

DIO – TQI Fullstack Developer Bootcamp

CURSO: INTRODUÇÃO À PROGRAMAÇÃO E PENSAMENTO COMPUTACIONAL

Juliana Mascarenhas – Instrutora Dio

Primeiros passos para começar a programar

Estas aulas tem o objetivo de que, ao final, o dev seja capaz de entender o que significa pensar computacional, para aplicar a qualquer área do conhecimento, resolvendo problemas de uma maneira mais objetiva e eficiente.

Aula 1 – Pensamento Computacional

Aula 2 – Introdução à lógica da programação

Aula 3 – Fundamentos de Algoritmos

Aula 4 – Linguagens de programação

Aula 5 – Primeiro contato com a programação

Aula 1 – Pensamento Computacional

Apresentar conceitos da área para que o Dev entenda o pensar computacional.

1 – Introdução ao pensamento computacional

Processo de pensamento envolvido na expressão de soluções em passos computacionais ou algoritmos que podem ser implementados no computador.

Pensamento computacional: sistemático e eficiente, formulação e resolução de problemas, que sejam capazes de serem resolvidos por humanos e máquinas.

É uma habilidade generalista utilizada em diversas áreas (matemática, leitura, escrita etc.).

4 Pilares: Decomposição, Reconhecimento de Padrões, Abstração e Design de Algoritmos.

Decomposição: Dividir um problema complexo em subproblemas (“menores”) para facilitar na resolução.

Reconhecimento de Padrões: Identificar padrões ou tendências; similaridades e diferenças entre os problemas.

Abstração: extrapolar o conceito do problema para uma forma generalista. Partir do particular, específico para o generalista. Parte do mundo concreto para o mundo das ideias.

Design de algoritmos: Automatizar. Definir passo a passo para a solução do problema. Input (entrada) – Operator (operações) – Output (resultado).

Processo contínuo: definir uma solução; testar a solução; e aperfeiçoar a solução. Refinamento – Teste – Análise. Sempre se perguntar se existe uma solução melhor.

Abstração – resolução de problemas; e Automatização (expressão de solução – dados, algoritmos): ambos são *habilidades humanas*. Análise – execução da solução e avaliação: *recurso computacional*.

Competências adquiridas com o pensamento lógico computacional: pensamento sistemático, colaboração dentro da equipe, criatividade e design, facilitador.

Habilidades Complementares

Raciocínio Lógico e Aperfeiçoamento.

Raciocínio Lógico é uma forma de pensamento estruturado, que permite encontrar a conclusão ou determinar a resolução de um problema.

Métodos:

Indução – fenômenos observados – induzem à Leis e teorias. Ligado às ciências experimentais.

Dedução – parte de Leis e teorias – e deduz as previsões e explicações. Método das ciências exatas.

Abdução – a partir de uma conclusão se tira a premissa. Observa algo e conclui daí alguma coisa. Diagnósticos e processos investigativos. Há um cenário e para chegar a esse situação tem que ter acontecido essa e aquela coisa.

Inferência – Sintética (Abdução e Indução) ou Analítica (Dedução).

Ex. Pai, mãe e casal de filhos. Homens Roberto e Sérgio, mulheres Teresa e Fernanda. Sabe-se que o pai está à frente de Fernanda e o filho está à esquerda; e que a mãe está do lado direito de Sérgio.

Se conclui que o pai só pode ser Sérgio, a filha é Fernanda, mãe é Teresa e Roberto o filho.

Aperfeiçoamento – melhoramento, ato de aperfeiçoar, aprimoramento, refinamento: a partir de uma solução, determinar pontos de melhora e refinamento. Pontuais ou globais/gerais.

Melhor uso de recursos: Encontrar soluções eficientes; e, otimizar processos.

Melhorar códigos e algoritmos: simplificar linhas de códigos; e, funções bem definidas.

PILARES: DECOMPOSIÇÃO

“Se você tem um problema que não consegue resolver, existe um problema mais fácil que você consegue resolver: encontre-o.” George Polya – matemático.

Decompor, segmentar, dividi-lo. Assim torna mais fácil o processo de encontrar a solução.

Dado um problema complexo, devemos quebra-lo em problemas menores. Portanto, problemas mais fáceis e gerenciáveis.

Estratégia:

Análise – processo de quebrar e determinar partes menores. Estudar, explorar, realizar exame detalhado.

Síntese – combinar os elementos recompondo o problema original. Processo de reconstrução, fundir os elementos de maneira coerente. Consiste em reunir elementos distintos em um único elemento. (É o processo inverso).

Ordem de execução de tarefas menores:

Sequencial – dependência entre tarefas. Executadas em fila. Pois, a ordem importa para o resultado final. Cada coisa pressupõe a etapa anterior para poder ser realizada.

Paralelo – executadas concomitantemente, porque são independentes e podem ser reunidas depois.

Neste processo podem ocorrer variáveis, pequenos problemas, segmentação. Para estes casos não basta aplicar, precisa treino, precisa experiência para resolver, e muitas vezes até mesmo uma nova solução que se adapte para atender essas questões.

Muitas vezes, ao criar a solução, pode haver maneiras distintas de decompor o mesmo problema. Uma pode ser mais eficiente que outra, assim, tem que testar cada uma e verificar qual a melhor para o caso. E mesmo depois da solução pronta, isso fará parte do ciclo de constante aperfeiçoamento.

Como decompor?

Identificar ou coletar dados – Agregar os dados – Funcionalidade (o resultado, a solução).

Ex. Cozinhar – identificar ingredientes, determinar as etapas, executar cada etapa, agregar ingredientes para finalizar. A ordem é muito importante para o resultado final, até mesmo, sob pena de não dar certo.

Ex. Funcionamento da bike. Identificar componentes; papel de cada componente; interdependência das peças.

Ex. Criar um app – finalidade; interface; funcionalidades; pré-requisitos. Definição de componentes e etapas.

Ex. Artigo – o tema abordado; estrutura (introdução, meio e conclusão); conteúdo de cada tópico; textos de conexão.

A decomposição é muito importante, em primeiro lugar, para facilitar na construção do código; em seguida, após ele estar pronto, para a manutenção e alterações necessárias, pois serão realizadas apenas na parte específicas, não necessitando mexer em toda a estrutura.

PILARES: PADRÕES

Reconhecimento de padrões.

Modelo base de referência. Estrutura invariante. Com repetição.

Identificação de *Similaridades* e *Diferenças* entre objetos.

Para os seres humanos é natural o reconhecimento de padrões.

Busca generalizar, com objetivo de obter resolução para problemas diferentes.

Categorias / Classe – depende do domínio, tipo de mídia. **Classificar**.

Grau de similaridade, por grupos conhecidos x objetos desconhecidos (discriminar).

Já o computador usa a comparação. Precisa de uma representação de atributos. A máquina tem aprendizado por conceito associado ao objeto; através do armazenamento de dados; regras de decisão.

Abordagem para reconhecer padrões: extração de características; classificação de dados.

Diferentes Métodos e Diferentes Aplicações.

Ex. classificação de dados; reconhecimento de imagem; reconhecimento de fala; análise de cenas; classificação de documentos etc.

Ex. Machine learning; Ciências de Dados; Redes Neurais; Inteligência Artificial etc.

PILARES: ABSTRAÇÃO

Abstrair – observar um ou mais elementos, avaliando características e propriedades em separado.

Abstração – processo intelectual de isolamento de um objeto da realidade. Extrapolar um objeto do mundo concreto para o mundo das ideias.

Generalizar – tornar geral, mais amplo, extenso.

Na Lógica, *generalização* é a operação intelectual que consiste em reunir numa classe geral, um conjunto de seres ou fenômenos similares.

Classificar dados – características; pontos essenciais; generalizar em detrimento de detalhes.

O resultado é uma Representação de Dados.

Ex. Dados dos Estudantes, quais necessários para o banco de dados e quais não são (irrelevantes).

Na computação: algoritmos de busca binária; estrutura de dados (árvore, lista, grafos); máquinas de estado finito; e a própria linguagem de programação.

Ex. Abstração – limpar o terreno: fazer um plano, com as distâncias, determinar percurso, estender para outros cenários.

PILARES: ALGORITMOS

Computador - energia, trabalhador, eficiência, rapidez – mas não opera sozinho. Necessita de instruções para ele executar. Processamento de dados, através dos programas. Ele recebe, manipula e armazena dados. Programas são instruções para a máquina executar algo, pelo qual se espera determinado resultado.

Algoritmo – processo de resolução de problemas “step by step” (passo a passo) utilizando instruções: informa a máquina o que precisa ser feito e qual a ordem de execução.

Ele precisa ser entendido por humanos e pela máquina.

Desenvolvimento do programa:

Análise – estudo e definição dos dados de entrada e saída;

Algoritmo – descreve o problema por meio de ferramentas narrativas, fluxograma, ou pseudocódigo;

Codificação – utilização de linguagem de programação para codificar o algoritmo.

Sequência de passos com objetivo definido.

Execução de tarefas específicas.

Conjunto de operações que resultam em uma sucessão finita de ações.

Ex. preparar um sanduíche; trocar uma lâmpada; trajeto ao trabalho; fazer uma receita de bolo.

Sequência de ações sendo executadas, instruções passo a passo para concluir uma tarefa.

Como construir um algoritmo?

Compreender o problema – pontos mais importantes.

Definir os dados de entrada – dados fornecidos e cenário, contexto.

Definir processamento – cálculos e restrições.

Definir dados de saída – após o processamento.

Utilizar método de construção – ferramenta a utilizar.

Teste e diagnóstico – refinamento.

Narrativa – utilização da linguagem natural, sem conceitos novos, pode ser ambígua, dando diversas interpretações possíveis.

Fluxograma – estrutura gráfica, utiliza símbolos pré-definidos (necessita conhecimento prévio da estrutura e dos símbolos utilizados).

Pseudocódigo – Portugol (regras definidas passo a passo a serem seguidos).

Ex. Multiplicação de dois números: recebe os valores; multiplica; imprime resultado.

Ex. Média de alunos – recebe valores; operação das médias; regra de aprovação; imprime resultado.

ESTUDO DE CASO CONCEITUAL: PERDIDO

Pessoa perdida na floresta. Como utilizar o pensamento computacional?

Identificar os mecanismos

Recursos Comuns

Detalhes mais importantes

Sobrevivência: água, comida e abrigo. (decomposição em problemas menores)

Água – chuva ou nascente. (fogo)

Comida – coletar ou caçar. (fogo)

Abrigo – localização (mapa), proteção (fogo e lança), quente e seco (fogo).

#Utilizamos decomposição; reconhecimento de padrões; abstrações.

Determinar instruções passo a passo para cozinhar.

Pegar peixe, colocar água para ferver, limpar o peixe, fazer o cozido, assar o filé.

Ordem das atividades. Alguma pode ser realizada concomitantemente. Algumas são independentes das outras e podem ser realizadas ao mesmo tempo.

O pensamento computacional permite analisar e aplicar para qualquer cenário.

ESTUDO DE CASO APLICADO: SOMA DE UM INTERVALO

Ex. Soma de n^0 entre 1 e 200

1+2 1+3 1+4 1+5 ... Solução ineficiente

Outra forma: 200+1 199+2 198+3 197+4... Decomposição

Padrão: 200+1=201 199+2=201 198+3=201 197+4=201...

Valor se repete – Quantas vezes? $200/2=100$ logo $201 \times 100 = 20.100$

Soma entre x e y

$[x,y]$ – intervalo 1 e 200 $X+y =$ resultado parcial

$(y-1) + (x+1) =$ resultado parcial $y/2 =$ total ($200/2=100$)

Passo 1: Recebe os valores (x e y)

Passo 2: Resolva: $y/2 =$ total

Passo 3: Resolva: $y+x =$ resultado parcial

Passo 4: Ache o total: Final = total x resultado parcial

Passo 5: Imprime o resultado

ESTUDO DE CASO APLICADO: ADIVINHE UM NÚMERO

Determinar o número escolhido por uma pessoa dentro de um intervalo. Perguntas com respostas sim ou não.

O número é: 1? 2? 3? ... ineficiente.

Possível solução com número de tentativas muito menor

O número é maior que 50? Não.

O número é menor que 20? Sim.

Busca binária. Mais eficiente como método de busca.

1: Ordenar o vetor

2: Módulo de $L/2$

3: Acessar estrutura

4: Comparar valores

5: Repita até encontrar o número

6: Imprima "busca bem-sucedida"

Como aprimorar essa habilidade? Permitir que a pessoa explique suas decisões e seu processo de desenvolvimento.

AULA 2 – INTRODUÇÃO À LÓGICA DE PROGRAMAÇÃO

O que é lógica?

Um processo de pensamento atrelado ao conceito de algoritmos e resolução de problemas.

Resolução de Problema

Problema é uma questão que foge a uma determinada regra, é um desvio de percurso, que impede de atingir um objetivo com eficiência e eficácia.

Lógica é a parte da filosofia que trata das formas do pensamento em geral (dedução, indução, hipótese, inferência etc.) e das operações intelectuais que visam à determinação do que é verdadeiro ou não.

Organização coesa. Forma como desencadeiam acontecimentos. Forma de raciocínio. Ordenação que segue convenções.

Organização e planejamento das instruções, assertivas em um algoritmo, a fim de viabilizar a implantação de um programa.

Sequência de instruções, baseadas em ações, baseadas numa ação maior, que visa alcançar um objetivo.

Seres humanos conseguem prever comportamentos, computadores não, eles precisam de informações detalhadas.

Aprender a programar, se aprende a pensar como chegar a um objetivo.

Técnicas de Lógica de Programação

Ex. Construção – planejamento, com arquiteto faz a planta baixa; depois, um projeto de um engenheiro; em seguida um planejamento da execução do projeto. Seguem uma lógica e determinam as instruções.

Modelos de desenvolvimento e resolução

Técnica Linear: Modelo tradicional; não tem vínculo; tem estrutura hierárquica; utilizada na matemática, mas também em programação.

Execução sequenciada de operações. Recursos limitados de elementos. Única dimensão. Ordenação de elementos por uma única propriedade.

Técnica Estruturada: organização, disposição e ordem dos elementos essenciais que compõem um corpo (concreto ou abstrato).

Processamento de Dados – escrita programas, entendimento, validação, manutenção.

Tem mais de uma opção (não é linear). Tem escolhas.

Técnica Modular: partes independentes, controlada por um conjunto de regras.

Dados de entrada – processo de transformação – dados de saída (Modelo Padrão).

Metas – Simplificação, decomposição do problema, verificação por módulos.

Cada módulo tem suas regras.

AULA 3 – FUNDAMENTOS DE ALGORITMOS

Tipologia e Variáveis

Qual a função do computador? Processar informações: dados e instruções.

Dados são tratados e processados: numéricos, caracteres e lógicos.

Numéricos – inteiros (positivos ou negativos) e reais (todos, inclusive decimais).

Caracteres – letras, números, símbolos.

Lógicos – Booleano (verdadeiro “1” ou falso “0”), no português estruturado: “.Verdadeiro” ou “.V” ou “.S”; “.Falso” ou “.F” ou “.N”.

Variável – estrutura mutável, possui variações, incerto, inconstante, instável, pode receber mais de um valor. Sabe o tipo de dado que vai receber, mas não sabe qual é esse valor.

A variável vai identificar o tipo de dado.

Ela pode assumir qualquer um dos valores de um determinado conjunto de valores.

Ex. $a - b = d$ $a + b = c$

Regras ou práticas para o nome da variável:

- Atribuição de um ou mais caracteres;
- Primeiro uma letra – não número;
- Sem espaços em branco;
- Vedado a utilização de palavras reservadas;
- Pode conter caracteres e números.

Ex. x2; telefone, Z4, user12, Nome_usuario.

Papéis da variável:

Ação – modificação de estado do algoritmo.

Controle – vigiada a estrutura.

Há ainda dados que são fixos ou estáveis. Inalteráveis, que não mudam: um dado invariável, que constitui uma constante.

Instruções Primitivas

Instruções são operações que processam os dados.

Ações a serem executadas com os dados.

São cálculos matemáticos, com informações (inputs) de variáveis e de constantes.

Operadores são binários ou unários.

Ex. Área do círculo: $A = \pi \cdot \text{raio}^2$

Instruções são linguagem de palavras-chave (vocabulário) de uma determinada programação que tem por finalidade comandar um computador que irá tratar os dados.

Linguagens de programação: tem sua sintaxe própria, notação diferentes – janela, window ou ventana.

Entrada, processamento e saída dos dados.

Estruturas condicionais e operadores

Condição – estado de uma pessoa ou coisa.

Condicional – que expressa uma condição ou suposição; contém ou implica uma suposição ou hipótese.

Estrutura condicional – dada a condição, caso seja satisfeita, executa uma operação; caso seja uma inverdade, não satisfeita, acarreta uma exceção.

Estrutura Simples – apenas verifica se a condição foi satisfeita;

Estrutura Composta – verifica também a exceção, se não satisfeita;

Estrutura Encadeada – uma sucessão de condicionais.

Operadores relacionais

Igual a: =

Diferente de: <>

Maior que: >

Menor que: <

Maior ou igual: >=

Menor ou igual: <=

Se (<condição>) então

<instruções para condição verdadeira>

fim_se

Estrutura Condicional

Início programa:

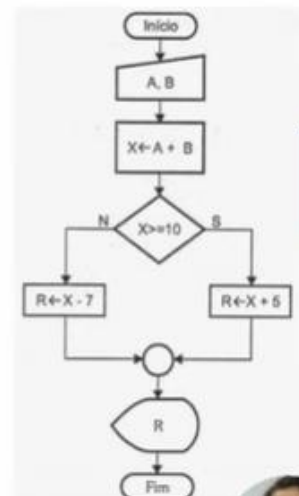
```
A = 0
B = 0
X = 0

leia A
leia B

X = A + B

se (X > 10)
    escreva X
Fim se

Fim programa
```



Fonte: livro de referência



Se (<condição>) então

<instruções para condição verdadeira>

Senão

<instruções para condição falsa>

fim_se

Estrutura Condicional

Condicional Encadeado

Se (<condição 1>) então

<instruções para condição verdadeira>

Senão

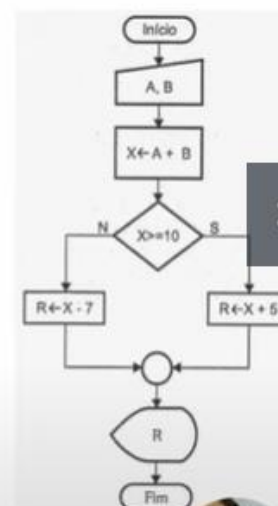
Se (<condição 2>) então

<instruções para: condição 2 verdadeira e condição 1 falsa>

Senão

<instruções para condição 1 e 2 falsas>

fim_se



Fonte: livro de referência



Operadores lógicos

AND – verifica as entradas, todas devem ser satisfeitas. Intersecção.

OR – precisa uma condição verdadeira. União.

NOT – de negação. Inversão do resultado lógico.

Utilizar para respostas simplificadas, como verdadeiro ou falso; substituição, encadeamento de condições.

Estruturas de Repetição

Trecho de um programa deve ser repetido, há laços, controle de fluxo, malhas de repetição, loop.

Condições de parada – para não ser um loop infinito. Número de repetições pré-fixadas, ou uma condição a ser satisfeita.

As vantagens são: redução de linhas, compreensão facilitada, redução de erro.

Ex. Enquanto ... faça Repita ... até Para ... de ... até ... faça

Ex. Grama alta. Enquanto tiver grama, cortar grama. Grama alta. Aparar grama. Analisar grama.

grama = Falso

Enquanto (grama == falso) faça

<instrução de cortar grama>

<atualiza grama>

fim enquanto

Ex. leitura livro. Repita, teste lógico no final.

Procurar artigo, virar a página, analisar conteúdo.

Ex. Para ... de ... até (for)

somatório = 0

para inicio = 1 até 10

somatório = somatório + inicio

Escreva somatório

fim para

É possível mesclar as estruturas uma dentro da outra.

Enquanto (<condição>)

Se (<condição2>)

 <instruções>

fim se

...

fim enquanto

Vetores e Matrizes

Vetor é uma sequência de tipos de dados.

Um vetor é caracterizado por uma variável dimensionada com tamanho pré-fixado.

Container ou Matriz unidimensional

Matriz é uma tabela organizada em linhas e colunas no formato $m \times n$, onde m representa o número de linhas (horizontal) e n o número de colunas (vertical).

Coleção de variáveis, contíguas em memória, índices.

Vetor conjunto [1...8]

Vetor[15]

Vetor = []

Serve para otimização, menor quantidade de linhas.

Funções

Subalgoritmo, Subprograma, Função, Bloco, Método, Sub-rotina.

Similar ao conceito de função matemática.

O elemento de um conjunto A está associado ao conjunto B.

São blocos de instruções que realizam tarefas específicas.

Decomposição do algoritmo, modular o problema.

Modularização do programa.

Código mais claro e conciso, mais legível.

Reutilização de instruções.

São blocos de instruções (código), identificados por nomes e parâmetros.

Arelados à função: Definição, Nome, Invocação, Variável local (são destruídas ao encerrar a função).

Dados são enviados para a função e ela retorna o resultado. Ela altera o estado do programa.

Instruções de Entrada / Saída

Conjunto de dados, para computador processar e retornar um resultado.

Entrada – Consiste na inserção e recebimento de dados do mundo real por meio de ação de alguma interface, seja teclado, mouse, arquivos, entre outros.

Saída – consiste na impressão de dados do mundo abstrato, digital por meio de ação de alguma interface. Os formatos podem variar desde simples arquivos binários até complexas queries de banco de dados.

Saída programada – condicional ou não. Aguarda um dispositivo para disparar o resultado.

Saída por interrupção – definida pelos periféricos.

Casos bem-sucedidos, erro de sintaxe, erros de programação, problemas com a interface.

LINGUAGEM DE PROGRAMAÇÃO

INTRODUÇÃO

Apresentar paradigmas existentes de programação.

As linguagens de programação são baseadas em um ou mais paradigmas: um paradigma ou multiparadigma.

A história da computação inicia para superar dificuldades, desenvolvendo processo de pensamento, baseado em pesquisas, estabelecendo fundamentos, e cada vez surgindo novos paradigmas.

Hardware e depois Software, pois o primeiro limita a capacidade do segundo.

O primeiro dispositivo de cálculo foi o ábaco, há mais de 3 mil anos. E no século XIX surgem alguns instrumentos computacionais. Na segunda guerra há uma ênfase na busca por instrumentos tecnológicos. Surge a álgebra booleana, que é a base para funcionamento computacional, mas que era apenas tese. Na década de 40 Ada Lovelace amplia o entendimento para a aplicação da álgebra booleana. Alan Turing e outros matemáticos, como Neuman, desenvolvem conceitos de algoritmo, um sistema formal, a ideia de inteligência artificial. A IBM traz grandes avanços na construção de máquinas computacionais com decisões baseadas em algoritmos.

Os PC surgem em 1975 com a Intel, é também quando é criada a linguagem basic da Microsoft. Em 1976 surge a Apple, e com primeiros PCs. Em 1985 surge o Windows 1.0, em 88 o Windows 2000 e em 2001 o XP. Até chegarmos aos sistemas que temos hoje.

Em 1949 surge a linguagem de máquina (computação). Década de 50 surgem linguagens de programação como o Cobol, Fortran, Lispe. Paradigmas a partir de 60 a 70 surge C, Machine Learning, Prolog. Já nos anos 90 alto nível JS, Java, C#, Python, Ruby. Novos conceitos nos anos 2000, GO, TS, outras.

Problemas computacionais são recorrentes. Objeto de discussão que possui instruções passo a passo que são mais facilmente resolvíveis em ambiente computacional (da sua época). Problemas de decisão, de busca, de otimização.

Problemas de decisão – caractere lógico: Sim ou Não. Ideia de pertencimento. Problema decidível.

Problemas de busca – relacionamento binário. Objetivo de busca: “x está em A?”. Recorrente em teoria de grafos.

Problemas de otimização – Maximizar ou minimizar uma função, recursos. Aperfeiçoamento.

Linguagem de programação

Método padronizado composto por um conjunto de regras sintáticas e semânticas de implementação de um código fonte. Conjunto de palavras e regras.

Código fonte – Tradução e Interpretação.

Como o computador entende o programa?

Há um processo de Tradução – dada a linguagem de alto nível, programa fonte, será executada a análise do programa, compilador, que traduz o que está sendo apresentado para linguagem de máquina, o programa objeto (baixo nível), o assembly.

Na Tradução há geração do programa objeto, para um interpretador, e depois a execução do programa objeto. Gera programas menores.

Já na Interpretação o programa fonte é executado diretamente. Processo mais lento, porém, com maior flexibilidade.

Características de um programa

Diretrizes:

Legibilidade – facilidade de leitura, compreensão, ortogonalidade (coerência nas instruções), definição adequada das estruturas.

Redigibilidade – simplicidade e facilidade de escrita do código, suporte à abstração, reuso do código, expressividade.

Confiabilidade – o que foi programado para fazer, verificação de tipos, trata exceções, uso de ponteiros, compatibilidade entre compiladores.

Custo – análise de impacto, eficiência, treinamento, codificação, compilação, execução, infra-estrutura.

Outras características

Atualizações; uso de IA; disponibilidade de novas ferramentas; comunidade ativa; adoção pelo mercado.

Análises de código

Léxica

Escaner ou leitura.

Elementos Lexos ou tokens, que são identificadores, palavras reservadas, números, strings. Que não são relevantes para a execução.

Particionar – Classificar – Eliminar.

Sintática

Componente do sistema linguístico que interligam os constituintes da sentença, atribuindo-lhe uma estrutura. Define a corretude do programa.

Padrão da gramática vai depender da linguagem de programação utilizada.

Semântica

É o estudo do significado. Incide sobre a relação entre significantes, como palavras, frases, sinais e símbolos.

Lógica do programa.

Paradigmas de programação

Forma de resolução de problemas com diretrizes e limitações específicas de cada paradigma utilizando linguagem de programação.

Regras para resolução de problemas, e está limitado por diretrizes, por contexto específico da linguagem utilizada.

Classificação:

Orientação à objeto – baseado na utilização de objetos e suas interações. Análogo ao mundo real.

Um objeto é descrito por características específicas (o que tenho), comportamentos (o que sou capaz de fazer) e estado (como faço). Que são os Atributos, Métodos e Estados (respectivamente).

Ponto de vista da programação o objeto há a Classe, que faz Alocação em memória e Operações Associadas. Estruturada conforme as variáveis.

Pilares – Herança (classe mãe e filha, que herdam comportamentos e estados específicos), encapsulamento, polimorfismo, abstração.

Com POO há o Reuso de código, que facilita na maioria dos casos.

Procedural – chamadas sucessivas e procedimentos separados (ideia de sequência).

Funcional – instruções são baseadas em funções.

Estruturado – blocos alinhados, sequência, decisão, iteração. Linguagem C. Teste lógico. Funções, laços, condições. Baixo nível, facilita o aprendizado. Problemas simples, específicos e diretos.

Computação distribuída – funções executadas de forma independente.

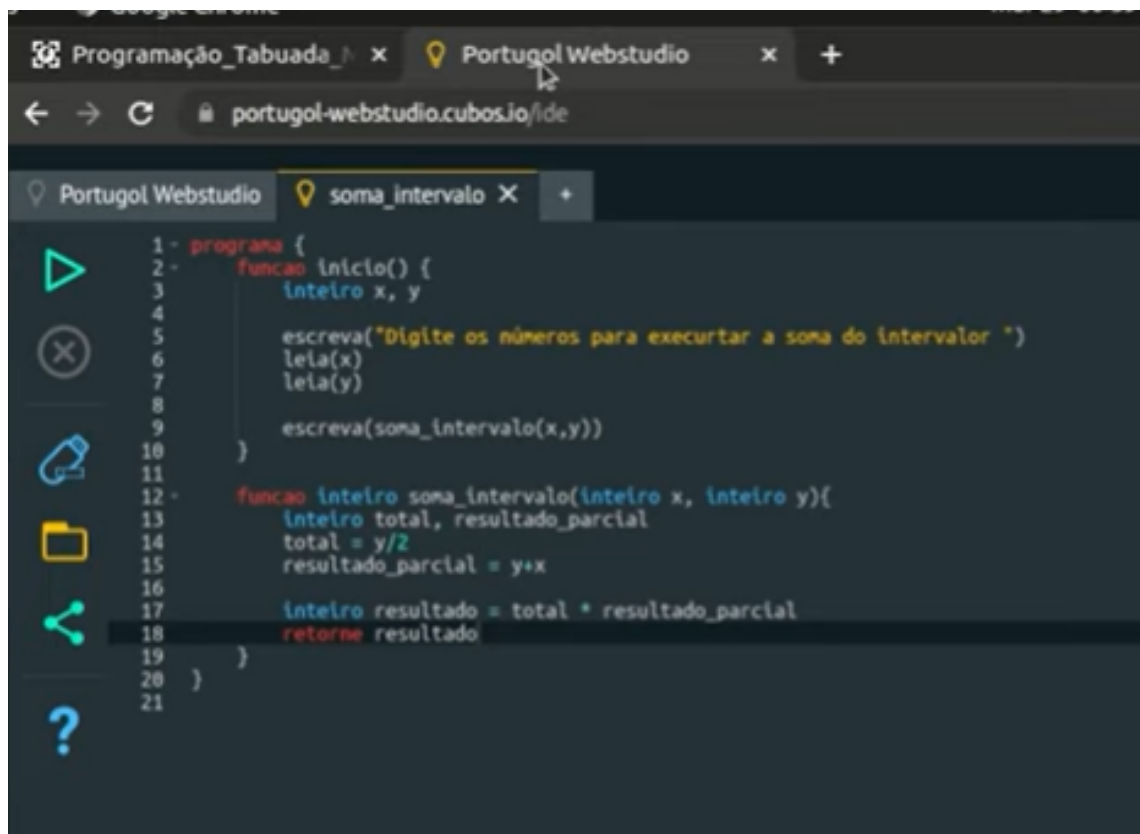
Lógico – também conhecido como restritivo. Muito utilizado em aplicações de IA. Ele chega nos resultados esperados a partir de avaliações lógico-matemáticas.

Deve analisar qual o melhor paradigma para resolução do problema.

PRIMEIRO CONTATO COM A PROGRAMAÇÃO

Algoritmos em Portugol

Portugol Web Studio



```
1 - programa {
2 -   funcao inicio() {
3 -     inteiro x, y
4 -
5 -     escreva("Digite os números para executar a soma do intervalo ")
6 -     leia(x)
7 -     leia(y)
8 -
9 -     escreva(soma_intervalo(x,y))
10 -  }
11 -
12 -  funcao inteiro soma_intervalo(inteiro x, inteiro y){
13 -    inteiro total, resultado_parcial
14 -    total = y/2
15 -    resultado_parcial = y+x
16 -
17 -    inteiro resultado = total * resultado_parcial
18 -    retorne resultado
19 -  }
20 - }
21 -
```

←

→

↻

portugol-webstudio.cubos.io/ide

Portugol Webstudio

soma_intervalo X

media_aluno X

+

▶

⊗

🔒

📁

↻

?

```
1 - programa {
2 -     funcao inicio() {
3         real a, b, nota_a, nota_b
4         escreva ("Digite as notas da p1 e p2 do aluno A ")
5         leia(a)
6         leia(b)
7         escreva("Digite as notas da p1 e p2 do aluno B ")
8         leia(nota_a)
9         leia(nota_b)
10
11         escreva("Média do aluno A: ", media_aluno(a, b))
12         escreva("\nMédia do aluno B: ", media_aluno(nota_a, nota_b))
13     }
14
15 -     funcao real media_aluno(real nota_a, real nota_b){
16         retorne (nota_a + nota_b + 1)/2
17     }
18 }
19
20
```

Digite as notas da p1 e p2 do aluno A 6

7

Digite as notas da p1 e p2 do aluno B 87.46

Média do aluno A: 6.5

Média do aluno B: 7.73

Programa finalizado.