Turma Compartilhada
Programação de Computadores

Prof. Carlos Eduardo Weber

UNICID
Universidade
Cidade de S. Paulo

Aula 02 - Conceitos

25/02/2025

Fundamentos da Programação e Lógica de Computação
Em Programação de Computadores, aprende-se a linguagem, a lógica por trás do código e como traduzir problemas do mundo real para soluções computacionais. Essa base é crucial para muitas disciplinas.

Banco de Dados
Ao adentrar em bancos de dados, percebemos a sinergia com a programação. A capacidade de desenvolver consultas SQL eficientes e integrar sistemas de informação depende diretamente de uma sólida compreensão da programação.

Desenvolvimento Web e Interfaces Gráficas
Nas disciplinas de desenvolvimento web, aplicamos conceitos de programação para criar interfaces interativas e amigáveis. Entendemos como o código do lado do cliente e do lado do servidor se unem para proporcionar uma experiência web robusta.

Redes de Computadores e Segurança da Informação
A programação é uma ferramenta vital na construção de sistemas seguros. Entendemos como desenvolver códigos que lidam com criptografia, autenticação e autorização, essenciais para proteger sistemas contra ameaças de segurança.

Inteligência Artificial e Aprendizado de Máquina
A programação é a espinha dorsal da inteligência artificial. Discutiremos como algoritmos e estruturas de dados são fundamentais para desenvolver modelos de aprendizado de máquina e como a programação influencia a evolução dessa área.

Em resumo, a programação não é apenas uma disciplina isolada, mas o fio condutor que conecta todas as áreas da Informática. Ao compreender a programação, vocês não apenas se tornam desenvolvedores proficientes, mas também profissionais capazes de integrar e inovar em diferentes segmentos da tecnologia da informação.

Lembre-se que a criatividade na programação é a chave para encontrar soluções inovadoras.

Conceitos Básicos

A programação de computadores baseia-se em um conjunto estruturado de instruções utilizadas para resolver problemas computacionais. Antes de escrever um código-fonte em qualquer linguagem de programação, é essencial compreender os fundamentos da lógica de programação e dos algoritmos, que formam a base do pensamento computacional.

A lógica de programação e os algoritmos são a base para o desenvolvimento de qualquer sistema computacional. Assim como uma casa precisa de uma fundação sólida para ser estável, um bom programador deve dominar esses conceitos para criar soluções eficientes e bem estruturadas.

Dominar a construção de algoritmos é o primeiro passo para aprender qualquer linguagem de programação e desenvolver aplicações que resolvam problemas do mundo real de forma eficiente.

Lógica de Programação
A lógica de programação é a estrutura mental necessária para criar soluções computacionais eficientes, o alicerce da construção de algoritmos. Ela envolve o pensamento estruturado e a capacidade de decompor problemas complexos em etapas lógicas, garantindo que as instruções sejam executadas de maneira ordenada e previsível.

Trata-se de pensar de maneira estruturada para resolver problemas. Quando falamos em lógica, estamos nos referindo à ordem e à sequência de passos necessários para atingir um objetivo.

No cotidiano, a lógica está presente em diversas atividades, como seguir uma receita de culinária, planejar uma viagem ou montar um móvel. No contexto da programação, a lógica é aplicada para organizar instruções de forma coerente e eficiente, garantindo que um computador possa executá-las corretamente.

Algoritmo
Um algoritmo é uma sequência finita e bem definida de passos que levam à resolução de um problema. Cada etapa de um algoritmo deve ser clara, precisa e executável.

Podemos comparar um algoritmo a uma receita de bolo, onde cada etapa deve ser seguida na ordem correta para garantir um resultado esperado. Assim como na culinária, um bom algoritmo precisa ser:

- **Claro:** Deve ser compreensível e livre de ambiguidades.
- **Preciso:** Cada etapa deve ter uma definição bem estabelecida.
- **Determinístico:** A execução deve sempre produzir o mesmo resultado para os mesmos dados de entrada.
- **Finito:** Deve ter um número limitado de etapas.

Elementos Básicos dos Algoritmos
Para que um algoritmo funcione corretamente, ele deve conter três elementos essenciais:

- **Entrada:** São os dados fornecidos ao algoritmo para processamento.
- **Processamento:** São as operações realizadas sobre os dados de entrada para gerar um resultado.
- **Saída:** É o resultado obtido após o processamento dos dados.

Esses elementos são fundamentais para entender como um algoritmo transforma dados brutos na base para informações úteis.

Tipos de Dados
Na programação, necessitamos armazenar dados temporários na memória do computador. Eles podem se apresentar em diferentes tipos, sendo os mais comuns:

- **Inteiros (*int*):** Representam números inteiros, como -3, 0, 42.
- **Ponto flutuante (*float/double*):** Representam números decimais, como 3.14 ou -0.001.
- **Caractere (*char*):** Armazena um único caractere, como 'A' ou 'z'.
- **Cadeia de caracteres (*string*):** Representa textos, como "Olá, mundo!".
- **Booleano (*bool*):** Assume apenas dois valores: verdadeiro (*true*) ou falso (*false*).

A escolha do tipo de dado adequado impacta diretamente no desempenho e no uso eficiente da memória do sistema.

Pseudocódigo e Fluxograma
Antes de implementar um algoritmo em uma linguagem de programação, é comum representá-lo de forma abstrata, utilizando pseudocódigo ou fluxogramas.

- **Pseudocódigo:** Representação textual estruturada de um algoritmo, utilizando comandos próximos da linguagem natural, mas com a lógica da programação.

  Portugol Webstudio: https://portugol.dev/

- **Fluxograma:** Representação gráfica de um algoritmo, usando símbolos padronizados para ilustrar o fluxo de execução.

  https://www.lucidchart.com/pages/what-is-a-flowchart-tutorial

Lucidchart

Ambas as abordagens ajudam a planejar e visualizar o funcionamento de um programa antes da implementação.

Raciocínio Lógico e Lógica de Programação
O raciocínio lógico e a lógica de programação são pilares cruciais para quem busca se destacar no universo da computação.

Raciocínio Lógico
O raciocínio lógico é a habilidade de pensar e tomar decisões de maneira consistente e ordenada. Ele é a base para a construção de algoritmos eficientes e soluções elegantes em programação.

Relação entre Raciocínio Lógico e Programação
Em programação, cada algoritmo segue uma sequência de passos lógicos para atingir um objetivo. Desenvolver essa habilidade ajuda a estruturar algoritmos de maneira clara e compreensível.

Ver como a aplicação de raciocínio lógico na programação deve ser realizada com cuidado no vídeo a seguir:

Como ensinar linguagem de programação para uma criança
https://www.youtube.com/watch?v=pdhqwbUWf4U

Tomada de Decisões
No desenvolvimento de software, frequentemente encontramos situações em que decisões precisam ser tomadas. Entender raciocínio lógico é crucial para implementar estruturas condicionais eficientes.

Resolução de Problemas
O raciocínio lógico é uma ferramenta valiosa na resolução de problemas. Desmembrar um desafio complexo em partes menores e mais gerenciáveis é uma habilidade que facilita a criação de algoritmos eficientes.
O raciocínio lógico é um aliado poderoso na lógica de programação. Quanto mais desenvolvida essa habilidade, mais eficientes e elegantes serão suas soluções. Lembrem-se, assim como um músculo, o raciocínio lógico se fortalece com prática e desafios constantes.

Como fazer um algoritmo
Existe uma sequência de passos recomendável para criar um algoritmo. Vejamos os passos

1. Compreenda o problema
2. Verifique quais dados de entrada serão necessários
3. Liste as saídas esperadas
4. Procure quais meios são os mais adequados para processar os dados de entrada, convertendo-os nas saídas
    a. Divida os passos necessários para a conversão em passos distintos, curtos e específicos
    b. Estruture os passos em uma sequência lógica
5. Teste as situações mais comuns e as situações de limite.

Estrutura de Programas
A estrutura de um programa é como a espinha dorsal da programação. Compreender como organizar dados é crucial para criar sistemas coerentes e compreensíveis.

Dados e Informações
Dados são a matéria-prima da programação, e informações são o que obtemos a partir desses dados. É vital entender como representar e manipular dados para construir soluções eficazes.

Variáveis
As variáveis são espaços de armazenamento nomeados que guardam valores mutáveis de determinados tipos de dados durante a execução do programa. Elas são fundamentais para a dinâmica e adaptabilidade dos algoritmos.

Constantes
Ao contrário das variáveis, as constantes mantêm valores fixos ao longo do programa, definidos na criação delas. Isso é útil quando temos dados que não devem ser alterados durante a execução.

Finais
São como as constantes, mas o valor não precisa ser definido na criação.

Declaração e Inicialização de Variáveis
A declaração informa ao programa o nome e tipo da variável, enquanto a inicialização atribui um valor a ela. Esses passos são cruciais para o uso correto das variáveis.

Escopo de Variáveis
O escopo define onde uma variável pode ser acessada. Entender o escopo é fundamental para evitar erros e garantir que a memória e as variáveis sejam utilizadas de maneira apropriada.

Boas Práticas e Convenções
Adotar boas práticas, como escolher nomes descritivos para variáveis e seguir convenções de codificação, torna o código mais legível e facilita a colaboração.

Linguagens de Programação
Entender as diferentes linguagens e suas características é essencial para escolher a ferramenta certa para cada tarefa.

As linguagens de programação são meios de comunicação entre os programadores e os computadores. Elas fornecem instruções para a máquina executar tarefas específicas e são essenciais para o desenvolvimento de software.

Classificação das Linguagens de Programação
As linguagens de programação podem ser classificadas de várias maneiras. Algumas categorias comuns incluem:
- Linguagens de Baixo Nível: Mais próximas da linguagem de máquina, como Assembly.
- Linguagens de Alto Nível: Mais abstratas e próximas da linguagem humana, como Python, Java, e C++.
- Linguagens de Script: Projetadas para automação de tarefas, como JavaScript e Python.
- Linguagens Funcionais e Orientadas a Objeto: Paradigmas que definem a estrutura do código, como Haskell (funcional) e Java (orientada a objeto).

Selecionando uma Linguagem
A escolha da linguagem depende do contexto e dos requisitos do projeto. Considerações incluem eficiência, legibilidade, suporte a paradigmas específicos e ecossistema de bibliotecas disponíveis.

Linguagens de Programação Populares

- Python: Versátil, fácil de aprender, amplamente utilizado em ciência de dados e desenvolvimento web.
- JavaScript: Linguagem de script para desenvolvimento web, suporta programação assíncrona.
- Java: Portátil, orientada a objetos, amplamente usada em desenvolvimento empresarial.
- C e C++: Poderosas, usadas em sistemas embarcados, jogos e software de sistema.
- Ruby: Foco em simplicidade e produtividade, frequentemente usado em desenvolvimento web.
- C#: Desenvolvimento de software Windows e jogos com a Unity.
- Swift: Desenvolvimento de aplicativos iOS.
- Go (Golang): Eficiente, focada em concorrência, ideal para servidores.
- Visual Basic: Linguagem de programação visual da Microsoft, amplamente utilizada para o desenvolvimento de aplicativos de desktop e interfaces gráficas de usuário
- PHP: Linguagem de programação de código aberto amplamente usada para desenvolvimento web, destacando-se pela integração fácil com HTML e pela capacidade de criar páginas dinâmicas e interativas.

Software proprietário e livre

O **software proprietário** é desenvolvido e distribuído por uma entidade que detém os direitos exclusivos sobre o código-fonte. Os usuários geralmente precisam pagar para adquirir uma licença e não têm acesso ao código-fonte do programa. Software livre refere-se a programas cujo código-fonte é disponibilizado ao público, permitindo que os usuários visualizem, modifiquem e distribuam o software. O conceito fundamental é a liberdade do usuário, não necessariamente a gratuidade. O código aberto vai além do software livre, enfatizando a acessibilidade ao código-fonte e promovendo uma comunidade colaborativa. Os termos específicos podem variar, mas muitas licenças de código aberto garantem a liberdade de uso, modificação e redistribuição.

Os programas de computador são distribuídos sob diversas licenças, como a GPL (GNU General Public License), uma licença de **software livre** que assegura que qualquer software derivado também seja distribuído sob a mesma licença. MIT e BSD são licenças permissivas que permitem maior flexibilidade, permitindo que o código seja incorporado em software proprietário sem a necessidade de abrir o código-fonte.

Além das licenças, as leis de propriedade intelectual, como patentes e direitos autorais, desempenham um papel importante na governança do uso e distribuição de software. Entender esses conceitos é crucial para desenvolvedores, empresas e usuários, pois influenciam como o software é criado, distribuído e compartilhado na comunidade tecnológica.

Evolução das Linguagens de Programação

Desde as linguagens de baixo nível, como Assembly, até as linguagens de alto nível como Python e JavaScript, observou-se uma evolução marcante. Cada nova linguagem trouxe avanços em produtividade, abstração e eficiência. Ao longo dos anos, surgiram diversos paradigmas, como programação funcional e orientada a objetos. Essa diversidade permite aos programadores escolher a abordagem mais adequada para cada tarefa.
A história das linguagens de programação é uma jornada fascinante que reflete a evolução da computação.
Na década de 1940, os primeiros computadores eram programados diretamente em código de máquina, uma tarefa complexa e propensa a erros. No início a programação era feita diretamente em painéis de pega.

O Assembly surgiu como uma linguagem simbólica para facilitar a programação de baixo nível.
Na década de 1950, a IBM desenvolveu o Fortran, a primeira linguagem de alto nível, projetada para cálculos científicos. Sua abordagem simplificada influenciou a criação de outras linguagens. A COBOL foi desenvolvida para aplicações comerciais, destacando-se pela legibilidade e uso de inglês estruturado. LISP foi criada para inteligência artificial e processamento simbólico. Introduziu conceitos inovadores, como listas e funções recursivas.

Evolução das Linguagens de Programação (continuação)
Já na década de 1970, a linguagem C, desenvolvida por Dennis Ritchie na Bell Labs, trouxe eficiência e portabilidade. Sua popularidade foi impulsionada pelo desenvolvimento do sistema operacional Unix.
Pascal foi projetada para ensinar programação estruturada. Surgiram diversas linguagens educacionais nesse período para facilitar o aprendizado.

Uma extensão do C, o C++ introduziu a programação orientada a objetos na década de 1980, proporcionando reutilização de código e modularidade. Esse paradigma ganhou mais notoriedade na década seguinte, quando Java foi desenvolvida pela Sun Microsystems para aplicativos distribuídos na web. Sua portabilidade e segurança foram fundamentais para o *boom* da programação web. Python emergiu como uma linguagem de alto nível, fácil de aprender e versátil, adequada para diversas aplicações, desde desenvolvimento web até análise de dados. A Internet trouxe evoluções como a JavaScript, que surgiu para tornar as páginas web interativas. Hoje, é uma das linguagens mais importantes para o desenvolvimento web.

Entre o final do século XX e início do século XXI, surgiram linguagens como Ruby, Swift, Kotlin, Rust e TypeScript, cada uma projetada para atender a necessidades específicas, desde desenvolvimento móvel até sistemas seguros.

Tendências para o Futuro

Linguagens como Python e R estão liderando no desenvolvimento de aplicações de IA e *machine learning*, com forte investimento e pesquisa contínua nessa área.

JavaScript continua sendo crucial para o desenvolvimento web, e há uma tendência crescente em frameworks como React, Angular e Vue.js, que oferecem experiências mais ricas e interativas.

Linguagens como Rust e Kotlin estão ganhando destaque devido ao foco em segurança e confiabilidade, atendendo às demandas por sistemas robustos.

Com o avanço da computação quântica, linguagens como Qiskit e Cirq estão sendo desenvolvidas para programação de computadores quânticos, abrindo novas possibilidades no processamento de dados em *Quantum Computing*.

A tendência é que diferentes linguagens coexistam, cada uma especializada em um domínio específico. Isso permite aos desenvolvedores escolher a ferramenta certa para a tarefa em questão.

Tendências para o Futuro

A evolução das linguagens de programação é um reflexo das necessidades em constante mudança na indústria de tecnologia. Estar ciente das tendências emergentes e adaptar-se às novas linguagens e paradigmas é crucial para se manter relevante como programador.

Lembrem-se, a curiosidade e a adaptabilidade são as chaves para acompanhar o dinamismo do mundo da programação.

Entender as linguagens de programação é fundamental para o sucesso na área. Cada linguagem tem suas características únicas, vantagens e limitações. A escolha certa depende do contexto e dos objetivos do projeto. A diversidade de linguagens é uma força, não uma barreira. Aprofundar-se em diferentes linguagens expande a visão e habilidades como programadores.

Iniciando em lógica e programação

Configurando o ambiente

Vamos mergulhar nos fundamentos da programação e entender a necessidade e as diferenças entre compilação e interpretação. Esses são conceitos fundamentais que moldam o modo como os programas são executados.

Introdução

Ao escrever um programa, estamos essencialmente dando instruções ao computador. A forma como essas instruções são processadas é o que diferencia a compilação de interpretação.

Compilação
Compilação é um processo que ocorre antes da execução do programa. Um compilador traduz o código-fonte escrito em uma linguagem de programação para código de máquina ou código intermediário, criando um arquivo executável. Isso significa que o programa é transformado em linguagem compreensível pela máquina antes de ser executado.

Vantagens da Compilação
- Desempenho: O código compilado geralmente é mais eficiente, pois a tradução prévia permite otimizações específicas da máquina.
- Segurança: Como o código-fonte não é necessário após a compilação, o executável é mais difícil de ser revertido para a forma original.

Desvantagens da Compilação
- Processo Adicional: A necessidade de compilar antes da execução adiciona uma etapa extra ao desenvolvimento.
- Portabilidade: O código compilado pode ser específico para uma plataforma, requerendo compilação separada para diferentes sistemas.

Interpretação
A interpretação ocorre durante a execução do programa. Um interpretador lê o código-fonte linha por linha e o traduz para código de máquina em tempo real. Isso significa que o código é executado diretamente, sem a necessidade de uma etapa de compilação anterior.

Vantagens da Interpretação
- Facilidade de Desenvolvimento: A execução imediata facilita o teste e a depuração do código.
- Portabilidade: Um único código-fonte pode ser executado em diferentes plataformas, desde que haja um interpretador disponível.

Desvantagens da Interpretação
- Desempenho: Geralmente é mais lento, já que a tradução ocorre em tempo real durante a execução.
- Segurança: O código-fonte pode ser mais exposto, pois o interpretador precisa acessá-lo durante a execução.

Híbridos
Algumas linguagens, como Java, adotam uma abordagem híbrida. O código é compilado para bytecode, que é interpretado por uma máquina virtual.

Conclusão
Em resumo, compilação e interpretação são abordagens diferentes para executar programas. A escolha entre elas depende das necessidades do projeto, do desempenho desejado e da facilidade de desenvolvimento.

“A vida é uma eterna fonte de aprendizado ”

UNICID
Universidade
Cidade de S. Paulo

www.unicid.edu.br

R. Cesário Galeno, 475
03071 000
São Paulo SP Brasil
T F 55 11 2178 1212