘if, elif, else’ : Digunakan ketika kita memiliki beberapa kondisi yang berbeda dan ingin menjalankan blok kode yang sesuai dengan kondisi yang terpenuhi.
4. ‘if bersarang’ : Digunakan ketika kita perlu menulis percabangan di dalam percabangan lain, memberikan fleksibilitas lebih dalam pengambilan keputusan.

Dengan memahami berbagai jenis percabangan ini, kita dapat membuat program yang lebih dinamis dan responsif terhadap berbagai situasi yang mungkin terjadi selama eksekusi program. Hal ini sangat penting dalam pengembangan aplikasi yang harus menangani banyak skenario dan kondisi yang berbeda.

2. Percabangan ‘if’

If bersarang adalah struktur di mana satu pernyataan if ditempatkan di dalam pernyataan if lainnya. Ini berguna ketika kita memerlukan lebih dari satu kondisi untuk memutuskan jalur eksekusi yang tepat. Dalam kondisi yang lebih kompleks, kita mungkin perlu menggunakan beberapa lapisan if di dalam if.

Sintaks Dasar:

```
if kondisi:
    # Blok kode yang akan dijalankan jika kondisi bernilai True
```

- Kondisi adalah sebuah ekspresi yang dievaluasi menjadi True atau False. Jika kondisi tersebut benar (True), maka blok kode yang berada di bawah if akan dieksekusi.
- Contoh sederhana dari kondisi adalah perbandingan, seperti ‘x > 0’, ‘y == 10’, atau ‘a != b’.

Contoh:

```python
bilangan = 10

if bilangan > 0:
    print(f"{bilangan} adalah bilangan positif")
```

Pada contoh di atas, jika 'bilangan' lebih besar dari 0, maka program akan mencetak bahwa 'bilangan' tersebut adalah bilangan positif. Jika 'bilangan' tidak lebih besar dari 0 (misalnya negatif atau nol), maka tidak ada yang dicetak, dan program akan melanjutkan ke instruksi berikutnya.

3. Percabangan 'if-else'

Percabangan if-else digunakan ketika kita ingin menentukan dua jalur yang berbeda berdasarkan kondisi yang diberikan. Jika kondisi tersebut benar, blok if akan dieksekusi; jika kondisi tersebut salah, blok else akan dieksekusi.

Sintaks Dasar:

```python
if kondisi:
    # Blok kode yang akan dijalankan jika kondisi bernilai True
else:
    # Blok kode yang akan dijalankan jika kondisi bernilai False
```

Pada dasarnya, struktur ini memberikan alternatif aksi jika kondisi tidak terpenuhi.

Contoh:

```python
bilangan = -5

if bilangan > 0:
    print(f"{bilangan} adalah bilangan positif")
else:
    print(f"{bilangan} adalah bilangan negatif atau nol")
```

Pada contoh ini, jika 'bilangan' lebih besar dari 0, program akan mencetak bahwa 'bilangan' adalah positif. Namun, jika 'bilangan' tidak lebih besar dari 0, maka blok 'else' akan dieksekusi dan mencetak bahwa bilangan tersebut negatif atau nol.

4. Percabangan 'if-elif-else'

Struktur if-elif-else digunakan untuk memeriksa beberapa kondisi yang berbeda secara berurutan. Ini memungkinkan kita untuk membuat program yang

lebih fleksibel, di mana kita bisa memiliki lebih dari dua jalur eksekusi berdasarkan berbagai kondisi.

Sintaks Dasar:

```
if kondisi_1:
    # Blok kode jika kondisi_1 bernilai True
elif kondisi_2:
    # Blok kode jika kondisi_1 False dan kondisi_2 True
elif kondisi_3:
    # Blok kode jika kondisi_2 False dan kondisi_3 True
else:
    # Blok kode jika semua kondisi False
```

'elif' adalah singkatan dari "else if", yang memungkinkan kita untuk menambahkan kondisi tambahan setelah kondisi pertama (if).

Contoh:

```
nilai = 85

if nilai >= 90:
    print("Grade: A")
elif nilai >= 80:
    print("Grade: B")
elif nilai >= 70:
    print("Grade: C")
else:
    print("Grade: D atau E")
```

Pada contoh di atas, program memeriksa beberapa kondisi secara berurutan. Jika 'nilai' lebih besar atau sama dengan 90, program mencetak "Grade: A". Jika 'nilai' tidak lebih besar atau sama dengan 90, tetapi lebih besar atau sama dengan 80, maka program mencetak "Grade: B", dan seterusnya. Jika semua kondisi 'if' dan 'elif' salah, maka blok 'else' yang dijalankan.

5. Percabangan 'if bersarang'/Nested if

'If bersarang' atau nested if adalah struktur di mana satu pernyataan 'if' ditempatkan di dalam pernyataan 'if' lainnya. Ini berguna ketika kita memerlukan lebih dari satu kondisi untuk memutuskan jalur eksekusi yang tepat. Dalam kondisi yang lebih kompleks, kita mungkin perlu menggunakan beberapa lapisan 'if' di dalam 'if'.

Sintaks dasar:

```
if kondisi_1:
    if kondisi_2:
        # Blok kode jika kondisi_1 dan kondisi_2 True
    else:
        # Blok kode jika kondisi_1 True tapi kondisi_2 False
else:
    # Blok kode jika kondisi_1 Fals
```

Struktur ini memungkinkan kita untuk menciptakan logika yang lebih kompleks dan bertingkat.

Contoh:

```
umur = 20
punya_ktp = True

if umur >= 17:
    if punya_ktp:
        print("Anda bisa membuat SIM")
    else:
        print("Anda perlu mengurus KTP terlebih dahulu")
else:
    print("Anda belum cukup umur untuk membuat SIM")
```

Dalam contoh ini, pertama-tama program memeriksa apakah ‘`umur`’ lebih besar atau sama dengan 17. Jika benar, program kemudian memeriksa apakah pengguna memiliki KTP (‘`punya_ktp`’). Jika kedua kondisi ini benar, program mencetak "Anda bisa membuat SIM". Jika salah satu kondisi tidak terpenuhi, maka program memberikan pesan yang sesuai.

B. Latihan/contoh

Latihan 1 :

Buatlah program yang meminta pengguna memasukkan sebuah bilangan, kemudian tentukan apakah bilangan tersebut positif, negatif, atau nol menggunakan percabangan ‘`if-elif-else`’.

Cara dan Petunjuk:

1. Minta pengguna memasukkan bilangan.
2. Gunakan percabangan if-elif-else untuk memeriksa apakah bilangan tersebut lebih besar dari nol (positif), kurang dari nol (negatif), atau sama dengan nol.
3. Tampilkan hasilnya kepada pengguna.

Contoh Penyelesaian

```python
bilangan = int(input("Masukkan bilangan: "))

if bilangan > 0:
    print(f"{bilangan} adalah bilangan positif")
elif bilangan < 0:
    print(f"{bilangan} adalah bilangan negatif")
else:
    print(f"{bilangan} adalah nol")
```

Penjelasan:

- Program memeriksa apakah bilangan lebih besar dari nol untuk menentukan bilangan positif.
- Jika tidak, program memeriksa apakah bilangan kurang dari nol untuk menentukan bilangan negatif.
- Jika kedua kondisi tersebut salah, bilangan dianggap nol.

Hasil Output:

Output dari kode di atas akan tergantung pada nilai bilangan yang dimasukkan oleh pengguna. Berikut adalah beberapa contoh hasil output berdasarkan berbagai input:

1. Jika pengguna memasukkan bilangan positif:
   - Input: 5
   - Output: 5 adalah bilangan positif
2. Jika pengguna memasukkan bilangan negatif:
   - Input: -3
   - Output: -3 adalah bilangan negatif
3. Jika pengguna memasukkan nol:
   - Input: 0
   - Output: 0 adalah nol

Program ini mengevaluasi bilangan yang dimasukkan dan mencetak apakah bilangan tersebut positif, negatif, atau nol.

Latihan 2 :

Buatlah program yang meminta pengguna memasukkan nilai suhu dalam Celsius dan kemudian menentukan kategori suhu tersebut: dingin, normal, atau panas.

Cara dan Petunjuk:

1. Minta pengguna memasukkan nilai suhu dalam Celsius.
2. Gunakan percabangan if-elif-else untuk mengkategorikan suhu:
   - if: suhu < 20, kategori "dingin"

- elif: 20 <= suhu <= 30, kategori "normal"
- else: suhu > 30, kategori "panas"

3. Tampilkan kategori suhu kepada pengguna.

Contoh Penyelesaian:

```
suhu = int(input("Masukkan suhu dalam Celsius: "))

if suhu < 20:
    print("Suhu dingin")
elif 20 <= suhu <= 30:
    print("Suhu normal")
else:
    print("Suhu panas")
```

Penjelasan:

- Program mengevaluasi suhu yang dimasukkan pengguna untuk menentukan apakah suhu tersebut tergolong dingin, normal, atau panas.
- Kategori suhu kemudian ditampilkan berdasarkan kondisi yang terpenuhi.

Hasil Output:

Output dari kode di atas akan tergantung pada nilai suhu yang dimasukkan oleh pengguna. Berikut adalah beberapa contoh hasil output berdasarkan berbagai input:

1. Jika pengguna memasukkan suhu kurang dari 20 derajat Celsius:
   - Input: 15
   - Output: Suhu dingin
2. Jika pengguna memasukkan suhu antara 20 dan 30 derajat Celsius:
   - Input: 25
   - Output: Suhu normal
3. Jika pengguna memasukkan suhu lebih dari 30 derajat Celsius:
   - Input: 35
   - Output: Suhu panas

Program ini mengevaluasi nilai suhu yang dimasukkan dan menentukan apakah suhu tersebut tergolong dingin, normal, atau panas.

C. Rangkuman

Algoritma percabangan adalah proses di mana program dapat memilih jalur eksekusi berdasarkan kondisi tertentu. Dalam Python, percabangan diimplementasikan menggunakan struktur ‘if’, ‘if-else’, ‘if-elif-else’, dan ‘if bersarang’, yang memungkinkan

program merespons berbagai situasi berbeda. Struktur ini membantu dalam pengambilan keputusan yang lebih kompleks, seperti menentukan apakah suatu bilangan genap atau ganjil, atau memberikan penilaian berdasarkan nilai. Dengan memahami dan menggunakan percabangan, kita dapat menciptakan program yang lebih dinamis dan fleksibel.

D. Tugas

1. Jelaskan dengan kata-kata Anda sendiri apa yang dimaksud dengan percabangan dalam algoritma dan mengapa percabangan ini penting dalam pemrograman. Sertakan contoh sederhana yang menggambarkan penggunaan percabangan.
2. Bandingkan antara struktur percabangan '`if`, `if-else`', dan '`if-elif-else`'. Dalam situasi apa masing-masing struktur ini lebih cocok digunakan? Berikan contoh situasi untuk setiap jenis percabangan.
3. Buatlah program yang meminta pengguna memasukkan nilai ujian dan menampilkan grade berdasarkan nilai tersebut menggunakan percabangan if-elif-else. Gunakan rentang nilai berikut:
   - A: 90 - 100
   - B: 80 - 89
   - C: 70 - 79
   - D: 60 - 69
   - E: < 60
4. Buatlah program yang meminta pengguna memasukkan umur dan status kepemilikan KTP. Program kemudian harus memeriksa apakah pengguna berusia di atas 17 tahun dan memiliki KTP. Jika iya, tampilkan pesan bahwa mereka bisa membuat SIM; jika tidak, tampilkan pesan yang sesuai.

E. Pustaka

Jubilee Enterprise. 2019. Python untuk Programer Pemula.

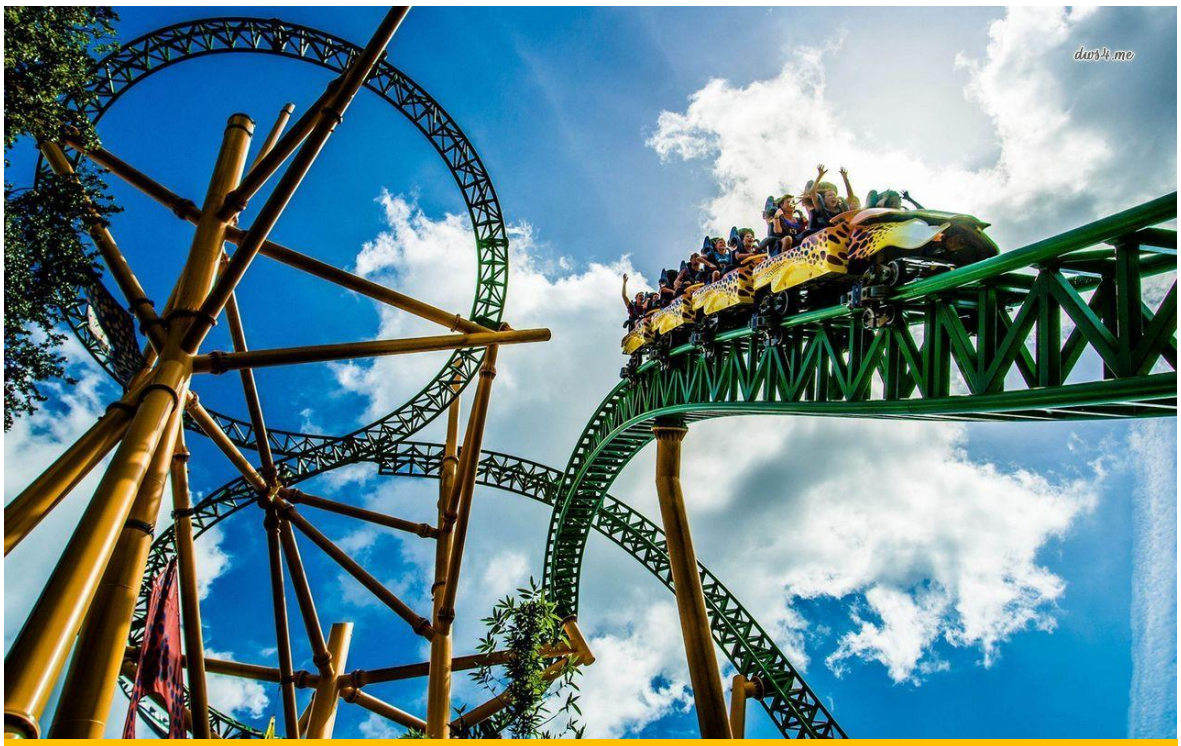
Pythonindo. 2019. Diakses pada 28 Juni 2020, dari https://www.pythonindo.com/program-pertama-dengan-python/.

Rafiqi, Aufa. 2019. Belajar Algoritma Pemrograman dengan Python.

Swastika, W. 2018. Belajar Algoritma Pemrograman Dengan Menggunakan Python.

Tutorial Dasar Python untuk Pemula. Diakses pada 5 Juli 2021, dari https://www.petanikode.com/tutorial/python/.

Modul 4: Perulangan (Looping)



Modul ini dirancang sebagai panduan komprehensif mengenai algoritma perulangan dalam pemrograman, menawarkan pemahaman mendalam tentang konsep "looping" yang memungkinkan eksekusi kode secara berulang berdasarkan kondisi tertentu. Modul ini memiliki relevansi besar dalam meningkatkan efisiensi dan kesederhanaan kode, memungkinkan otomatisasi tugas repetitif yang kompleks tanpa perlu menulis kode berulang kali. Dengan mengikuti modul ini, mahasiswa diharapkan mampu memahami dan menerapkan berbagai jenis perulangan seperti ‘`for`’, ‘`while`’, dan ‘`nested loop`’, serta mengoptimalkan perulangan untuk meningkatkan performa program mereka. Dilengkapi dengan contoh praktis dan teknik optimasi, modul ini memberikan fondasi yang kuat untuk menyelesaikan berbagai masalah pemrograman secara efektif dan efisien.

A. Penyajian Materi

1. Pengertian Algoritma Perulangan

Algoritma perulangan, atau yang sering disebut dengan "looping," adalah sebuah konsep fundamental dalam pemrograman yang memungkinkan eksekusi suatu blok kode berulang kali berdasarkan kondisi tertentu. Algoritma perulangan digunakan untuk mengotomatisasi tugas-tugas yang berulang, sehingga programmer tidak perlu menulis kode yang sama berulang kali. Perulangan ini sangat berguna dalam situasi di mana suatu aksi perlu dilakukan secara berkali-kali, misalnya untuk memproses setiap elemen dalam sebuah kumpulan data, atau untuk mengulang suatu proses hingga tercapai kondisi tertentu.

Perulangan memungkinkan sebuah program untuk mengeksekusi serangkaian instruksi berulang kali hingga kondisi yang ditentukan terpenuhi. Dengan demikian, program yang menggunakan perulangan dapat menjadi lebih singkat, lebih mudah dipahami, dan lebih mudah di-maintain. Contohnya, jika Anda ingin mencetak angka 1 hingga 1000, tanpa perulangan Anda harus menulis 1000 baris kode. Namun, dengan perulangan, Anda hanya perlu menulis beberapa baris kode saja. Python menyediakan tiga cara utama untuk melakukan perulangan, yaitu:

- For
- While
- Nested Loop (Perulangan Bersarang)

Ketigangan memiliki perbedaan pada segi Struktur dan Penggunaan, Kendali Iterasi serta Efisiensi dan Kinerja

a. Struktur dan Penggunaan:

- For Loop: Digunakan ketika jumlah iterasi telah diketahui sebelumnya. For loop sangat efektif untuk mengiterasi item dalam sebuah koleksi seperti list, tuple, atau range.
- While Loop: Digunakan ketika jumlah iterasi tidak diketahui dan iterasi harus terus berjalan hingga kondisi tertentu terpenuhi. Perulangan ini lebih fleksibel karena kondisi dievaluasi di setiap iterasi.

- Nested Loop: Merupakan perulangan di dalam perulangan lain. Struktur ini digunakan untuk situasi di mana diperlukan iterasi ganda, seperti dalam pemrosesan matriks atau array multidimensi.

b. Kendali Iterasi

- For Loop: Setiap iterasi pada for loop biasanya dikontrol oleh nilai yang diambil dari sebuah koleksi atau range, sehingga iterasi berlanjut secara otomatis hingga semua elemen telah diakses.
- While Loop: Kontrol iterasi pada while loop sepenuhnya ditentukan oleh kondisi yang diberikan. Iterasi dapat terus berlangsung selama kondisi bernilai True, sehingga memungkinkan kontrol yang lebih dinamis.
- Nested Loop: Pada nested loop, kontrol iterasi bergantung pada loop luar dan loop dalam. Loop dalam biasanya selesai sepenuhnya sebelum loop luar melanjutkan ke iterasi berikutnya.

c. Efisiensi dan Kinerja

- For Loop: Umumnya lebih efisien dan lebih mudah diprediksi dalam situasi di mana iterasi pada koleksi data tertentu diperlukan.
- While Loop: Meskipun lebih fleksibel, while loop bisa menjadi tidak efisien jika kondisi tidak dikelola dengan baik, terutama jika tidak ada mekanisme penghentian yang jelas.
- Nested Loop: Cenderung memerlukan lebih banyak sumber daya dan bisa mengakibatkan penurunan kinerja jika digunakan secara berlebihan, terutama ketika tingkat nesting terlalu dalam.

2. Jenis-jenis Perulangan

a. Perulangan For

Perulangan for digunakan untuk iterasi yang telah ditentukan jumlahnya, misalnya iterasi melalui elemen-elemen dalam sebuah koleksi seperti list, tuple, atau range. Ini adalah perulangan yang umum digunakan ketika kita mengetahui dengan pasti jumlah langkah yang diperlukan. Konsep ini memungkinkan kita untuk mengotomatisasi tugas berulang dengan cara yang mudah dan efisien.

Sintaks Dasar:

```
for variabel in koleksi:
    # blok kode yang akan dieksekusi berulang kali
```

Penjelasan:

- 'Variabel': Variabel ini akan mengambil nilai dari setiap elemen dalam koleksi pada setiap iterasi.
- 'koleksi': Ini bisa berupa 'list', 'tuple', 'string', atau objek lain yang mendukung iterasi.

Contoh Implementasi:

```
for i in range(5):
    print("Iterasi ke-", i)
```

Penjelasan Contoh:

Pada contoh di atas, '`range(5')` menghasilkan urutan angka dari 0 hingga 4. Setiap angka dalam urutan tersebut diambil oleh variabel '`i'`, dan blok kode di dalam '`for'` dieksekusi lima kali, mencetak "Iterasi ke-" diikuti oleh nilai '`i'`.

Penggunaan Lain:

Perulangan '`for'` juga bisa digunakan untuk mengakses elemen dalam koleksi yang lebih kompleks seperti '`list'` yang berisi '`tuple'`:

```
data = [(1, 'A'), (2, 'B'), (3, 'C')]
for (angka, huruf) in data:
    print(f"Angka: {angka}, Huruf:
{huruf}")
```

Dalam kasus ini, setiap '`tuple'` dalam '`data'` dipisahkan ke dalam dua variabel '`angka'` dan '`huruf'`, dan kode di dalam perulangan '`for'` akan mencetak pasangan ini.

b. Perulangan While

While loop digunakan ketika kondisi tertentu harus terus diperiksa, dan perulangan akan terus berjalan selama kondisi tersebut benar (True). Ini sangat berguna dalam situasi di mana kita tidak tahu sebelumnya berapa kali loop harus dijalankan, tetapi kita tahu kondisi yang harus dipenuhi untuk menghentikan loop tersebut.

Sintaks Dasar:

```
while kondisi:
    # blok kode yang akan dieksekusi selama kondisi True
```

Penjelasan:

'kondisi': Kondisi ini dievaluasi sebelum setiap iterasi. Jika bernilai 'True', maka blok kode di dalam 'while' akan dieksekusi; jika 'False', perulangan berhenti.

Contoh Implementasi :

```
x = 0
while x < 5:
    print("x bernilai", x)
    x += 1
```

Penjelasan Contoh:

Di sini, nilai x awalnya adalah 0. Selama x kurang dari 5, perulangan akan terus berjalan, mencetak nilai x dan menambah x sebesar 1 pada setiap iterasi. Ketika x mencapai 5, kondisi x < 5 menjadi False, dan perulangan berhenti.

Penggunaan dalam Kondisi yang Dinamis:

Perulangan while sangat berguna ketika kita tidak mengetahui jumlah iterasi yang diperlukan sebelumnya. Misalnya, menghitung nilai faktorial hingga mencapai angka tertentu:

```
n = 1
faktorial = 1
while faktorial < 1000:
    n += 1
    faktorial *= n
print(f"Faktorial {n} adalah {faktorial}, yang pertama kali melebihi 1000.")
```

Pada contoh di atas, perulangan berlanjut hingga nilai faktorial lebih besar dari 1000.

c. Perulangan Bersarang (Nested Loop)

Perulangan bersarang terjadi ketika satu perulangan ditempatkan di dalam perulangan lainnya. Ini sering digunakan untuk bekerja dengan struktur data multidimensi, seperti matriks. Dengan menggunakan perulangan

bersarang, kita bisa mengelola dan memanipulasi data dalam struktur yang lebih kompleks secara lebih efisien.

Sintaks Dasar:

```
for variabel1 in koleksi1:
    for variabel2 in koleksi2:
        # blok kode yang akan dieksekusi
```

Penjelasan:

Perulangan luar (‘for variabel1 in koleksi1’) memulai iterasi pertama, dan untuk setiap iterasi di perulangan luar, perulangan dalam (‘for variabel2 in koleksi2’) akan dieksekusi sepenuhnya.

Contoh Implementasi:

```
for i in range(3):
    for j in range(2):
        print(f"i: {i}, j: {j}")
```

Penjelasan Contoh:

Dalam contoh ini, untuk setiap nilai ‘i’ dalam ‘range(3)’, yang menghasilkan 0 hingga 2, perulangan dalam akan berjalan untuk nilai ‘j’ dari ‘range(2)’ yang menghasilkan 0 hingga 1. Output akan mencetak kombinasi nilai ‘i’ dan ‘j’.

Penggunaan untuk Matriks:

Perulangan bersarang sangat berguna dalam pengoperasian matriks, misalnya, untuk menjumlahkan dua matriks:

```
matriks1 = [[1, 2], [3, 4]]
matriks2 = [[5, 6], [7, 8]]
hasil = [[0, 0], [0, 0]]

for i in range(len(matriks1)):
    for j in range(len(matriks1[0])):
        hasil[i][j] = matriks1[i][j] + matriks2[i][j]
print("Hasil penjumlahan matriks:", hasil)
```

Pada contoh ini, kita menambahkan dua matriks 2x2 menggunakan perulangan bersarang, di mana hasilnya disimpan dalam matriks ‘`hasil`’.

d. Menggunakan Break dan Continue dalam Perulangan

Kedua perintah ini digunakan untuk mengontrol aliran perulangan.

- Break: Digunakan untuk keluar dari perulangan sepenuhnya, terlepas dari kondisi yang mengontrol perulangan.
- Continue: Melompat ke iterasi berikutnya dari perulangan, melewati kode yang tersisa dalam iterasi saat ini.

Contoh Break:

```
for i in range(10):
    if i == 5:
        break
    print(i)
```

Penjelasan Contoh:

Pada contoh ini, perulangan berhenti ketika i mencapai 5. Hanya nilai dari 0 hingga 4 yang akan dicetak.

Contoh Continue:

```
for i in range(10):
    if i % 2 == 0:
        continue
    print(i)
```

Penjelasan Contoh:

Dalam contoh ini, perulangan akan melompati semua angka genap karena perintah continue membuat program melompati iterasi saat i bernilai genap. Hanya angka ganjil yang akan dicetak.

3. Optimasi Perulangan

Mengoptimalkan perulangan sangat penting untuk meningkatkan efisiensi program, terutama saat menangani data dalam jumlah besar atau menjalankan perulangan dalam waktu yang lama. Beberapa teknik optimasi meliputi:

- Menghindari Perhitungan Ulang: Simpan hasil perhitungan yang berulang kali digunakan dalam variabel. Ini akan mengurangi overhead dan meningkatkan kecepatan eksekusi program.
- Menggunakan Loop Unrolling: Mempercepat eksekusi dengan mengurangi overhead loop. Teknik ini melibatkan penulisan kode

perulangan secara eksplisit untuk beberapa iterasi sekaligus, yang dapat mengurangi jumlah perulangan yang sebenarnya dilakukan oleh CPU.

- Pemilihan Struktur Data yang Tepat: Memilih struktur data yang lebih efisien untuk iterasi dapat meningkatkan kinerja program. Misalnya, menggunakan set alih-alih list ketika tidak diperlukan urutan khusus, karena set menawarkan pencarian yang lebih cepat.

B. Latihan/contoh

1. **Latihan Soal 1 : Menghitung Jumlah Bilangan**

   Problem Statement: Hitung jumlah semua bilangan genap dari 1 hingga 100.

   Implementasi:

```
jumlah = 0
for i in range(1, 101):
    if i % 2 == 0:
        jumlah += i
print("Jumlah bilangan genap dari 1 hingga 100 adalah", jumlah)
```

   Penjelasan Implementasi: Pada contoh ini, kita menggunakan perulangan `for` untuk iterasi dari 1 hingga 100. Pada setiap iterasi, kita memeriksa apakah angka tersebut genap (`i % 2 == 0`). Jika genap, kita menambahkan angka tersebut ke variabel `jumlah`. Setelah perulangan selesai, jumlah total bilangan genap dicetak.

2. **Latihan 2 : Mencetak Pola Bintang**

   Problem Statement: Mencetak pola bintang seperti segitiga terbalik.

   Implementasi:

```
baris = 5
for i in range(baris, 0, -1):
    for j in range(i):
        print("*", end="")
    print("")
```

   Penjelasan Implementasi:

   Di sini, kita menggunakan perulangan bersarang untuk mencetak pola bintang. Perulangan luar mengontrol jumlah baris (`baris`), dan perulangan dalam mencetak jumlah bintang yang sesuai di setiap baris. `end=""` digunakan untuk

memastikan bahwa bintang-bintang dicetak pada baris yang sama, dan `print("")` digunakan untuk berpindah ke baris berikutnya setelah setiap baris selesai dicetak.

Variasi Implementasi: Pola bintang bisa dimodifikasi untuk mencetak berbagai bentuk, seperti segitiga atau piramida, dengan mengubah logika dalam perulangan.

3. **Latihan 3: Menghitung Faktorial**

Problem Statement: Menghitung faktorial dari sebuah angka yang diberikan.

Implementasi:

```
n = 5
faktorial = 1
for i in range(1, n + 1):
    faktorial *= i
print(f"Faktorial dari {n} adalah {faktorial}")
```

Penjelasan Implementasi:

Dalam contoh ini, faktorial dari sebuah angka n dihitung dengan mengalikan semua angka dari 1 hingga n. Prosesnya menggunakan perulangan for:

- Inisialisasi: Variabel faktorial diinisialisasi dengan nilai 1, karena faktorial adalah hasil perkalian dan dimulai dari 1.
- Perulangan: Perulangan dimulai dari angka 1 hingga n. Pada setiap iterasi, nilai i akan dikalikan dengan faktorial, dan hasilnya disimpan kembali ke dalam variabel faktorial.
- Output: Setelah perulangan selesai, hasil faktorial dicetak menggunakan print.

Contoh Penghitungan:

Untuk n = 5, perhitungan faktorial berjalan sebagai berikut:

- Iterasi 1: faktorial = 1 * 1 = 1
- Iterasi 2: faktorial = 1 * 2 = 2
- Iterasi 3: faktorial = 2 * 3 = 6
- Iterasi 4: faktorial = 6 * 4 = 24
- Iterasi 5: faktorial = 24 * 5 = 120

Jadi, hasil faktorial dari 5 adalah 120. Program akan mencetak: "Faktorial dari 5 adalah 120".

C. Rangkuman

Dalam modul ini, kita telah membahas perulangan sebagai alat kontrol aliran program yang penting dalam pemrograman. Berbagai jenis perulangan seperti for, while, dan perulangan bersarang telah dijelaskan secara detail, serta bagaimana mengoptimalkan perulangan untuk meningkatkan efisiensi program. Contoh studi kasus memberikan gambaran praktis tentang bagaimana konsep ini diterapkan dalam situasi nyata, seperti menghitung jumlah bilangan genap, mencetak pola bintang, dan menghitung faktorial. Melalui penerapan praktis ini, diharapkan pemahaman tentang penggunaan perulangan dapat diperkuat, sehingga Anda dapat mengimplementasikannya dengan lebih efektif dalam proyek-proyek pemrograman Anda.

D. Tugas

1. Jelaskan perbedaan antara perulangan for dan while.
2. Apa yang dimaksud dengan perulangan bersarang? Berikan contoh kasus nyata di mana perulangan bersarang diperlukan.
3. Bagaimana cara mengoptimalkan perulangan untuk menangani data dalam jumlah besar?
4. Buat program yang mencetak bilangan prima antara 1 hingga 50.
5. Implementasikan program yang mencetak tabel perkalian menggunakan perulangan.

E. Pustaka

Munir, Rinaldi. 2005. Algoritma dan Pemrograman dalam Bahasa Pascal dan C, Buku 1, Edisi Ketiga, Penerbit Informatika Bandung.

Purnomo, H. (2020). Belajar Python untuk Pemula. Bandung: Informatika.

Pythonindo.2019. Diakses pada 30 Juli 2024, dari https://www.pythonindo.com/program-pertama-dengan-python/.

Suryadi, S. (2021). Pemrograman Dasar dengan Python. Bandung: CV. Alfabeta.

Susetyo, R. (2020). Python: Pemrograman untuk Semua Kalangan. Jakarta: Elex Media Komputindo.

Modul 5: Fungsi dan Prosedur

Modul ini dirancang sebagai panduan komprehensif bagi mahasiswa untuk memahami konsep dasar dan aplikasi praktis fungsi dan prosedur dalam pemrograman Python, yang merupakan fondasi penting dalam pengembangan perangkat lunak. Modul ini memiliki relevansi yang sangat tinggi, karena fungsi dan prosedur adalah elemen kunci yang memungkinkan kode menjadi lebih modular, efisien, dan mudah dikelola, yang sangat dibutuhkan dalam dunia pemrograman profesional. Dengan mengikuti modul ini, mahasiswa diharapkan mampu memahami cara mendefinisikan, mengimplementasikan, dan memanfaatkan fungsi serta prosedur untuk memecahkan berbagai masalah pemrograman secara efektif, serta dapat menerapkan konsep ini dalam proyek dan tugas-tugas pemrograman mereka, sehingga memperkuat kemampuan teknis dan logika pemrograman yang mereka miliki.

A. Penyajian Materi

1. Pengertian Fungsi dan Prosedur

a. Fungsi

Fungsi dalam Python adalah blok kode terpisah yang dirancang untuk melakukan tugas spesifik dan dapat dipanggil dari bagian mana pun dalam program. Fungsi didefinisikan menggunakan kata kunci ‘def’ diikuti dengan nama fungsi dan daftar parameter di dalam tanda kurung. Setelah didefinisikan, fungsi dapat dipanggil dengan menggunakan namanya diikuti oleh tanda kurung yang berisi argumen yang sesuai dengan parameter.

Fungsi merupakan salah satu konsep penting dalam pemrograman karena memungkinkan pembagian kode menjadi bagian-bagian yang lebih kecil dan modular, yang dapat digunakan kembali (reusable). Ini sangat membantu dalam memecahkan masalah yang kompleks dengan membagi mereka menjadi sub-masalah yang lebih kecil dan lebih mudah dikelola.

b. Prosedur

Prosedur adalah jenis fungsi yang tidak mengembalikan nilai apa pun kepada pemanggilnya. Dalam bahasa pemrograman lain, prosedur sering disebut sebagai "fungsi void". Dalam Python, prosedur dapat dianggap sebagai fungsi yang tidak menggunakan pernyataan return atau yang return-nya adalah None.

Prosedur biasanya digunakan untuk menjalankan serangkaian instruksi atau operasi, seperti mencetak hasil ke layar, memodifikasi variabel global, atau melakukan tugas lain yang tidak memerlukan pengembalian nilai.

c. Perbedaan utama antara Fungsi dan Prosedur

- Pengembalian Nilai: Fungsi biasanya mengembalikan nilai dengan menggunakan pernyataan return, sedangkan prosedur tidak mengembalikan nilai apa pun (atau mengembalikan None secara implisit).
- Tujuan: Fungsi digunakan untuk menghitung dan mengembalikan nilai, sedangkan prosedur digunakan untuk melakukan tugas tertentu tanpa perlu mengembalikan nilai.
- Penggunaan: Fungsi lebih umum digunakan saat hasil dari operasi diperlukan untuk langkah berikutnya dalam program, sementara prosedur

lebih sering digunakan untuk operasi yang efek sampingnya adalah tujuan utamanya, seperti output ke layar.

d. Mengapa Fungsi dan Prosedur penting?

- Modularitas: Dengan menggunakan fungsi dan prosedur, kode dapat dibagi menjadi bagian-bagian kecil yang terisolasi, memungkinkan pengembang untuk fokus pada satu bagian kode pada suatu waktu.
- Reuse: Fungsi dan prosedur dapat dipanggil berulang kali, sehingga menghindari pengulangan kode (code duplication).
- Kemudahan Pemeliharaan: Karena fungsi dan prosedur dapat digunakan kembali, setiap perubahan yang diperlukan hanya perlu dilakukan di satu tempat (yaitu, dalam definisi fungsi atau prosedur), bukan di setiap tempat kode tersebut muncul.
- Keterbacaan Kode: Membagi program menjadi fungsi dan prosedur yang bermakna membuat kode lebih mudah dibaca dan dipahami, baik oleh penulis kode maupun orang lain yang mungkin perlu memelihara atau memperluas kode tersebut di masa mendatang.

2. Mendefinisikan Fungsi

Untuk mendefinisikan fungsi di Python, Anda harus menggunakan kata kunci def diikuti dengan nama fungsi yang Anda pilih. Nama fungsi harus deskriptif dan mengikuti aturan penamaan variabel, yaitu:

- Harus dimulai dengan huruf atau garis bawah (_).
- Tidak boleh mengandung spasi; gunakan underscore (_) untuk menggantikan spasi jika diperlukan.
- Tidak boleh berupa kata kunci yang sudah digunakan di Python (seperti if, else, for, dll).

Setelah nama fungsi, tambahkan tanda kurung yang mungkin berisi daftar parameter yang akan diterima oleh fungsi tersebut. Parameter ini adalah variabel input yang akan digunakan dalam fungsi. Jika fungsi tidak membutuhkan input, tanda kurung tetap diperlukan tetapi kosong. Kemudian, tambahkan tanda titik dua : di akhir baris pertama.

Blok kode yang ingin dijalankan dalam fungsi tersebut ditulis di baris berikutnya dengan indentasi yang benar. Di Python, indentasi yang umum

digunakan adalah empat spasi atau satu tab. Semua kode yang masuk dalam blok fungsi harus diindentasi pada level yang sama.

Sintaks dasar untuk mendefinisikan fungsi adalah sebagai berikut:

```
def nama_fungsi(parameter1, parameter2):
    # Blok kode yang akan dieksekusi
    return nilai  # Pernyataan return bersifat opsional
```

- ‘def’: Kata kunci yang digunakan untuk mendefinisikan fungsi.
- ‘nama_fungsi’: Nama fungsi yang Anda buat.
- ‘parameter1’, ‘parameter2’: Daftar parameter (input) yang diterima oleh fungsi.
- ‘:’ Tanda titik dua yang mengindikasikan dimulainya blok kode fungsi.
- # Blok kode: Kode yang akan dieksekusi ketika fungsi dipanggil.
- ‘return’: Pernyataan opsional yang digunakan untuk mengembalikan nilai dari fungsi.

Contoh Mendefinisikan Fungsi:

Mari kita lihat contoh sederhana dari fungsi yang menjumlahkan dua angka:

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

Dalam contoh di atas:

- ‘tambah’ adalah nama fungsi.
- ‘a’ dan ‘b’ adalah parameter yang diterima oleh fungsi.
- Fungsi ini mengembalikan hasil penjumlahan dari ‘a’ dan ‘b’ dengan menggunakan pernyataan return.

Cara menggunakan fungsi ini dalam kode Anda adalah dengan memanggilnya menggunakan nama fungsi diikuti oleh tanda kurung yang berisi argumen:

```
hasil = tambah(3, 5)
print(hasil)  # Output: 8
```

Di sini, fungsi tambah dipanggil dengan argumen 3 dan 5, dan hasilnya (yaitu 8) disimpan dalam variabel hasil, lalu dicetak ke layar.

3. Memanggil Fungsi

Memanggil fungsi adalah proses menggunakan fungsi yang telah Anda definisikan untuk melakukan tugas tertentu dalam program Anda. Ketika Anda memanggil sebuah fungsi, Python menjalankan blok kode di dalam fungsi tersebut.

Untuk memanggil fungsi, gunakan nama fungsi diikuti dengan tanda kurung yang berisi argumen yang diperlukan. Argumen ini adalah nilai yang Anda kirimkan ke fungsi untuk diproses. Jika fungsi tidak memerlukan argumen, Anda cukup menuliskan tanda kurung kosong setelah nama fungsi.

Contoh Memanggil Fungsi:

```
hasil = tambah(5, 3)
print(hasil)  # Output: 8
```

Pada contoh di atas:

- ‘tambah(5, 3)’ memanggil fungsi tambah dengan dua argumen, 5 dan 3.
- Fungsi tambah menjumlahkan kedua angka tersebut dan mengembalikan hasilnya, yang kemudian disimpan dalam variabel hasil.
- print(hasil) mencetak nilai dari variabel hasil, yaitu 8.

Fungsi juga bisa dipanggil di dalam fungsi lain, atau bahkan di dalam dirinya sendiri untuk membuat rekursi.

4. Parameter dan Argumen

Parameter dan argumen adalah konsep penting dalam pemrograman fungsi. Mereka memungkinkan Anda untuk mengirim data ke dalam fungsi dan menerima hasilnya.

- Parameter adalah variabel yang Anda tentukan di dalam tanda kurung saat mendefinisikan fungsi. Mereka bertindak sebagai placeholder untuk nilai yang akan Anda masukkan ke dalam fungsi.
- Argumen adalah nilai aktual yang Anda berikan kepada parameter ketika Anda memanggil fungsi. Mereka adalah data yang akan diproses oleh fungsi.

Python menawarkan fleksibilitas dalam menangani parameter dan argumen dengan berbagai fitur seperti parameter default, parameter variadic, dan parameter kunci.

Parameter Default:

Parameter default adalah parameter yang memiliki nilai awal yang ditentukan saat fungsi didefinisikan. Jika Anda tidak memberikan argumen untuk parameter ini ketika memanggil fungsi, maka nilai default akan digunakan.

```
def sapa(nama="Tamu"):
    print(f"Halo, {nama}")

sapa()          # Output: Halo, Tamu
sapa("Ali")     # Output: Halo, Ali
```

Dalam contoh ini:

- Fungsi sapa memiliki parameter nama dengan nilai default "Tamu".
- Ketika sapa() dipanggil tanpa argumen, fungsi menggunakan nilai default "Tamu".
- Ketika sapa("Ali") dipanggil, nilai default digantikan oleh argumen "Ali".

Parameter Variadic:

Parameter variadic memungkinkan Anda untuk mengirimkan sejumlah argumen yang tidak terbatas ke dalam fungsi. Ini dilakukan dengan menggunakan tanda bintang (*) di depan nama parameter.

```
def cetak(*args):
    for arg in args:
        print(arg)

cetak(1, 2, 3, 4)  # Output: 1 2 3 4
```

Pada contoh ini:

- *args menunjukkan bahwa fungsi cetak dapat menerima sejumlah argumen yang tidak terbatas.
- Fungsi kemudian mencetak setiap argumen yang diberikan satu per satu.

5. Prosedur dalam Python

Prosedur adalah konsep dalam pemrograman yang mengacu pada fungsi yang tidak mengembalikan nilai. Dalam Python, tidak ada jenis yang terpisah untuk prosedur; fungsi yang tidak menggunakan pernyataan `return` atau yang hanya mengembalikan `None` dapat dianggap sebagai prosedur.

Contoh Prosedur:

```python
def cetak_salam(nama):
    print(f"Selamat datang, {nama}")

cetak_salam("Budi")  # Output: Selamat datang, Budi
```

Dalam contoh ini:

- cetak_salam adalah sebuah prosedur karena tidak mengembalikan nilai apa pun.
- Prosedur ini hanya mencetak pesan ke layar.

6. Manfaat Fungsi dan Prosedur

Menggunakan fungsi dan prosedur dalam pemrograman menawarkan berbagai keuntungan yang membuat pengembangan perangkat lunak menjadi lebih efisien dan terorganisir.

a. Modularitas:

Fungsi dan prosedur memungkinkan Anda memecah program besar menjadi bagian-bagian kecil yang dapat dikelola. Setiap bagian berfokus pada tugas tertentu, membuat pengembangan lebih terstruktur.

b. Penggunaan Kembali:

Setelah didefinisikan, fungsi dapat digunakan kembali di berbagai bagian program atau bahkan di proyek lain. Ini mengurangi duplikasi kode dan meningkatkan efisiensi.

c. Pemeliharaan:

Kode yang terorganisir dalam fungsi lebih mudah dipelihara. Perubahan pada satu bagian kode tidak akan mempengaruhi bagian lain, selama fungsi tersebut digunakan dengan benar.

d. Abstraksi:

Fungsi dan prosedur memungkinkan Anda menyembunyikan detail implementasi kompleks dan hanya menampilkan antarmuka sederhana yang relevan. Ini membuat kode lebih mudah dipahami oleh orang lain atau oleh Anda sendiri di masa depan.

Dengan memahami dan menerapkan konsep-konsep ini, Anda dapat menulis kode Python yang lebih efisien, dapat digunakan kembali, dan mudah dipelihara, yang merupakan kunci untuk menjadi seorang pemrogram yang handal.

B. Latihan/contoh

Soal Latihan : Menggabungkan Konsep Fungsi dan Prosedur di Python

Problem Statement: Buatlah sebuah program Python yang terdiri dari beberapa fungsi. Program ini akan meminta pengguna untuk memasukkan dua bilangan, kemudian menggunakan fungsi untuk melakukan operasi penjumlahan, pengurangan, perkalian, dan pembagian.

Langkah-Langkah Penyelesaian:

1. **Mendefinisikan Fungsi untuk Operasi Aritmatika:**
   - Buat fungsi-fungsi terpisah untuk setiap operasi: '**tambah(a, b)'**, '**kurang(a, b)**', '**kali(a, b)'**, dan '**bagi(a, b)'**.
2. **Menggunakan Fungsi dalam Program Utama:**
   - Buat program utama yang meminta pengguna memasukkan dua bilangan, lalu memanggil setiap fungsi untuk melakukan operasi aritmatika dan menampilkan hasilnya.

Contoh Implementasi:

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

def kurang(a, b):
    return a - b

def kali(a, b):
    return a * b

def bagi(a, b):
    if b != 0:
        return a / b
    else:
        return "Pembagian dengan nol tidak diperbolehkan."

    # Blok utama program
a = float(input("Masukkan bilangan pertama: "))
b = float(input("Masukkan bilangan kedua: "))

print(f"Hasil penjumlahan: {tambah(a, b)}")
print(f"Hasil pengurangan: {kurang(a, b)}")
print(f"Hasil perkalian: {kali(a, b)}")
print(f"Hasil pembagian: {bagi(a, b)}")
```

Penjelasan Implementasi:

1. **Fungsi tambah(a, b), kurang(a, b), kali(a, b), dan bagi(a, b)**:
   - Masing-masing fungsi menerima dua parameter, a dan b, yang mewakili dua bilangan yang akan dioperasikan. Fungsi ini mengembalikan hasil dari operasi yang sesuai (penjumlahan, pengurangan, perkalian, atau pembagian).

2. **Blok Utama Program:**
   - Program meminta pengguna untuk memasukkan dua bilangan. Kemudian, fungsi-fungsi operasi aritmatika dipanggil untuk menghitung hasilnya dan menampilkan setiap hasil ke layar.

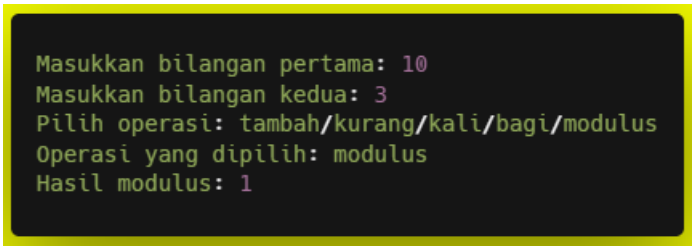
C. Rangkuman

Dalam modul ini, kita telah membahas perulangan serta fungsi dan prosedur sebagai elemen penting dalam pemrograman Python. Perulangan seperti for, while, dan perulangan bersarang dijelaskan untuk mengontrol aliran program dan mengoptimalkan efisiensi. Fungsi dan prosedur memungkinkan pembagian kode menjadi bagian-bagian kecil yang dapat digunakan kembali, meningkatkan keterbacaan, modularitas, dan memudahkan pemeliharaan kode. Dengan contoh-contoh praktis, seperti menghitung jumlah bilangan genap, mencetak pola bintang, menghitung faktorial, serta menghitung luas persegi panjang, pemahaman mengenai konsep ini diharapkan dapat diperkuat, memungkinkan Anda untuk menulis kode yang lebih bersih, terstruktur, dan efisien.

D. Tugas

1. Jelaskan perbedaan antara fungsi yang mengembalikan nilai dan fungsi yang tidak mengembalikan nilai (prosedur) dalam pemrograman Python.
2. Apa keuntungan dari menggunakan fungsi dalam pemrograman? Jelaskan bagaimana fungsi membantu dalam pengelolaan kode program yang kompleks.
3. Berikan contoh penggunaan parameter default dalam sebuah fungsi dan jelaskan situasi di mana ini sangat berguna.
4. **Modifikasi Program Latihan:** Tambahkan fungsi baru bernama modulus(a, b) yang mengembalikan sisa hasil bagi dari a dan b. Tambahkan juga fitur dalam program utama yang memungkinkan pengguna memilih operasi mana yang ingin dijalankan.

   Contoh Outputnya :



5. **Buat Program Baru:** Buatlah sebuah program Python yang meminta pengguna memasukkan sebuah kata, kemudian menggunakan fungsi untuk menghitung dan menampilkan jumlah huruf vokal dalam kata tersebut. Fungsi yang dibutuhkan adalah ‘hitung_vokal(kata)’.

   Contoh Outputnya:

```
Masukkan sebuah kata: 'Python'
Jumlah huruf vokal dalam kata 'Python' adalah: 1
```

E. Pustaka

Lutz, M. (2013). *Learning Python* (5th ed.). O'Reilly Media.

Purwanto, A. (2020). "Pemrograman Python Dasar: Fungsi dan Prosedur". *Jurnal Teknologi Informasi dan Ilmu Komputer*, 7(2), 102-110.

Rahman, R., & Lubis, M. (2017). *Pemrograman Python untuk Pemula*. Informatika.

Van Rossum, G., & Drake, F. L. (2009). *The Python Language Reference Manual*. Network Theory Ltd.

Zelle, J. M. (2017). *Python Programming: An Introduction to Computer Science* (3rd ed.). Franklin, Beedle & Associates Inc.

Modul 6: Konsep Rekursi

Modul ini dirancang sebagai panduan komprehensif untuk memahami konsep dasar rekursi dalam pemrograman, yang merupakan teknik penting di mana sebuah fungsi memanggil dirinya sendiri untuk menyelesaikan masalah yang dapat dipecah menjadi sub-masalah serupa. Modul ini memiliki relevansi tinggi bagi mahasiswa, terutama dalam mengembangkan kemampuan untuk memecahkan masalah kompleks melalui pendekatan yang lebih sederhana dan terstruktur. Dengan mengikuti modul ini, mahasiswa diharapkan mampu memahami konsep rekursi secara mendalam, mengimplementasikannya dalam berbagai kasus pemrograman, serta mengaplikasikan teknik ini untuk menyelesaikan masalah seperti faktorial, Fibonacci, dan perhitungan pangkat secara efisien.

A. Penyajian Materi

Rekursi adalah teknik dalam pemrograman di mana sebuah fungsi memanggil dirinya sendiri untuk menyelesaikan suatu masalah. Teknik ini sangat berguna ketika suatu masalah dapat dipecah menjadi sub-masalah yang lebih kecil dan serupa dengan masalah aslinya. Dalam rekursi, fungsi memanggil dirinya sendiri dengan argumen yang lebih kecil atau lebih sederhana sampai mencapai kondisi dasar yang dapat diselesaikan secara langsung tanpa perlu memanggil fungsi itu lagi.

Elemen Penting dalam Rekursi

1. **Base Case (Kondisi Dasar):**
   - Ini adalah kondisi yang menghentikan rekursi. Tanpa base case, fungsi rekursif akan memanggil dirinya sendiri tanpa henti, menyebabkan infinite loop atau kehabisan memori (stack overflow).
   - Base case adalah kasus sederhana yang dapat diselesaikan tanpa memerlukan pemanggilan rekursi lebih lanjut.
   - Contoh: Dalam perhitungan faktorial, faktorial(1) akan menjadi base case karena faktorial(1) = 1 dan tidak membutuhkan pemanggilan fungsi lebih lanjut.

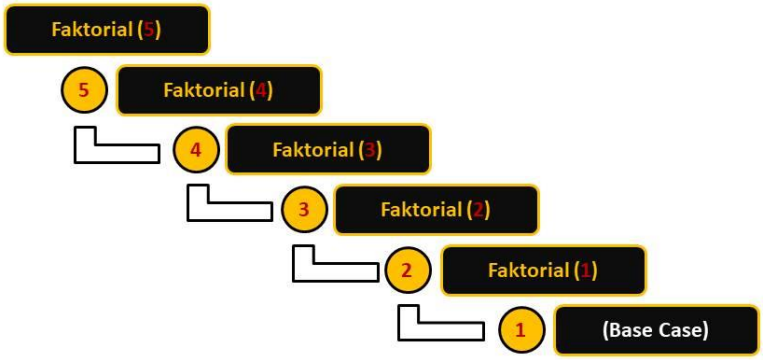
```python
def faktorial(n):
    if n == 1:  # Base case
        return 1
    else:
        return n * faktorial(n - 1)
```

2. **Recursive Case (Kondisi Rekursif):**
   - Ini adalah bagian dari fungsi di mana rekursi benar-benar terjadi. Fungsi memanggil dirinya sendiri dengan argumen yang lebih kecil atau sederhana, secara bertahap mendekati base case.
   - Kondisi rekursif harus mengarah ke base case agar rekursi dapat berhenti.
   - Dalam contoh fungsi faktorial, faktorial(n-1) adalah recursive case yang mengarah ke base case faktorial(1).

```python
def faktorial(n):
    if n == 1:  # Base case
        return 1
    else:  # Recursive case
        return n * faktorial(n - 1)
```

3. **Stack Rekursi:**
   - Ketika sebuah fungsi rekursif dipanggil, panggilan tersebut ditempatkan di dalam stack (tumpukan) sampai base case tercapai. Setelah itu, nilai-nilai dari panggilan rekursif sebelumnya dihitung dan diambil dari stack.
   - Jika base case tidak tercapai atau recursive case tidak mengurangi kompleksitas masalah, stack akan terus menumpuk, yang bisa menyebabkan stack overflow.

4. **Visualisasi Rekursi:**
   - Rekursi dapat divisualisasikan sebagai sebuah pohon di mana setiap panggilan rekursif adalah cabang baru, dan base case adalah daun (ujung) pohon.
   - Dalam perhitungan faktorial '**5!**', pohon rekursinya akan terlihat seperti ini:



Cara Kerja Diagram:

- Faktorial(5) memanggil faktorial(4), dan menunggu hasil dari faktorial(4).
- Faktorial(4) memanggil faktorial(3), dan menunggu hasil dari faktorial(3).
- Proses ini terus berlanjut hingga faktorial(1) dipanggil.
- Faktorial(1) adalah base case, yang mengembalikan 1. Kemudian, setiap pemanggilan fungsi kembali ke level sebelumnya dengan hasil perhitungannya.
- Faktorial(2) menghitung 2 * 1.
- Faktorial(3) menghitung 3 * 2.
- Faktorial(4) menghitung 4 * 6.

- Akhirnya, faktorial(5) menghitung 5 * 24, dan menghasilkan 120.

5. Contoh Sederhana : Faktorial

Faktorial dari angka n adalah hasil kali dari semua angka dari 1 sampai n. Misalnya, faktorial dari 5 (ditulis 5!) adalah:

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

Cara Kerja Rekursi untuk Faktorial:

- Jika ‘n = 1’, maka ‘1! = 1’.(ini adalah kasus dasar)
- Jika ‘n > 1’, maka ‘n! – n x (n-1)!’. Misalnya,’5! – 5 x 4!’.

Contoh Kode Python:

```python
def faktorial(n):
    if n == 1:  # Kasus dasar
        return 1
    else:
        return n * faktorial(n - 1)  # Kasus rekursif

# Contoh penggunaan:
print(faktorial(5))  # Output 120
```

6. Mengapa Rekursi Penting?

Rekursi penting karena memungkinkan kita menyelesaikan masalah yang kompleks dengan cara yang sederhana. Banyak masalah besar bisa dipecah menjadi masalah yang lebih kecil dan lebih mudah dipecahkan dengan rekursi.

7. Kelebihan dan Kekurangan Rekursi

Kelebihan:

- Mempermudah penyelesaian masalah yang dapat dipecah menjadi sub-masalah serupa.
- Membuat kode lebih bersih dan lebih mudah dibaca untuk masalah yang kompleks.

Kekurangan:

- Penggunaan memori yang lebih besar karena tumpukan panggilan fungsi.
- Risiko stack overflow jika tidak ada base case yang tepat atau masalah tidak dipecah dengan benar.

B. Latihan/contoh

Soal Latihan:

1. Buatlah fungsi rekursif untuk menghitung faktorial dari suatu bilangan bulat positif.
2. Implementasikan fungsi rekursif untuk menghitung deret Fibonacci hingga elemen ke-n.
3. Buatlah fungsi rekursif untuk menghitung pangkat dari suatu bilangan.

Cara Pengerjaan dan Coding:

Latihan 1: Faktorial

Soal: Hitung faktorial dari angka n menggunakan rekursi.

Kode Python:

```python
def faktorial(n):
    if n == 1:
        return 1  # Base Case
    else:
        return n * faktorial(n - 1)  # Recursive Case

# Contoh Penggunaan
hasil = faktorial(5)
print(f"Hasil faktorial 5 adalah: {hasil}")
```

Penjelasan:

- Base Case: if n == 1, fungsi akan mengembalikan 1 tanpa memanggil dirinya lagi.
- Recursive Case: return n * faktorial(n - 1) mengurangi nilai n secara bertahap hingga mencapai base case.

Latihan 2: Fibonacci

Soal: Hitung deret Fibonacci ke-n menggunakan rekursi.

Kode Python:

```python
def fibonacci(n):
    if n == 0:
        return 0  # Base Case 1
    elif n == 1:
        return 1  # Base Case 2
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)  #
Recursive Case

# Contoh Penggunaan
hasil = fibonacci(6)
print(f"Fibonacci ke-6 adalah: {hasil}")
```

Penjelasan:

- **Base Case:** if n == 0 mengembalikan 0 dan if n == 1 mengembalikan 1.
- **Recursive Case:** return fibonacci(n - 1) + fibonacci(n - 2) menghitung nilai Fibonacci berdasarkan dua nilai sebelumnya.

Latihan 3: Pangkat

Soal: Hitung nilai pangkat dari bilangan x terhadap y menggunakan rekursi.

Kode Python:

```
def pangkat(x, y):
    if y == 0:
        return 1  # Base Case
    else:
        return x * pangkat(x, y - 1)  # Recursive
Case

# Contoh Penggunaan
hasil = pangkat(2, 3)
print(f"2 pangkat 3 adalah: {hasil}")
```

Penjelasan:

- **Base Case:** if y == 0 mengembalikan 1 karena setiap bilangan pangkat 0 adalah 1.
- **Recursive Case:** return x * pangkat(x, y - 1) mengurangi nilai y secara bertahap hingga mencapai base case.

C. Rangkuman

Rekursi adalah teknik pemrograman di mana fungsi memanggil dirinya sendiri untuk menyelesaikan masalah yang dapat dipecah menjadi sub-masalah yang lebih kecil dan mirip dengan masalah awal. Elemen penting dalam rekursi meliputi kondisi dasar (base case) yang menghentikan rekursi, kondisi rekursif yang membuat fungsi memanggil dirinya sendiri, dan stack rekursi yang menyimpan setiap panggilan fungsi hingga base case tercapai. Rekursi sering digunakan dalam penyelesaian masalah seperti faktorial, Fibonacci, dan pangkat, karena memungkinkan penyelesaian yang lebih sederhana untuk masalah kompleks, meskipun dengan risiko penggunaan memori yang lebih besar dan kemungkinan stack overflow.

D. Tugas

1. Jelaskan konsep base case dan recursive case dalam rekursi, serta berikan contohnya.
2. Diskusikan kelebihan dan kekurangan penggunaan rekursi dalam pemrograman.

3. Implementasikan fungsi rekursif untuk menghitung faktorial dari bilangan n.
4. Buatlah fungsi rekursif untuk menghitung nilai pangkat x^y.
5. Tulis fungsi rekursif untuk menentukan apakah sebuah bilangan adalah bilangan prima.

E. Pustaka

Kadir, A. (2016). Dasar-Dasar Algoritma. Yogyakarta: Andi Offset.

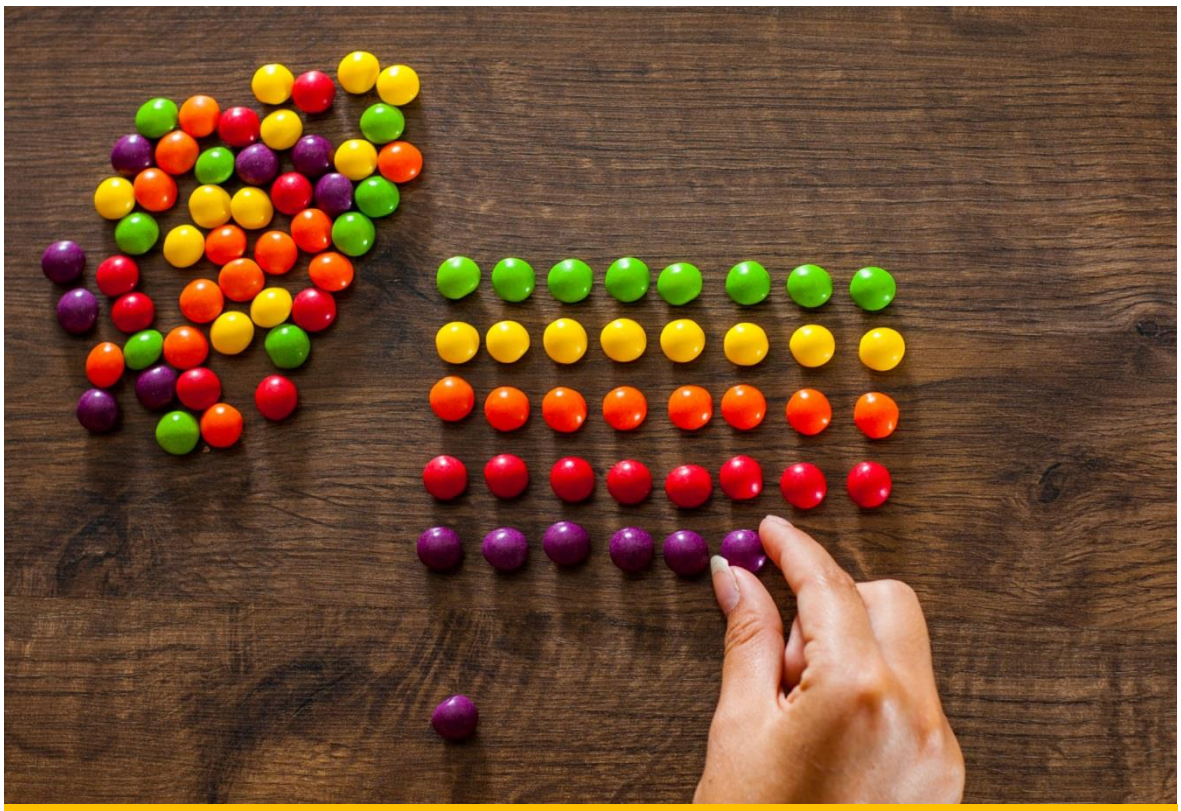
Knuth, D. E. (1997). The Art of Computer Programming, Volume 1: Fundamental Algorithms. Addison-Wesley.

Mulyono, D. (2018). Pemrograman Komputer dengan C dan Python. Yogyakarta: Graha Ilmu.

Purnomo, H. (2020). Belajar Python untuk Pemula. Bandung: Informatika.

Suryadi, S. (2021). Pemrograman Dasar dengan Python. Bandung: CV. Alfabeta.

Modul 7: Sorting



Modul ini dirancang sebagai panduan komprehensif untuk mempelajari dasar-dasar sorting, mencakup konsep, algoritma, dan implementasi praktis. Modul ini memiliki relevansi tinggi bagi mahasiswa, terutama dalam memahami bagaimana data dapat diorganisir secara efisien, yang merupakan fondasi penting dalam berbagai aplikasi pemrograman dan pengembangan perangkat lunak. Dengan mengikuti modul ini, mahasiswa diharapkan mampu memahami konsep sorting, menguasai tiga algoritma sorting sederhana (Bubble Sort, Selection Sort, dan Insertion Sort), serta mampu mengimplementasikan dan menganalisis kinerja dari masing-masing algoritma tersebut dalam berbagai situasi yang berbeda.

A. Penyajian Materi

1. Definisi Sorting :

Sorting adalah proses menyusun sekumpulan data dalam urutan tertentu. Data yang diurutkan dapat berupa angka, string, atau objek lain yang memiliki relasi urutan. Sorting penting untuk meningkatkan efisiensi dalam pencarian, penggabungan, dan operasi lain yang memerlukan data terstruktur.

2. Algoritma Sorting Sederhana :

Algoritma sorting sederhana adalah teknik pengurutan yang mudah dipahami dan diimplementasikan, meskipun mungkin tidak selalu paling efisien. Beberapa algoritma sorting sederhana meliputi:

a. Bubble Sort: Bubble Sort adalah algoritma sorting yang bekerja dengan cara membandingkan setiap pasangan elemen bersebelahan dalam daftar dan menukar mereka jika urutannya salah. Proses ini diulangi terus menerus hingga tidak ada lagi penukaran yang diperlukan, yang berarti daftar sudah terurut.

- **Langkah-langkah**
  1. Mulai dari elemen pertama dalam daftar.
  2. Bandingkan elemen saat ini dengan elemen berikutnya.
  3. Jika elemen saat ini lebih besar dari elemen berikutnya, tukar keduanya.
  4. Pindah ke elemen berikutnya dan ulangi proses ini hingga akhir daftar.
  5. Ulangi seluruh proses untuk daftar yang lebih pendek (tidak termasuk elemen terakhir yang sudah terurut) hingga tidak ada lagi penukaran yang terjadi.
- **Contoh**

  Misalkan kita memiliki daftar

5	3	8	4	2

  1. Iterasi 1 :
  - Langkah 1: Bandingkan 5 dan 3. Karena 5 > 3, tukar mereka. Daftar menjadi

3	5	**8**	4	2

  - Langkah 2: Bandingkan 5 dan 8. Karena 5 < 8, tidak ada penukaran. Daftar tetap

3	**5**	8	**4**	2

- Langkah 3: Bandingkan 8 dan 4. Karena 8 > 4, tukar mereka. Daftar menjadi

3	5	**4**	8	**2**

- Langkah 4: Bandingkan 8 dan 2. Karena 8 > 2, tukar mereka. Daftar menjadi

3	5	4	**2**	**8**

Pada akhir iterasi pertama, elemen terbesar, 8, sudah berada di posisi akhirnya.

2. **Iterasi 2 :**
   - Langkah 1: Bandingkan 3 dan 5. Karena 3 < 5, tidak ada penukaran. Daftar tetap

3	5	**4**	**2**	**8**

   - Langkah 2: Bandingkan 5 dan 4. Karena 5 > 4, tukar mereka. Daftar menjadi

3	**4**	5	2	**8**

   - Langkah 3: Bandingkan 5 dan 2. Karena 5 > 2, tukar mereka. Daftar menjadi

3	4	**2**	**5**	8

   Pada akhir iterasi kedua, elemen kedua terbesar, 5, sudah berada di posisi akhirnya.

3. **Iterasi 3 :**
   - Langkah 1: Bandingkan 3 dan 4. Karena 3 < 4, tidak ada penukaran. Daftar tetap

3	4	**2**	5	8

   - Langkah 2: Bandingkan 4 dan 2. Karena 4 > 2, tukar mereka. Daftar menjadi

3	**2**	4	**5**	8

   Pada akhir iterasi ketiga, elemen ketiga terbesar, 4, sudah berada di posisi akhirnya.

4. **Iterasi 5 :**
   - Langkah 1: Bandingkan 3 dan 2. Karena 3 > 2, tukar mereka. Daftar menjadi

2	3	4	**5**	**8**

Hasil akhir: Daftar kini terurut menjadi [**2, 3, 4, 5, 8**].

b. Selection Sort: Selection Sort bekerja dengan cara menemukan elemen terkecil dalam daftar dan menukarnya dengan elemen pertama. Kemudian, algoritma ini melanjutkan dengan mencari elemen terkecil di sub-daftar yang belum terurut dan menukarnya dengan elemen pertama dari sub-daftar tersebut. Proses ini diulang hingga seluruh daftar terurut.

- **Langkah-langkah :**
  a. Cari elemen terkecil dalam daftar.
  b. Tukar elemen terkecil tersebut dengan elemen pertama.
  c. Ulangi proses untuk sub-daftar yang lebih kecil (dimulai dari elemen kedua) hingga semua elemen terurut.
- **Contoh :**

  Misalkan kita memiliki daftar

29	10	14	37	14

  - **Iterasi 1:** Cari elemen terkecil, yaitu 10. Tukar dengan elemen pertama (29)

10	29	14	37	14

  - **Iterasi 2:** Cari elemen terkecil di sub-daftar [29, 14, 37, 14], yaitu 14. Tukar dengan elemen kedua:

10	**14**	29	37	14

  - **Iterasi 3:** Cari elemen terkecil di sub-daftar [29, 37, 14], yaitu 14. Tukar dengan elemen ketiga:

10	14	**14**	37	29

  - **Iterasi 4:** Tidak perlu menukar elemen keempat karena hanya dua elemen tersisa:

10	14	14	**29**	37

  Hasil akhir: Daftar kini terurut menjadi [10, 14, 14, 29, 37].

c. Insertion Sort: Insertion Sort bekerja dengan cara membangun daftar terurut satu elemen pada satu waktu. Ini berarti setiap elemen dari daftar yang belum terurut diambil dan dimasukkan ke posisi yang benar dalam sub-daftar yang sudah terurut.

- **Langkah-langkah**
  a. Mulai dari elemen kedua (karena elemen pertama sudah dianggap terurut).
  b. Bandingkan elemen ini dengan elemen sebelumnya.

c. Geser elemen sebelumnya ke posisi berikutnya jika lebih besar dari elemen saat ini.
d. Masukkan elemen saat ini ke posisi yang tepat.
e. Ulangi proses untuk semua elemen berikutnya.

- **Contoh**

Misalkan kita memiliki daftar [12, 11, 13, 5, 6].

Kita akan menggunakan Insertion Sort untuk mengurutkan daftar ini.

1. **Iterasi 1 :**
   - Cari elemen terkecil dalam seluruh daftar. Elemen terkecil adalah 10.
   - Tukar elemen terkecil (10) dengan elemen pertama (29). Daftar menjadi [10, 29, 14, 37, 14].
2. **Iterasi 2 :**
   - Cari elemen terkecil dalam sub-daftar [29, 14, 37, 14]. Elemen terkecil adalah 14.
   - Tukar elemen terkecil (14) dengan elemen kedua (29). Daftar menjadi [10, 14, 29, 37, 14].
3. **Iterasi 3 :**
   - Cari elemen terkecil dalam sub-daftar [29, 37, 14]. Elemen terkecil adalah 14.
   - Tukar elemen terkecil (14) dengan elemen ketiga (29). Daftar menjadi [10, 14, 14, 37, 29].
4. **Iterasi 4 :**
   - Pada tahap ini, hanya dua elemen tersisa [37, 29]. Cari elemen terkecil, yaitu 29.
   - Tukar elemen terkecil (29) dengan elemen keempat (37). Daftar menjadi [10, 14, 14, 29, 37].

Hasil akhir: Daftar kini terurut menjadi [5, 6, 11, 12, 13].

B. Latihan/contoh

1. Latihan 1: Implementasi Bubble Sort

Petunjuk:

- Tulis fungsi untuk mengurutkan array menggunakan algoritma Bubble Sort.

- Gunakan dua loop: loop luar untuk iterasi seluruh daftar, dan loop dalam untuk membandingkan elemen-elemen yang bersebelahan.
- Lakukan penukaran elemen jika urutannya salah.

Kode Implementasi:

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr

# Contoh penggunaan
data = [64, 34, 25, 12, 22, 11, 90]
sorted_data = bubble_sort(data)
print("Sorted array:", sorted_data)
```

Penjelasan:

- Langkah 1: Mulai dari elemen pertama, bandingkan dengan elemen berikutnya, dan tukar jika perlu.
- Langkah 2: Lanjutkan ke elemen berikutnya dan ulangi proses hingga tidak ada lagi penukaran yang diperlukan.
- Output: Daftar terurut ditampilkan pada akhir program.

2. Latihan 2 : Selection Sort

Petunjuk :

- Tulis fungsi untuk mengurutkan array menggunakan algoritma Selection Sort.
- Gunakan loop untuk menemukan elemen terkecil dan tukar elemen tersebut dengan elemen pertama yang belum terurut.
- Lanjutkan proses untuk sub-daftar yang lebih kecil.

Kode Implementasi :

```python
def selection_sort(arr):
    for i in range(len(arr)):
        min_idx = i
        for j in range(i+1, len(arr)):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr

# Contoh penggunaan
data = [64, 25, 12, 22, 11]
sorted_data = selection_sort(data)
print("Sorted array:", sorted_data)
```

Penjelasan :

- Langkah 1: Temukan elemen terkecil dari seluruh array dan tukar dengan elemen pertama.
- Langkah 2: Ulangi untuk sub-daftar yang lebih kecil hingga seluruh array terurut.
- Output: Daftar terurut ditampilkan pada akhir program.

3. Latihan 3 : Insertion Sort

Petunjuk :

- Tulis fungsi untuk mengurutkan array menggunakan algoritma Insertion Sort.
- Mulai dari elemen kedua, bandingkan dengan elemen-elemen sebelumnya, dan sisipkan di tempat yang tepat.
- Geser elemen-elemen yang lebih besar untuk memberi ruang bagi elemen baru.

Kode Implementasi :

```python
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr

# Contoh penggunaan
data = [12, 11, 13, 5, 6]
sorted_data = insertion_sort(data)
print("Sorted array:", sorted_data)
```

Penjelasan :

- Langkah 1: Mulai dari elemen kedua dan bandingkan dengan elemen di sub-daftar terurut.
- Langkah 2: Geser elemen-elemen yang lebih besar untuk memberi tempat bagi elemen yang sedang disisipkan.

Output: Daftar terurut ditampilkan pada akhir program.

C. Rangkuman

Modul ini telah membahas konsep dasar sorting dan tiga algoritma sorting sederhana: Bubble Sort, Selection Sort, dan Insertion Sort. Masing-masing algoritma memiliki kelebihan dan kekurangan, dan penggunaannya tergantung pada situasi dan data yang dihadapi. Memahami dan mengimplementasikan algoritma sorting adalah keterampilan dasar yang penting dalam pemrograman.

D. Tugas

1. Implementasikan algoritma Bubble Sort, Selection Sort, dan Insertion Sort untuk mengurutkan daftar angka acak yang lebih panjang.
2. Analisis waktu eksekusi dari setiap algoritma dengan menggunakan data input yang bervariasi.
3. Jelaskan situasi di mana satu algoritma lebih efisien dibandingkan yang lain.

E. Pustaka

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms. MIT Press.

Sedgewick, R., & Wayne, K. (2011). Algorithms. Addison-Wesley.

McDowell, G. L. (2015). Cracking the Coding Interview: 189 Programming Questions and Solutions. CareerCup.

Modul 8: Searching

Modul ini dirancang sebagai panduan komprehensif untuk memahami berbagai teknik pencarian data yang esensial dalam pemrograman. Mahasiswa akan mempelajari konsep dasar pencarian data, termasuk pencarian linear, pencarian biner, dan pencarian interpolasi, serta penerapan struktur data seperti pohon untuk meningkatkan efisiensi pencarian.Modul ini memiliki relevansi tinggi bagi mahasiswa, terutama dalam memahami bagaimana cara menemukan data dengan cepat dan efisien dalam berbagai struktur data. Pengetahuan ini sangat penting dalam pengembangan perangkat lunak yang memerlukan pencarian data secara cepat dan tepat, seperti dalam pengelolaan basis data, sistem pencarian teks, dan aplikasi lain yang menangani volume data besar.Dengan mengikuti modul ini, mahasiswa diharapkan mampu memahami dan mengimplementasikan teknik pencarian yang tepat sesuai dengan kebutuhan dan karakteristik data. Mereka juga akan mendapatkan wawasan tentang cara meningkatkan kinerja aplikasi dengan memilih teknik pencarian yang paling efisien untuk situasi tertentu.

A. Penyajian Materi

1. Konsep Dasar

Pencarian data, atau search, adalah operasi fundamental yang sering digunakan dalam pemrograman. Dalam konteks array atau list di Python, pencarian linier adalah metode yang paling sederhana, namun tergolong lambat. Efisiensi pencarian dapat ditingkatkan dengan menggunakan pencarian biner. Namun, perlu diingat bahwa pencarian biner hanya dapat diterapkan pada data yang telah diurutkan. Ketika kunci baru ditambahkan atau dihapus dari array atau list, urutannya harus tetap dipertahankan, yang bisa memakan waktu karena perlu menggeser elemen-elemen lain.

Untuk mengatasi kelemahan tersebut, list tertaut dapat digunakan karena memungkinkan penyisipan dan penghapusan yang lebih cepat tanpa harus menggeser elemen lain. Namun, list tertaut hanya mendukung pencarian linier, meskipun datanya diurutkan. Oleh karena itu, penggunaan struktur data seperti pohon bisa menjadi solusi yang lebih efisien dalam melakukan pencarian.

Pohon adalah struktur data yang memungkinkan pengorganisasian data secara hierarkis. Terdapat berbagai jenis pohon pencarian, yang masing-masing memiliki keunggulan tertentu. Salah satu keuntungan utama dari pohon pencarian adalah kemampuannya untuk melakukan operasi pencarian dengan efisien, memungkinkan pencarian item tertentu dalam waktu yang lebih singkat dibandingkan dengan metode lain.

Dalam pengembangan aplikasi, pohon pencarian dapat digunakan untuk mengimplementasikan berbagai jenis wadah data. Beberapa di antaranya hanya perlu menyimpan kunci pencarian, sementara yang lain mungkin mengasosiasikan kunci dengan data tambahan, mirip dengan bagaimana ADT (Abstract Data Type) peta bekerja. Pohon pencarian ini juga bisa digunakan untuk meningkatkan efisiensi operasi seperti penyisipan, penghapusan, dan pencarian.

Konsep teknik pencarian dikembangkan untuk mengatasi keterbatasan yang ada dalam teknologi pencarian kata kunci klasik, terutama ketika menangani kumpulan data teks yang besar dan tidak terstruktur. Misalnya, dalam pencarian kata kunci, sering kali terjadi kesalahan seperti positif palsu (hasil yang tidak relevan) atau negatif palsu (hasil yang relevan tetapi tidak terdeteksi). Masalah ini

dapat terjadi karena adanya sinonim (kata yang berbeda tetapi memiliki makna yang sama) dan polisemi (satu kata yang memiliki lebih dari satu makna). Dalam pencarian konsep, teknik disambiguasi makna kata digunakan untuk memahami arti sebenarnya dari kata-kata tersebut, sehingga hasil pencarian menjadi lebih relevan.

2. Teknik Pencarian

Teknik pencarian dalam pemrograman digunakan untuk menemukan data tertentu dari sekumpulan data yang memiliki tipe yang sama. Proses pencarian biasanya dilakukan sebelum melakukan update atau penghapusan data. Ada beberapa teknik pencarian yang digunakan untuk menemukan data dalam kumpulan data. Berikut adalah beberapa teknik yang sering digunakan:

a. Pencarian Sekuensial

Pencarian sekuensial, atau linear search, adalah metode pencarian yang paling sederhana, di mana setiap elemen dalam struktur data diperiksa satu per satu mulai dari elemen pertama hingga elemen yang dicari ditemukan. Meskipun metode ini mudah dipahami dan diimplementasikan, namun tidak efisien karena memerlukan pemeriksaan setiap elemen dalam struktur data, terutama jika jumlah data sangat besar.

Contoh implementasi pencarian Sekuensial dalam Python :

```python
def sequential_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i
    return -1

arr = [5, 3, 7, 2, 8, 1]
target = 2
result = sequential_search(arr, target)
print("Indeks elemen yang dicari:", result)
```

Hasil Eksekusi :

```
Indeks elemen yang dicari: 3
```

Penjelasan :

- Algoritma mencari elemen 2 dalam array [5, 3, 7, 2, 8, 1].

- Algoritma memeriksa setiap elemen dalam array secara berurutan dari indeks 0 hingga elemen ditemukan.
- Elemen 2 ditemukan pada indeks 3, dan algoritma mengembalikan indeks tersebut.

b. Pencarian Linear

Dalam pencarian linear, elemen dicari secara berurutan satu per satu dalam struktur data. Jika ditemukan kecocokan, proses pencarian dihentikan dan elemen tersebut dikembalikan. Jika tidak ditemukan, pencarian dilanjutkan hingga seluruh elemen dalam struktur data diperiksa.

Contoh implementasi pencarian linear dalam Python:

```
def linear_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i  # Elemen ditemukan, kembalikan indeksnya
    return -1  # Elemen tidak ditemukan

arr = [3, 8, 2, 4, 6, 9, 1]
target = 4
result = linear_search(arr, target)

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

Hasil Eksekusi :

```
Elemen ditemukan pada indeks 3
```

Penjelasan :

- Algoritma melakukan pencarian elemen 4 dalam array [3, 8, 2, 4, 6, 9, 1].
- Algoritma mulai dari elemen pertama dan membandingkan setiap elemen satu per satu dengan target.
- Saat algoritma mencapai elemen pada indeks 3, ditemukan bahwa arr[3] adalah 4, sehingga algoritma mengembalikan indeks 3.
- Jika elemen tidak ditemukan, algoritma akan mengembalikan -1.

c. Pencarian Interpolasi

Pencarian interpolasi adalah teknik pencarian yang bekerja berdasarkan posisi yang diestimasi dari elemen yang dicari. Algoritma ini efektif pada data yang diurutkan dan didistribusikan secara merata. Proses pencarian dimulai dengan menghitung posisi probe sebagai estimasi posisi elemen tengah dalam array. Jika elemen yang dicari lebih besar dari elemen tengah, pencarian dilanjutkan pada subarray di sebelah kanan, dan sebaliknya. Proses ini terus berlanjut hingga elemen ditemukan atau subarray menjadi nol.

Contoh implementasi pencarian Interpolasi dalam Python:

```
def interpolation_search(arr, target):
    low = 0
    high = len(arr) - 1

    while low <= high and arr[low] <= target <= arr[high]:
        pos = low + ((target - arr[low]) * (high - low)) // (arr[high] - arr[low])

        if arr[pos] == target:
            return pos
        if arr[pos] < target:
            low = pos + 1
        else:
            high = pos - 1

    return -1

arr = [10, 20, 30, 40, 50, 60, 70, 80, 90]
target = 70
result = interpolation_search(arr, target)
print("Indeks elemen yang dicari:", result)
```

Hasil Eksekusi :

```
Indeks elemen yang dicari: 6
```

Penjelasan :

- Algoritma mencari elemen 70 dalam array terurut [10, 20, 30, 40, 50, 60, 70, 80, 90].
- Algoritma menggunakan rumus interpolasi untuk memperkirakan lokasi target dalam array berdasarkan nilai-nilai minimum (arr[low]) dan maksimum (arr[high]) dalam array.
- Pada iterasi pertama, posisi yang dihitung adalah 6, dan ditemukan bahwa arr[6] adalah 70, sehingga algoritma mengembalikan indeks 6.

d. Pencarian Biner

Pencarian biner adalah algoritma pencarian yang digunakan pada data yang telah diurutkan. Metode ini jauh lebih efisien dibandingkan pencarian linear karena hanya memerlukan waktu pencarian sebesar O(log n) dibandingkan O(n) pada pencarian linear. Pencarian biner membagi array menjadi dua bagian secara rekursif dan membandingkan elemen tengah dengan elemen yang dicari, terus mengulangi proses hingga elemen ditemukan atau semua elemen telah diperiksa.

3. Cara Kerja Pencarian Biner

Prinsip dasar pencarian biner adalah membagi array menjadi dua bagian setiap kali melakukan pencarian. Jika elemen tengah adalah elemen yang dicari, pencarian berhenti. Jika elemen yang dicari lebih kecil dari elemen tengah, pencarian dilanjutkan pada bagian kiri array, dan jika lebih besar, pada bagian kanan array. Proses ini terus berlanjut hingga elemen ditemukan atau bagian array yang tersisa menjadi kosong.

Contoh implementasi pencarian Interpolasi dalam Python:

```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  # Elemen ditemukan, kembalikan indeksnya
        elif arr[mid] < target:
            low = mid + 1  # Fokus pada setengah bagian kanan
        else:
            high = mid - 1  # Fokus pada setengah bagian kiri

    return -1  # Elemen tidak ditemukan

arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
target = 7
result = binary_search(arr, target)

if result != -1:
    print(f"Elemen ditemukan pada indeks {result}")
else:
    print("Elemen tidk ditemukan dalam array")
```

Hasil Eksekusi:

```
Elemen ditemukan pada indeks 6
```

Penjelasan:

- Algoritma melakukan pencarian elemen 7 dalam array terurut [1, 2, 3, 4, 5, 6, 7, 8, 9, 10].
- Algoritma pertama kali membagi array menjadi dua bagian dengan menentukan elemen tengah (mid).
- Pada iterasi pertama, mid adalah 5, dan karena 7 lebih besar dari arr[5], algoritma melanjutkan pencarian pada bagian kanan array.
- Pada iterasi kedua, mid adalah 6, dan ditemukan bahwa arr[6] adalah 7, sehingga algoritma mengembalikan indeks 6.

B. Latihan/contoh

1. Berikan implementasi kode Python untuk pencarian biner pada array yang terurut menurun.
2. Gunakan pencarian interpolasi untuk menemukan elemen dalam array [15, 18, 22, 27, 31, 35, 42, 49] dengan target 35.
3. Implementasikan pencarian sekuensial untuk mencari elemen 8 dalam array [9, 3, 7, 6, 8, 2, 1].

Cara Pengerjaan:

1. **Pencarian Biner Array Terurut Menurun:**
   - Modifikasi algoritma pencarian biner untuk mengakomodasi urutan menurun.
   - Bandingkan elemen tengah dengan target, dan tentukan apakah pencarian harus dilanjutkan di bagian kiri atau kanan array berdasarkan urutan menurun.

```python
def binary_search_desc(arr, target):
    low = 0
    high = len(arr) - 1

    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] > target:
            low = mid + 1  # Fokus pada setengah bagian kanan
        else:
            high = mid - 1  # Fokus pada setengah bagian kiri

    return -1

arr = [10, 9, 8, 7, 6, 5, 4]
target = 7
result = binary_search_desc(arr, target)

print(f"Elemen ditemukan pada indeks {result}")  # Output: Elemen
ditemukan pada indeks 3
```

Penjelasan:

Algoritma ini menyesuaikan pencarian biner dengan urutan menurun. Jika elemen tengah lebih besar dari target, pencarian dilanjutkan ke bagian kanan array, dan sebaliknya.

2. **Pencarian Interpolasi:**
   - Gunakan rumus interpolasi untuk memperkirakan lokasi target.
   - Periksa apakah target berada dalam subarray yang telah diperkirakan, dan lanjutkan pencarian sesuai dengan hasil interpolasi.

```
def interpolation_search(arr, target):
    low = 0
    high = len(arr) - 1

    while low <= high and arr[low] <= target <= arr[high]:
        pos = low + ((target - arr[low]) * (high - low)) //
(arr[high] - arr[low])

        if arr[pos] == target:
            return pos
        if arr[pos] < target:
            low = pos + 1
        else:
            high = pos - 1

    return -1

arr = [15, 18, 22, 27, 31, 35, 42, 49]
target = 35
result = interpolation_search(arr, target)
print(f"Indeks elemen yang dicari: {result}")  # Output: Indeks
elemen yang dicari: 5
```

Penjelasan:

Algoritma ini memperkirakan posisi target dalam array berdasarkan nilai elemen-elemen di dalamnya dan mempersempit pencarian sesuai dengan posisi yang telah diperkirakan.

3. **Pencarian Sekuensial:**

   Periksa setiap elemen dalam array satu per satu mulai dari elemen pertama hingga target ditemukan.

```
def sequential_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i
    return -1

arr = [9, 3, 7, 6, 8, 2, 1]
target = 8
result = sequential_search(arr, target)
print(f"Elemen ditemukan pada indeks {result}")  # Output: Elemen
ditemukan pada indeks 4
```

Penjelasan:

Algoritma ini mencari elemen 8 dengan memeriksa setiap elemen dalam array hingga elemen ditemukan pada indeks 4.

C. Rangkuman

Pencarian data adalah operasi dasar yang sering digunakan dalam pemrograman untuk menemukan elemen tertentu dalam suatu struktur data. Terdapat beberapa teknik pencarian yang umum, seperti pencarian sekuensial yang sederhana namun tidak efisien untuk dataset besar, pencarian biner yang efisien namun hanya bekerja pada data yang terurut, dan pencarian interpolasi yang efektif pada data yang terdistribusi merata. Pencarian biner dan interpolasi mengandalkan sifat urutan data untuk mempercepat pencarian, sedangkan pencarian sekuensial memeriksa setiap elemen secara berurutan.

D. Tugas

1. Jelaskan perbedaan antara pencarian linear dan pencarian biner, serta berikan contoh situasi di mana masing-masing lebih efektif.
2. Apa keuntungan dan kelemahan dari pencarian interpolasi dibandingkan dengan pencarian biner?
3. Implementasikan pencarian interpolasi pada array terurut [5, 10, 15, 20, 25, 30] dengan target 25.
4. Buat implementasi pencarian sekuensial untuk mencari elemen 100 dalam array [50, 40, 30, 20, 10].

E. Pustaka

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.

Knuth, D. E. (1998). The Art of Computer Programming, Volume 3: Sorting and Searching (2nd ed.). Addison-Wesley.

Mohamad Aslam Katahman, M. F. (2021). Pembangunan Aplikasi Realiti Terimbuh Untuk Pengenalan Struktur Data. Information Technology And Computer Science.

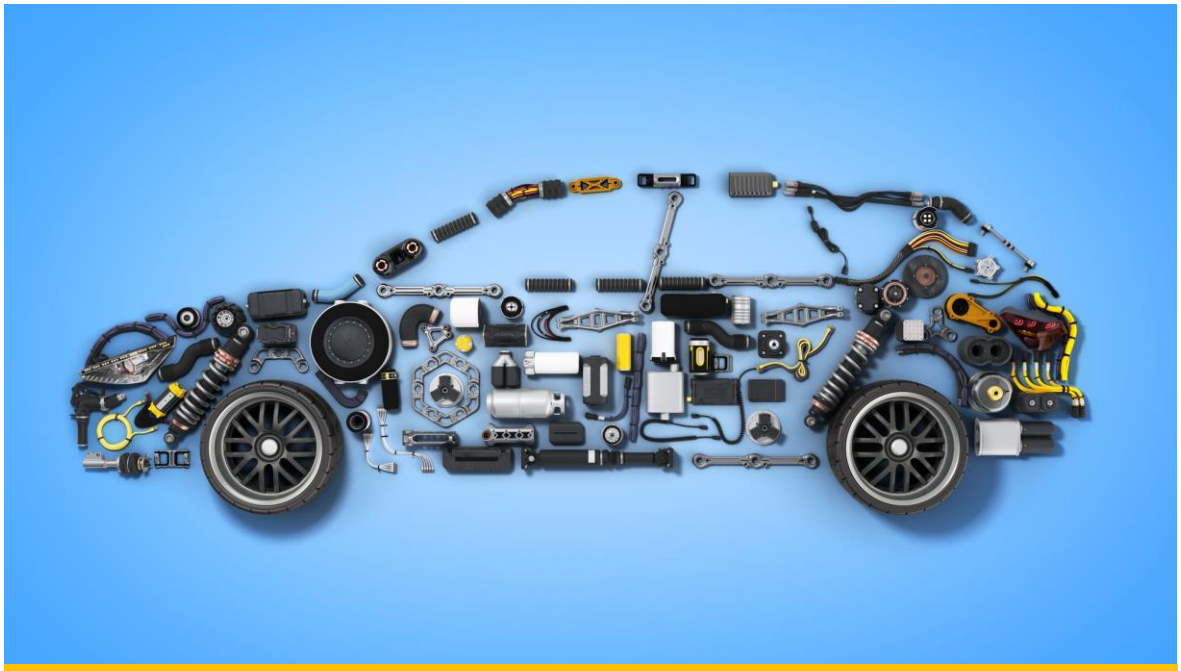
Nasrullah, A. H. (2021). Implementasi Algoritma Decision Tree Untuk Klasifikasi Produk Laris. Jurnal Ilmiah Ilmu Komputer I.

Pradana Setialana, T. B. (2017). Pencarian Hubungan Kekerabatan Pada Struktur Data Genealogy Dalam Graph Databas.

Risah Subariah, E. S. (T.Thn.). Praktikum Analisis & Perancangan Sistem

Sedgewick, R., & Wayne, K. (2011). Algorithms (4th ed.). Addison-Wesley.

Modul 9: Pemrograman Berorientasi Objek



Modul ini dirancang sebagai panduan komprehensif bagi mahasiswa untuk memahami dan mengimplementasikan konsep-konsep utama dalam Pemrograman Berorientasi Objek (OOP). Modul ini memiliki relevansi tinggi bagi mahasiswa, terutama dalam mempelajari cara kerja dan penerapan OOP dalam pengembangan perangkat lunak modern. Dengan mengikuti modul ini, mahasiswa diharapkan mampu memahami dan menerapkan prinsip-prinsip OOP seperti kelas dan objek, enkapsulasi, pewarisan, polimorfisme, dan abstraksi dalam berbagai proyek pemrograman. Modul ini juga bertujuan untuk membekali mahasiswa dengan keterampilan praktis yang diperlukan untuk menulis kode yang bersih, modular, dan mudah dikelola menggunakan paradigma OOP.

A. Penyajian Materi

Pemrograman Berorientasi Objek (Object-Oriented Programming/OOP) adalah sebuah paradigma pemrograman yang berfokus pada penggunaan "objek" sebagai unit dasar untuk membangun perangkat lunak. Objek-objek ini merupakan representasi dari dunia nyata yang berisi data dan perilaku. Dalam OOP, program dibangun dengan menggabungkan data (disebut atribut atau properti) dan fungsi (disebut metode) dalam satu unit yang disebut kelas.

1. Kelas dan Objek

a. Pengertian:

- **Kelas** adalah blueprint atau cetak biru dari objek. Kelas mendefinisikan atribut (properti) dan metode (fungsi) yang dimiliki oleh objek.
- **Objek** adalah instance atau contoh nyata dari kelas. Ketika kita membuat objek, kita sebenarnya membuat entitas yang mengikuti struktur yang sudah ditentukan oleh kelas.

b. Cara Kerja:

- Ketika sebuah kelas didefinisikan, itu hanya sebuah template.
- Ketika kita membuat objek dari kelas tersebut, objek tersebut akan memiliki atribut dan metode sesuai dengan definisi yang ada di kelas.

c. Sintaks Dasar:

```
class NamaKelas:
    def __init__(self, atribut1, atribut2):
        self.atribut1 = atribut1
        self.atribut2 = atribut2

    def metode(self):
        # kode untuk metode
        pass

# Membuat objek dari kelas
objek = NamaKelas(nilai1, nilai2)
```

d. Penjelasan:

- class NamaKelas: mendefinisikan sebuah kelas baru.
- __init__ adalah metode khusus yang disebut konstruktor, yang dipanggil ketika objek dibuat.
- self mengacu pada instance saat ini dari kelas.

2. Enkapsulasi

a. Pengertian:

Enkapsulasi adalah proses membungkus data (atribut) dan metode (fungsi) dalam satu unit, yaitu kelas. Enkapsulasi juga memungkinkan kita untuk melindungi data dari akses langsung dari luar kelas.

b. Cara Kerja:

- Data atau atribut tertentu dapat dibuat private dengan menambahkan dua garis bawah __ di awal nama atribut.
- Atribut private ini hanya bisa diakses melalui metode yang ada di dalam kelas itu sendiri.

c. Sintaks Dasar:

```
class NamaKelas:
    def __init__(self, nilai):
        self.__atribut_private = nilai  # Atribut ini dilindungi oleh enkapsulasi

    def get_atribut(self):
        return self.__atribut_private

    def set_atribut(self, nilai):
        self.__atribut_private = nilai

# Membuat objek dari kelas
objek = NamaKelas(nilai_awal)
```

d. Penjelasan:

- __atribut_private adalah contoh atribut yang dilindungi oleh enkapsulasi.
- Metode get_atribut dan set_atribut digunakan untuk mengakses dan mengubah nilai dari atribut private tersebut.

3. Pewarisan

a. Pengertian:

Pewarisan memungkinkan sebuah kelas baru (subclass) untuk mewarisi sifat dan metode dari kelas lain (superclass). Hal ini memudahkan dalam membuat kelas baru yang memiliki fitur-fitur yang sudah ada di kelas lain.

b. Cara Kerja:

Subclass akan mewarisi semua atribut dan metode dari superclass, dan bisa menambahkan atau mengubah fitur-fitur yang ada sesuai kebutuhan.

c. Sintaks Dasar:

```python
class Superclass:
    def __init__(self, atribut):
        self.atribut = atribut

    def metode_super(self):
        return "Ini dari superclass"

class Subclass(Superclass):
    def metode_sub(self):
        return "Ini dari subclass"

# Membuat objek dari subclass
objek = Subclass(nilai)
```

d. Penjelasan:

- class Subclass(Superclass): menunjukkan bahwa Subclass mewarisi dari Superclass.
- metode_super dapat digunakan oleh Subclass karena diwariskan dari Superclass.

4. Polimorfisme

a. Pengertian:

Polimorfisme memungkinkan metode yang sama untuk digunakan oleh objek dari kelas yang berbeda. Dengan polimorfisme, satu metode bisa bekerja dengan cara yang berbeda tergantung pada objek yang memanggilnya.

b. Cara Kerja:

Dalam OOP, polimorfisme sering diimplementasikan melalui metode yang memiliki nama yang sama di beberapa kelas berbeda. Metode tersebut dapat melakukan tugas yang berbeda tergantung pada kelas yang digunakan.

c. Sintaks Dasar:

```python
class KelasA:
    def metode(self):
        return "Metode di Kelas A"

class KelasB:
    def metode(self):
        return "Metode di Kelas B"

# Polimorfisme dalam aksi
objek_a = KelasA()
objek_b = KelasB()

for objek in (objek_a, objek_b):
    print(objek.metode())
```

d. Penjelasan:

- KelasA dan KelasB memiliki metode dengan nama yang sama, tetapi melakukan hal yang berbeda.
- Ketika kita memanggil metode pada objek yang berbeda, hasilnya akan disesuaikan dengan kelas dari objek tersebut.

B. Latihan/contoh

Latihan 1: Membuat Kelas dan Objek

Soal:

Buatlah kelas Mahasiswa yang memiliki atribut nama, nim, dan jurusan. Tambahkan metode untuk menampilkan informasi mahasiswa.

Cara Pengerjaan:

1. Definisikan kelas Mahasiswa dengan konstruktor __init__ untuk inisialisasi atribut.
2. Tambahkan metode tampilkan_info untuk menampilkan informasi mahasiswa.

Kode Implementasi:

```
class Mahasiswa:
    def __init__(self, nama, nim, jurusan):
        self.nama = nama
        self.nim = nim
        self.jurusan = jurusan

    def tampilkan_info(self):
        print(f"Nama: {self.nama}")
        print(f"NIM: {self.nim}")
        print(f"Jurusan: {self.jurusan}")

# Membuat objek dari kelas Mahasiswa
mhs1 = Mahasiswa("Budi", "123456", "Sistem Informasi")
mhs1.tampilkan_info()
```

Penjelasan:

- Kelas Mahasiswa memiliki tiga atribut: nama, nim, dan jurusan.
- Metode tampilkan_info digunakan untuk mencetak informasi dari objek mhs1.

Latihan 2: Implementasi Enkapsulasi

Soal: Buatlah kelas RekeningBank yang memiliki atribut private saldo. Tambahkan metode untuk menyetor dan menarik uang, serta menampilkan saldo saat ini.

Cara Pengerjaan:

1. Definisikan kelas RekeningBank dengan atribut __saldo.

2. Tambahkan metode untuk menyetor (setor_uang), menarik uang (tarik_uang), dan menampilkan saldo (tampilkan_saldo).

Kode Implementasi:

```
class RekeningBank:
    def __init__(self, saldo_awal):
        self.__saldo = saldo_awal

    def setor_uang(self, jumlah):
        self.__saldo += jumlah
        print(f"Setor: {jumlah}, Saldo saat ini: {self.__saldo}")

    def tarik_uang(self, jumlah):
        if jumlah > self.__saldo:
            print("Saldo tidak mencukupi")
        else:
            self.__saldo -= jumlah
            print(f"Tarik: {jumlah}, Saldo saat ini:
{self.__saldo}")

    def tampilkan_saldo(self):
        print(f"Saldo saat ini: {self.__saldo}")

# Membuat objek dari kelas RekeningBank
rek = RekeningBank(100000)
rek.setor_uang(50000)
rek.tarik_uang(30000)
rek.tampilkan_saldo()
```

Penjelasan:

- Atribut __saldo dilindungi dari akses langsung.
- Metode setor_uang dan tarik_uang mengubah saldo, sedangkan tampilkan_saldo menampilkan saldo saat ini.

Latihan 3: Implementasi Pewarisan

Soal:

Buatlah kelas Pegawai dengan atribut nama dan gaji. Buat kelas Manajer yang mewarisi dari Pegawai dan menambahkan atribut divisi.

Cara Pengerjaan:

1. Definisikan kelas Pegawai dan tambahkan metode tampilkan_info.
2. Definisikan kelas Manajer sebagai subclass dari Pegawai dengan tambahan atribut divisi.

Kode Implementasi:

```python
class Pegawai:
    def __init__(self, nama, gaji):
        self.nama = nama
        self.gaji = gaji

    def tampilkan_info(self):
        print(f"Nama: {self.nama}")
        print(f"Gaji: {self.gaji}")

class Manajer(Pegawai):
    def __init__(self, nama, gaji, divisi):
        super().__init__(nama, gaji)
        self.divisi = divisi

    def tampilkan_info(self):
        super().tampilkan_info()
        print(f"Divisi: {self.divisi}")

# Membuat objek dari kelas Manajer
manajer1 = Manajer("Siti", 15000000, "IT")
manajer1.tampilkan_info()
```

Penjelasan:

- Kelas Manajer mewarisi atribut dan metode dari kelas Pegawai.
- Metode tampilkan_info di kelas Manajer diperluas untuk menampilkan divisi.

C. Rangkuman

Pemrograman Berorientasi Objek (OOP) adalah paradigma yang memanfaatkan konsep kelas dan objek untuk membangun perangkat lunak. Melalui penggunaan enkapsulasi, pewarisan, dan polimorfisme, OOP memungkinkan pengembang untuk menciptakan kode yang modular, mudah dipelihara, dan dapat digunakan kembali. Dengan menggabungkan atribut dan metode dalam satu unit, OOP menyederhanakan proses pengembangan perangkat lunak yang kompleks.

D. Tugas

1. Jelaskan perbedaan antara kelas dan objek dalam Pemrograman Berorientasi Objek.
2. Bagaimana konsep pewarisan meningkatkan efisiensi dalam pengembangan perangkat lunak? Berikan contoh.
3. Buatlah kelas Perpustakaan yang memiliki atribut nama_buku dan stok. Tambahkan metode untuk menambahkan buku, meminjam buku, dan menampilkan daftar buku.

4. Buatlah kelas Karyawan dan KepalaBagian yang menggunakan pewarisan. KepalaBagian harus memiliki tambahan atribut departemen dan metode khusus untuk mengatur proyek.

E. Pustaka

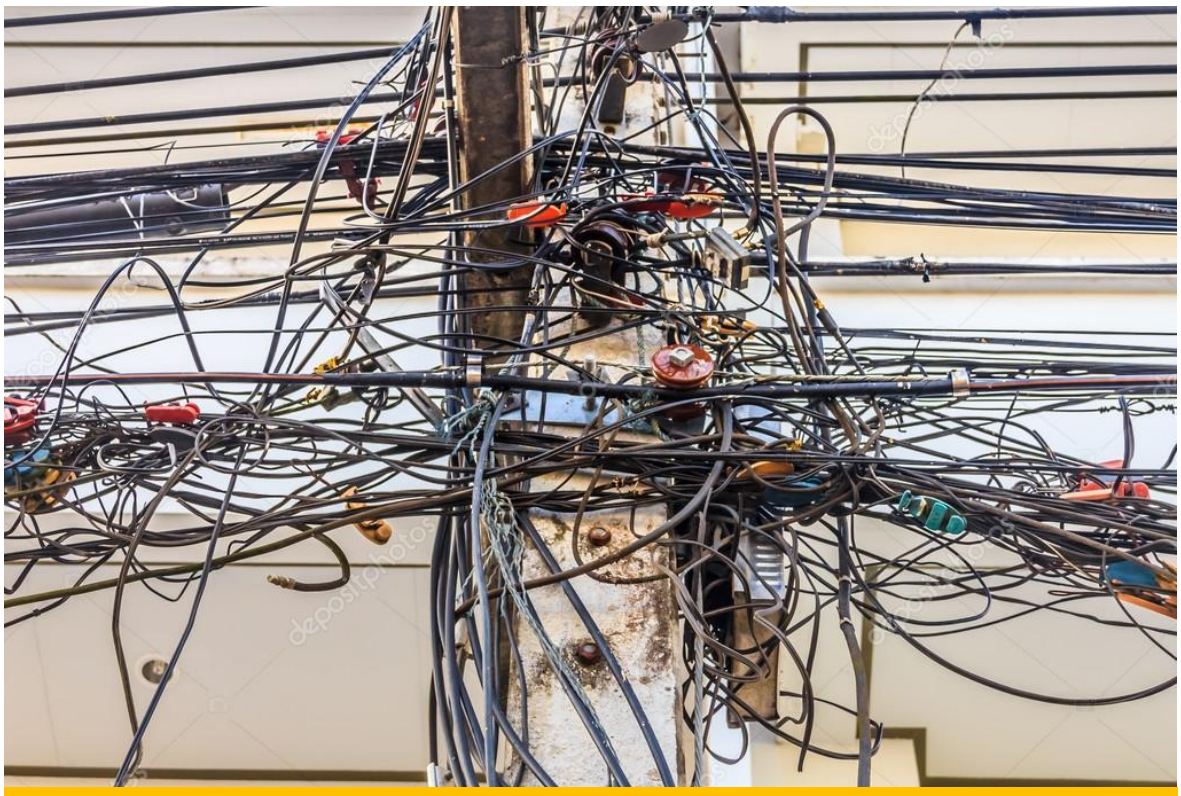
Budd, T. A. (2002). An Introduction to Object-Oriented Programming. Addison-Wesley.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms. MIT Press.

Larman, C. (2004). Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. Pearson.

McDowell, G. L. (2015). Cracking the Coding Interview: 189 Programming Questions and Solutions. CareerCup.

Modul 10: Debugging



Modul ini dirancang sebagai panduan komprehensif bagi mahasiswa untuk memahami konsep dan praktik debugging dalam pemrograman, khususnya menggunakan bahasa Python. Modul ini memiliki relevansi tinggi bagi mahasiswa, terutama dalam mengidentifikasi dan memperbaiki kesalahan yang muncul selama pengembangan program. Dengan mengikuti modul ini, mahasiswa diharapkan mampu memahami dan menerapkan teknik debugging secara efektif untuk meningkatkan kualitas kode dan efisiensi proses pengembangan perangkat lunak.

A. Penyajian materi

1. Pengertian Debugging

Debugging adalah proses mendeteksi, mendiagnosis, dan memperbaiki kesalahan atau bug dalam sebuah program. Bug dapat berupa kesalahan sintaksis, logika, atau runtime yang menyebabkan program tidak berfungsi sebagaimana mestinya. Debugging merupakan keterampilan penting yang harus dimiliki oleh setiap programmer, karena kesalahan dalam kode sering kali sulit dideteksi tanpa alat dan teknik yang tepat.

2. Jenis-jenis bug

- Kesalahan Sintaks (**Syntax Errors**): Kesalahan yang terjadi karena pelanggaran aturan sintaksis bahasa pemrograman. Misalnya, lupa menutup tanda kurung atau salah mengetik kata kunci.
- Kesalahan Logika (**Logic Errors**): Kesalahan yang tidak menyebabkan program gagal tetapi menghasilkan output yang salah. Misalnya, kesalahan dalam perhitungan atau penggunaan variabel.
- Kesalahan Runtime (**Runtime Errors**): Kesalahan yang terjadi saat program dijalankan, seperti pembagian dengan nol atau mengakses elemen yang tidak ada dalam list.

3. Teknik Debugging dasar

- **Print Statements**: Menyisipkan pernyataan print untuk melacak nilai variabel dan alur eksekusi kode.
- **Manual Debugging**: Memeriksa kode secara manual untuk menemukan kesalahan.
- **Unit Testing**: Menulis tes otomatis untuk memastikan bagian-bagian kecil dari kode berfungsi dengan benar.

4. Debugging dengan PyCharm

PyCharm menyediakan fitur debugging yang kuat untuk membantu menemukan dan memperbaiki kesalahan dengan lebih efisien:

- **Breakpoint**: Titik berhenti yang dapat diletakkan pada baris tertentu dalam kode. Program akan berhenti pada titik ini selama eksekusi, memungkinkan Anda untuk memeriksa nilai variabel dan alur eksekusi.
- **Stepping**: Menjalankan kode satu langkah pada satu waktu. Ada beberapa opsi stepping, seperti step over (melewati fungsi), step into (masuk ke dalam fungsi), dan step out (keluar dari fungsi).
- **Watches dan Evaluasi**: Memantau nilai variabel atau mengekspresikan nilai selama debugging. Anda dapat menambahkan variabel ke daftar watch untuk memantau nilai mereka secara real-time.
- **Console Debugger**: Konsol interaktif yang memungkinkan Anda menjalankan perintah Python dan memeriksa nilai variabel selama sesi debugging.

5. Contoh Debugging di PyCharm

```python
def calculate_average(numbers):
    total = 0
    for num in numbers:
        total += num
    # Misalnya kesalahan logika, seharusnya total dibagi dengan panjang list
    average = total / len(numbers)
    return average

# Daftar angka
numbers = [10, 20, 30]
print(calculate_average(numbers))  # Output seharusnya 20.0
```

- Langkah 1: Set breakpoint pada baris average = total / len(numbers).
- Langkah 2: Jalankan program dalam mode debugging.
- Langkah 3: Periksa nilai total dan len(numbers) di jendela variabel PyCharm.
- Langkah 4: Temukan bahwa nilai total dan len(numbers) sesuai, tetapi jika tidak sesuai, Anda dapat mengidentifikasi dan memperbaiki masalah.

B. Latihan/contoh

Latihan 1: Debugging Kesalahan Sintaks

Soal Latihan: Temukan dan perbaiki kesalahan sintaks pada kode berikut:

```
def greet(name):
    print("Hello, " + name + "!")

greet("Alice"
```

Cara Pengerjaan: Identifikasi kesalahan dan perbaiki kode dengan menambahkan tanda kurung yang hilang.

Kode yang Diperbaiki:

```
def greet(name):
    print("Hello, " + name + "!")

greet("Alice")
```

Penjelasan: Kesalahan sintaks disebabkan oleh tanda kurung yang tidak ditutup pada pemanggilan fungsi greet. Menambahkan tanda kurung yang hilang memperbaiki kesalahan ini.

Latihan 2: Debugging Kesalahan Logika

Soal Latihan: Debugging kode yang salah menghitung rata-rata:

```
def calculate_sum(numbers):
    total = 0
    for num in numbers:
        total += num
    return total

numbers = [5, 10, 15]
print(calculate_sum(numbers) / len(numbers))
 # Output yang diharapkan adalah 10
```

Cara Pengerjaan: Set breakpoint pada baris `return total` dan periksa nilai `total` dan `len(numbers)`.

Penjelasan: Dalam hal ini, kode sudah benar tetapi jika terdapat kesalahan, Anda harus memeriksa nilai variabel dan memastikan bahwa perhitungan dilakukan dengan benar.

Latihan 3: Menggunakan Breakpoints

Soal Latihan: Tempatkan breakpoint dan gunakan fitur stepping untuk menemukan kesalahan pada kode berikut:

```python
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

print(factorial(5))  # Output yang diharapkan adalah 120
```

Cara Pengerjaan: Tempatkan breakpoint di baris result *= i, jalankan program dalam mode debugging, dan gunakan fitur stepping untuk melacak perubahan nilai result dan i.

Penjelasan: Menyisipkan breakpoint dan menggunakan fitur stepping membantu memverifikasi setiap langkah eksekusi dan memeriksa hasil yang diharapkan.

C. Rangkuman

Modul ini membahas konsep dasar debugging, termasuk jenis-jenis kesalahan seperti kesalahan sintaks, logika, dan runtime. Teknik debugging yang dibahas meliputi penggunaan pernyataan print, manual debugging, dan unit testing. PyCharm menyediakan alat yang berguna seperti breakpoints, stepping, dan console debugger untuk mempermudah proses debugging. Dengan memahami dan memanfaatkan fitur ini, mahasiswa dapat lebih efektif dalam menemukan dan memperbaiki kesalahan dalam kode Python mereka.

D. Tugas

1. Jelaskan perbedaan antara kesalahan sintaks, logika, dan runtime dalam pemrograman. Berikan contoh untuk masing-masing jenis kesalahan.
2. Diskusikan manfaat menggunakan alat debugging seperti breakpoints dan watches dalam lingkungan pengembangan seperti PyCharm.
3. Tulis program Python yang mengandung kesalahan logika dan gunakan teknik debugging untuk menemukan dan memperbaikinya. Sertakan penjelasan tentang proses debugging yang dilakukan.
4. Implementasikan unit tests untuk fungsi sederhana seperti penjumlahan atau faktorial, dan gunakan PyCharm untuk menjalankan dan menganalisis hasil tes.

E. Pustaka

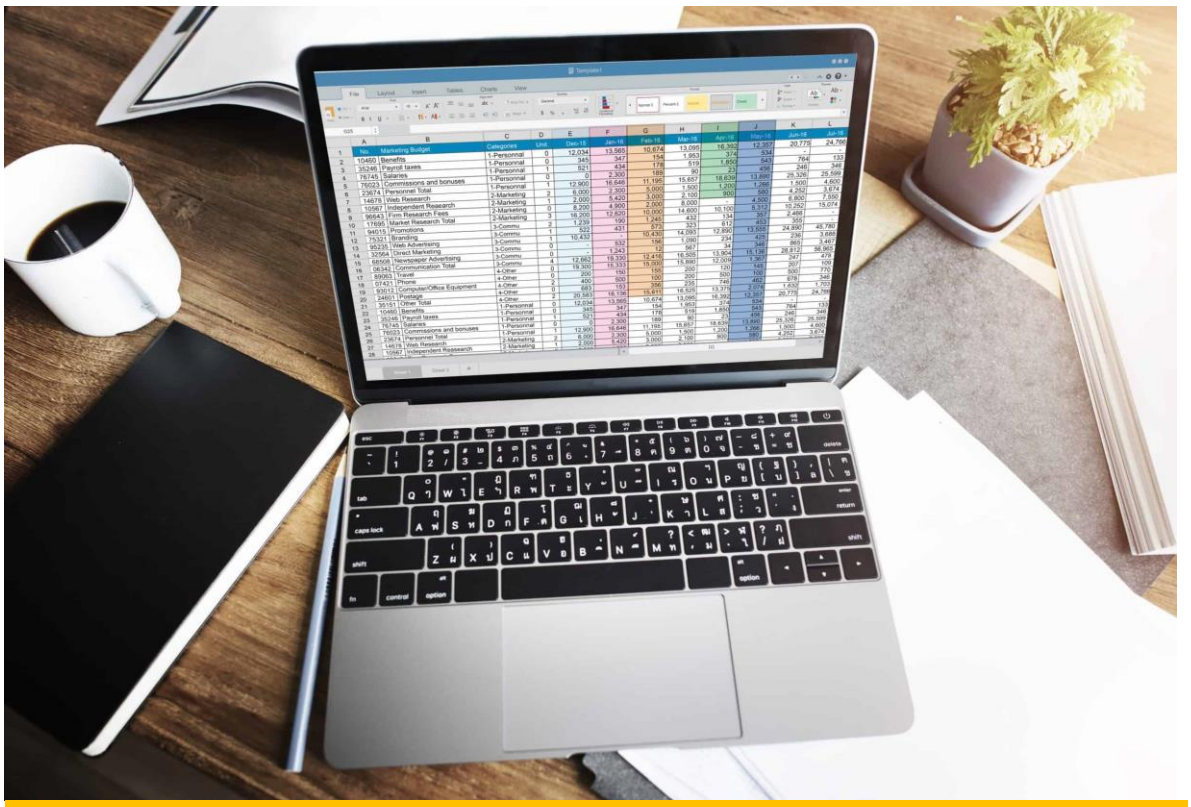
Al Sweigart. (2019). Automate the Boring Stuff with Python: Practical Programming for Total Beginners. No Starch Press.

Luciano Ramalho. (2015). Fluent Python: Clear, Concise, and Effective Programming. O'Reilly Media.

Python Software Foundation. (2023). Python Documentation: pdb — The Python Debugger. Retrieved from https://docs.python.org/3/library/pdb.html

Hunt, A., & Thomas, D. (1999). The Pragmatic Programmer: From Journeyman to Master. Addison-Wesle

Implementasi Proyek Akhir Pemrograman



Proyek akhir ini dirancang untuk menguji pemahaman dan penerapan konsep-konsep pemrograman yang telah dipelajari selama kursus, dengan mahasiswa ditantang mengembangkan solusi pemrograman kompleks dan terstruktur yang mencakup penggunaan berbagai teknik dan alat yang telah dibahas. Tujuan proyek ini adalah untuk mengintegrasikan konsep seperti algoritma pencarian dan pengurutan, pemrograman berorientasi objek, serta debugging, dan untuk mengembangkan kemampuan mahasiswa dalam merancang dan mengimplementasikan solusi pemrograman yang komprehensif, baik secara mandiri maupun dalam tim. Selain itu, proyek ini bertujuan meningkatkan keterampilan dalam penggunaan alat pengembangan seperti PyCharm dan bahasa pemrograman Python. Relevansi bagi mahasiswa terletak pada kesempatan untuk menerapkan pengetahuan teoritis dalam konteks praktis, yang penting untuk persiapan menghadapi tantangan di dunia kerja, terutama dalam mengembangkan dan mengelola proyek pemrograman yang kompleks. Pengalaman ini juga akan meningkatkan kemampuan analitis dan teknis mahasiswa melalui penyelesaian masalah nyata.

A. Deskripsi Proyek :

Mahasiswa akan mengembangkan sebuah aplikasi manajemen toko buku menggunakan Python, yang akan mencakup berbagai fitur untuk mengelola inventaris buku, penjualan, serta data pelanggan. Aplikasi ini harus menerapkan konsep-konsep yang telah dipelajari, seperti algoritma pencarian dan pengurutan, pemrograman berorientasi objek (OOP), dan debugging. Proyek ini akan diselesaikan dalam tim kecil atau secara individu.

B. Fitur Utama :

1. **Manajemen Buku:**
   - Tambah, edit, dan hapus data buku.
   - Pencarian buku menggunakan berbagai algoritma pencarian (Linear Search, Binary Search).
   - Pengurutan daftar buku berdasarkan kriteria tertentu (judul, harga, dll.) menggunakan algoritma pengurutan (Bubble Sort, Selection Sort, Insertion Sort).
2. **Manajemen Penjualan:**
   - Pencatatan transaksi penjualan buku.
   - Perhitungan total penjualan harian, bulanan, dan tahunan.
   - Penyimpanan riwayat transaksi dengan opsi pencarian dan pengurutan.
3. **Manajemen Pelanggan:**
   - Tambah, edit, dan hapus data pelanggan.
   - Pencarian pelanggan berdasarkan nama atau ID pelanggan.
   - Penyimpanan riwayat pembelian pelanggan.
4. **Pelaporan:**
   - Pembuatan laporan penjualan berdasarkan periode tertentu.
   - Laporan buku terlaris dan pelanggan dengan transaksi terbanyak**.**

C. Teknologi yang digunakan :

- Bahasa Pemrograman: Python
- IDE: PyCharm
- Database: SQLite atau file CSV untuk penyimpanan data

D. Kriteria Penilaian

1. **Penerapan Kondep:**
   - Penggunaan algoritma pencarian dan pengurutan yang tepat.
   - Penerapan prinsip OOP dalam desain kelas dan objek.
   - Efektivitas dan efisiensi kode yang dihasilkan.
2. **Kelengkapan Fitur:**
   - Implementasi semua fitur utama.
   - Kemudahan penggunaan dan antarmuka yang ramah pengguna.
3. **Debugging dan Pengujian:**
   - Identifikasi dan perbaikan bug dalam kode.
   - Pengujian aplikasi untuk memastikan semua fitur berfungsi dengan baik.
4. **Dokumentasi:**
   - Dokumentasi kode yang jelas dan mudah dipahami.
   - Laporan proyek yang menjelaskan arsitektur aplikasi, fitur, dan proses pengembangan.

E. Timeline :

- Minggu 1-2: Perencanaan proyek dan desain awal.
- Minggu 3-5: Pengembangan fitur dasar (manajemen buku, penjualan, dan pelanggan).
- Minggu 6-7: Pengembangan fitur lanjutan dan pelaporan.
- Minggu 8: Debugging, pengujian, dan finalisasi proyek.
- Minggu 9: Presentasi dan penyerahan proyek akhir.