Sistema Fiep

FIEP SESI SENAI IEL

LÓGICA DE PROGRAMAÇÃO

RESUMO DE ORIENTAÇÃO PARA ESTUDO

Prof. Anderson Luiz Menezes
Técnico em Desenvolvimento de Sistemas
Unidade Dr. Celso Charuri

ALGORITMOS

DEFINIÇÕES

Os algoritmos já fazem parte, naturalmente, do dia a dia das pessoas

Exemplos de algoritmos:

- Instruções para a administração de medicamentos;
- Indicações de como montar/usar um aparelho;
- Uma receita culinária.

DEFINIÇÕES

Mais formalmente...

"é um conjunto de regras formais para a obtenção de um resultado ou da solução de um problema."

FORBELLONE & EBERSPACHER, 2000.

DEFINIÇÕES

Segundo Dijkstra, um algoritmo corresponde a uma descrição de um padrão de comportamento, expresso em

termos de um conjunto finito de ações

Executando a operação *a* + *b* percebemos um padrão de comportamento, mesmo que a operação seja realizada para valores diferentes de *a* e *b*

DEFINIÇÕES

Portanto, um algoritmo é uma sequência ordenada de passos a ser seguida para a realização de uma tarefa

Computacionalmente, são passos criados a partir do **entendimento lógico** de um problema realizado por um programador

O objetivo é transformar esse problema em algo (um programa) que possa ser tratado e executado por um computador

ESTRUTURA

ENTRADA

↓

PROCESSAMENTO

↓

SAÍDA

Professor: Anderson Luiz Menezes

REPRESENTAÇÃO

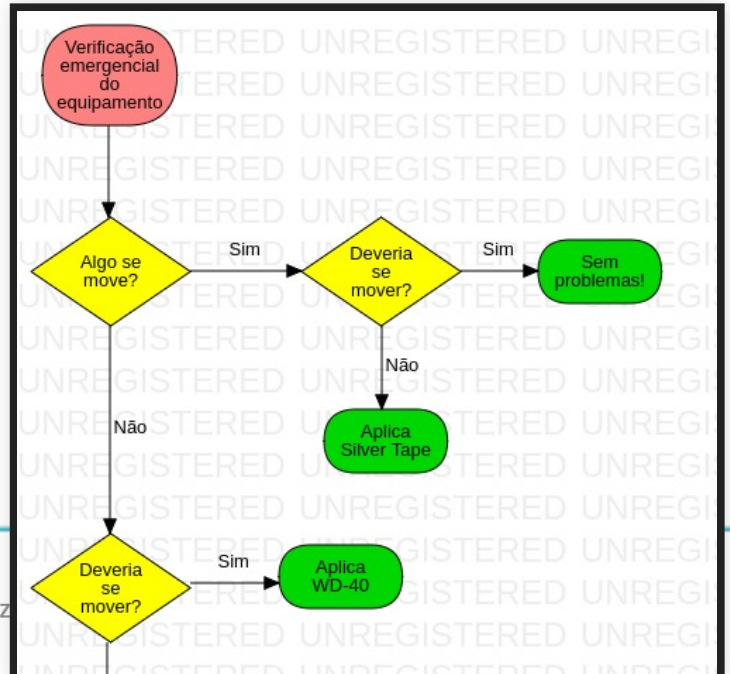
Linguagem natural: descrição textual da sequência de passos necessária para a resolução do problema

Pseudocódigo: descrição estruturada da sequência de passos

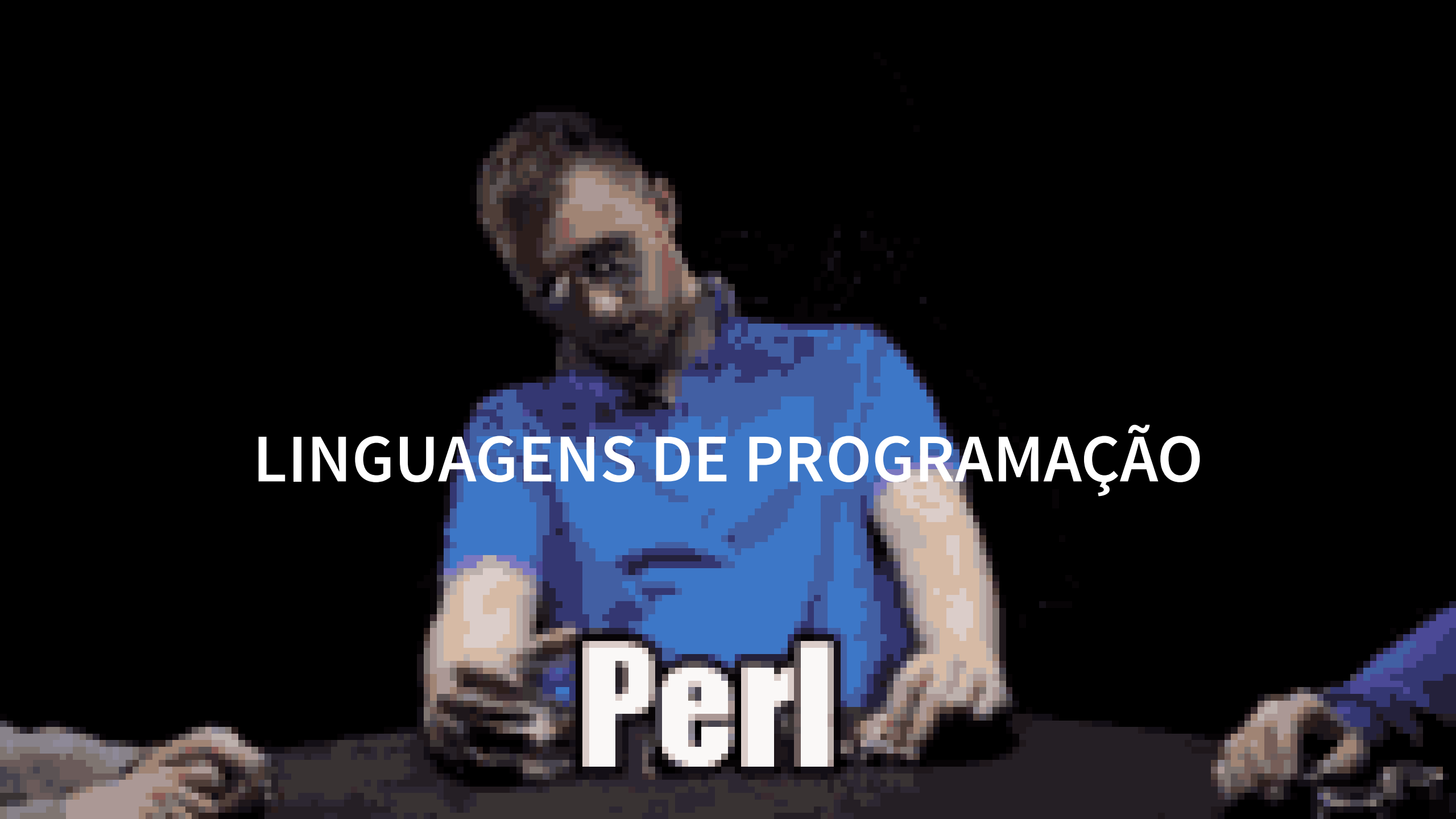
Fluxograma: diagrama de blocos com representação gráfica dos passos

REPRESENTAÇÃO

EXEMPLO DE FLUXOGRAMA



LINGUAGENS DE PROGRAMAÇÃO



DEFINIÇÕES

São técnicas de notação para programar, com a intenção de servir de veículo tanto para a expressão do raciocínio algorítmico, quanto para a execução de um algoritmo por um computador

Como escrever programas diretamente em bits seria inviável, as linguagens de programação têm o objetivo de ser compreendidas pelos humanos e executáveis pelo computador

DEFINIÇÕES

São métodos padronizados para expressar instruções para um computador

São conjuntos de regras sintáticas e semânticas usadas para definir um programa de computador

Ex.: Pascal, Cobol, Java, C, C++, C#, Ruby, Python...

TIPOS

Linguagem de máquina: binário

de difícil compreensão

Linguagem de montagem: Assembly

de difícil compreensão

há incompatibilidade entre tipos de processadores

Linguagens de programação

mais próxima da linguagem natural

TIPOS

Baixo Nível

mais próximas da linguagem de máquina

linguagens de microprocessador e montagem

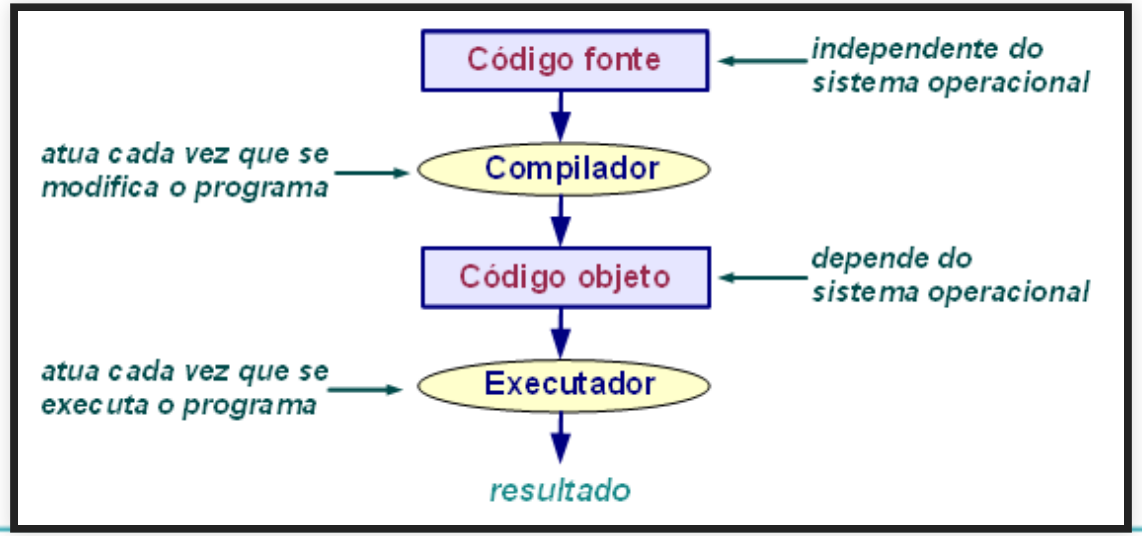
Alto Nível

mais próximas da linguagem natural

ex.: C, C++, Java, C#

PROCESSO

LINGUAGENS COMPILADAS



Ex.: C, C++

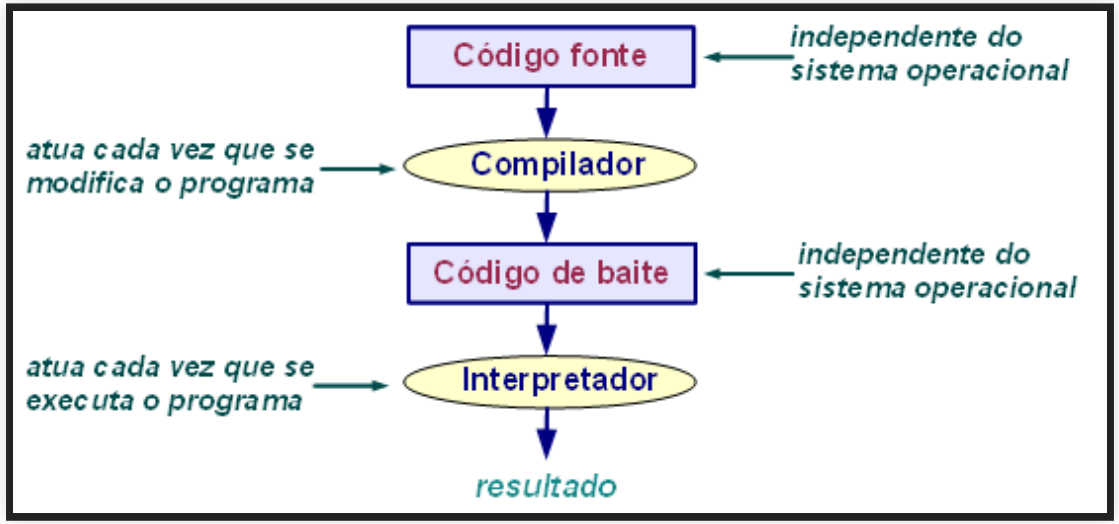
PROCESSO

LINGUAGEM INTERPRETADA

Ex.: Python

PROCESSO

LINGUAGEM PROCESSADA E INTERPRETADA



Ex.: Java

LÓGICA DE PROGRAMAÇÃO

CONTROLE DE FLUXO

ESTRUTURA SEQUENCIAL

Na estrutura sequencial os comandos de um algoritmo são executados numa sequência pré-estabelecida

Cada comando é executado somente após o término do comando anterior

A estrutura sequencial caracteriza-se, portanto, por um conjunto de comandos dispostos ordenadamente

ESTRUTURA SEQUENCIAL

EXEMPLO

```
...
instrucao_1
instrucao_2
instrucao_3
...
```


ESTRUTURAS DE DECISÃO

SE

Nesta estrutura, uma única condição (expressão lógica) é avaliada

Se o resultado desta avaliação for verdadeiro (V), **então** um determinado conjunto de instruções é executado

Senão, ou seja, se o resultado da avaliação for falso (F), um conjunto de instruções diferentes é executado

ESTRUTURA DE DECISÃO SE

EXEMPLO

```
se (condicao) entao
    instrucoes
senao
    instrucoes
fimse
```


ESTRUTURAS DE DECISÃO

ESCOLHA

Este tipo de estrutura é um generalização da estrutura **Se**

Na estrutura de decisão do tipo Escolha podem existir uma ou mais possibilidades de ações a serem tomadas

ESTRUTURA DE DECISÃO ESCOLHA

EXEMPLO

```
escolha(variavel)
    caso(valor_1): {instrucoes}
    caso(valor_2): {instrucoes}
        .
        .
        .
    caso(valor_n): {instrucoes}
    senao: {instrucoes}
fimescolha
```


ESTRUTURAS DE REPETIÇÃO

São muito comuns as situações em que se deseja repetir um determinado trecho de um programa certo número de vezes

Pode-se citar o caso em que se deseja realizar o mesmo processamento para um conjunto de dados diferentes

ESTRUTURAS DE REPETIÇÃO

Exemplo: o processamento da folha de pagamentos de uma empresa em que exatamente o mesmo cálculo é efetuado

para cada um dos funcionários

As estruturas de repetição são também denominadas **Laços** (***Loops***)

ESTRUTURAS DE REPETIÇÃO

ENQUANTO

Ao início da estrutura de repetição Enquanto, uma condição é testada

Se o resultado do teste for falso, então as instruções no seu interior **não** serão executadas e o fluxo segue normalmente para a instrução seguinte ao Fim Enquanto

ESTRUTURAS DE REPETIÇÃO

ENQUANTO

Se a condição for verdadeira, as instruções **serão** executadas e, ao seu término, retorna-se ao teste da condição

Assim, o processo será repetido **enquanto** a condição testada for verdadeira

ESTRUTURA DE REPETIÇÃO ENQUANTO

EXEMPLO

```
{iniciar variável controle}
enquanto (condicao) faca
    instrucoes
    {atualizar variável controle}
fimenquanto
```

ESTRUTURAS DE REPETIÇÃO

REPITA

Seu funcionamento é bastante parecido ao da construção Enquanto

A diferença é que, aqui, as instruções contidas no interior do laço serão sempre executadas pelo menos uma vez

ESTRUTURAS DE REPETIÇÃO

REPITA

Em seguida a condição é testada. Caso ela seja verdadeira, as instruções serão executadas novamente

Este processo é repetido até que a condição seja falsa, então a execução prossegue pela instrução imediatamente posterior ao fim da estrutura

ESTRUTURA DE REPETIÇÃO REPITA

EXEMPLO

```
{iniciar variável controle}
repita
    instrucoes
    {atualizar variável controle}
enquanto (condicao)
```

ESTRUTURAS DE REPETIÇÃO

PARA

Este tipo de estrutura é útil quando se conheçe previamente o números de vezes que se deseja executar determinado conjunto de comandos

O Para nada mais é que uma estrutura dotada de mecanismos para contar o número de vezes que o corpo do laço é executado

ESTRUTURA DE REPETIÇÃO PARA

EXEMPLO

```
para ({var} = {inicial} ate {final} passo {incremento}) faca
    instrucoes
fimpara
```

VETORES E MATRIZES

VETORES

São conjuntos de variáveis do mesmo tipo acessíveis por um único nome

Essas variáveis são armazenadas de forma contínua e ocupam as posições de forma fixa

Pode-se dizer que um vetor é uma **matriz unidimensional**

VETORES

Exemplo: considere uma sala de aula onde há 10 alunos e queremos armazenar suas idades em variáveis

- inicialmente precisaríamos criar dez variáveis diferentes para armazenar essas idades
- utilizando vetor, basta criarmos um único com dez posições

VETORES

Uma posição do vetor é sempre acessada pelo seu nome e por um índice

No exemplo acima teríamos, por exemplo, *idades[5]*, onde *idades* é o nome do vetor e *[5]* o índice (posição) que desejamos manipular

MATRIZES

Semelhante ao que foi dito sobre os vetores

A diferença é que as matrizes podem ser n-dimensionais

Pensando em uma matriz de duas dimensões (largura e altura), temos que considerar sua quantidade de linhas e colunas

Seguindo no exemplo da turma de 10 alunos, podemos usar uma matriz para armazenar as 4 notas de todos os alunos em uma única estrutura

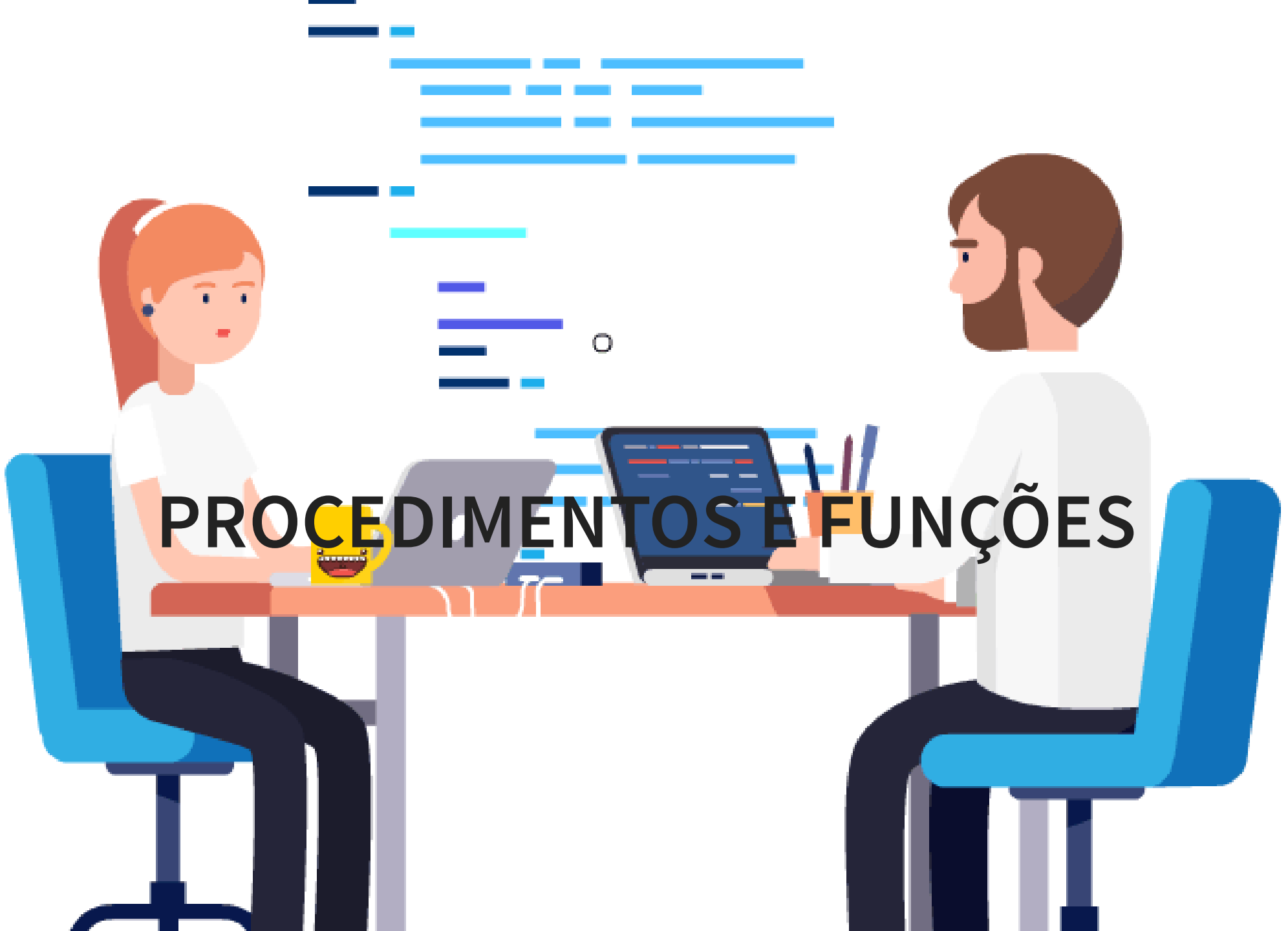
MATRIZES

Para isso precisamos de uma matriz com 10 linhas (uma para cada aluno) e 4 colunas (uma para cada nota)

Uma posição da matriz é sempre acessada pelo seu nome e por um conjunto de índices.

No exemplo acima, teríamos *notas[2][1]*, por exemplo, onde *notas* é o nome da matriz e *[2][1]* os índices da linha e coluna, respectivamente.

PROCEDIMENTOS E FUNÇÕES



PROCEDIMENTOS E FUNÇÕES

DEFINIÇÕES

Também conhecidos como sub-rotinas, são amplamente utilizados na modularização dos códigos

O objetivo da utilização dos procedimentos e funções é a centralização de um trecho de código que é comumente utilizado na implementação

Criando essas sub-rotinas evitamos a repetição desse trecho por toda a implementação

PROCEDIMENTOS E FUNÇÕES

DEFINIÇÕES

Os procedimentos e as funções têm exatamente a mesma aplicação

Com a diferença de que as **funções retornam o resultado do processamento** para quem a invocou, enquanto o **procedimento não retorna nada**

É comum, portanto, que ambos sejam chamados de funções: **função com retorno** e **função sem retorno**

PROCEDIMENTOS E FUNÇÕES

DEFINIÇÕES

As funções (e procedimentos) podem ou não receber **parâmetros**

Os parâmetros indicam as dependências dessas sub-rotinas. Ou seja, para que a sub-rotina efetue seu processamento, ela pode precisar de informações que não pertencem ao seu escopo

Essas informações são passadas através dos parâmetros

pass by reference

cup =

```
fillCup(      )
```

pass by value

cup =

```
fillCup(      )
```

PONTEIROS

DEFINIÇÕES

Um ponteiro (ou apontador) é um tipo especial de variável qie armazena um endereço

Ponteiros, na verdade, são referências para valores armazenados na memória

Se um ponteiro *p* armazena o endereço de uma variável *i*, podemos dizer que *"p aponta para i"* ou *"p é o endereço de i"*

DEFINIÇÕES

Ou ainda, em termos um pouco mais abstratos, dizemos que *"p é uma referência à variável i"*

Ponteiros são diretamente suportados, sem restrições, em linguagens como C e C++

São utilizados para construir referências, elementos fundamentais da maioria das estruturas de dados, especialmente aquelas não alocadas em um bloco contínuo de memória

```
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1):
```

RECURSIVIDADE

factorial(n) =

RECURSIVIDADE

DEFINIÇÕES

Muitos problemas têm a seguinte propriedade: cada instância do problema contém uma instância menor do mesmo problema

Dizemos que esses problemas têm *estrutura recursiva*

RECURSIVIDADE

DEFINIÇÕES

O objetivo dos algoritmos recursivos é, portanto, trabalhar com a redução de um problema até a sua instância mais simples

Essa instância mais simples é facilmente resolvida e serve de "guia" para a resolução das demais instâncias

RECURSIVIDADE

DEFINIÇÕES

O programador, portanto, só precisa mostrar como obter uma solução da instância original a partir da instância menor

O comportamento recursivo é implementado em funções que fazem chamadas a elas mesmas

RECURSIVIDADE

EXEMPLO

A solução de um problema por recursão é dada como segue:

```
Se for a instância mais simples então
    {resolva-a diretamente}
    {retorne o resultado}
Senão
    {reduza-a a uma instância menor}
    {tente novamente submetendo-a à mesma função}
Fim Se
```

Sistema Fiep

FIEP
SESI
SENAI
IEL

nosso i é de indústria.