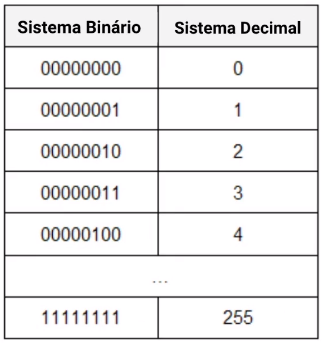
**Arquitetura de Computadores do Século XXI**

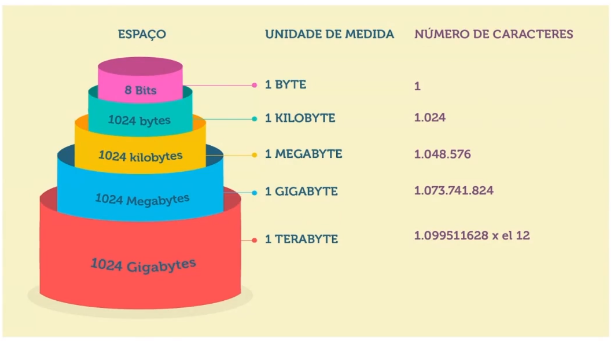
**Sistemas de Numeração**

Bit = “0, aberto” ou “1, fechado”.

Byte = 8 vezes o bit. Ex: A= 10010010, a= 11001100



**Grandezas de Armazenamento de Informação**



**Sistemas de Numeração**

A quantidade de algarismos disponíveis num sistema de numeração designa-se de **base**.

Alguns sistemas de numeração.

- Decimal (base 10): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

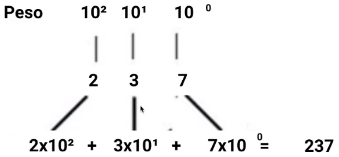
- Binário (base 2): 0, 1.

- Octal (base 8): 0, 1, 2, 3, 4, 5, 6, 7.

- Hexadecial (base 16): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

**Sistema Decimal**

No sistema decimal cada algarismo tem um valor posicional, ou seja, cada algarismo tem um peso de acordo com a sua posição na representação do valor.



**Sistema Binário**

É o sistema mais utilizado por máquinas, uma vez que os sistemas digitais trabalham internamente com dois estados (ligado/desligado, verdadeiro/falso, aberto/fechado). O sistema binário utiliza os símbolos: 0, 1, sendo cada símbolo designado por bit (**binary digit**).

**Sistema Octal**

É um sistema de numeração de base 8, ou seja, recorre a 8 símbolos (0, 1, 2, 3, 4, 5, 6, 7) para representação de um determinado valor.

**Sistema Hexadecimal**

Muito utilizado na programação de microprocessadores, especialmente nos equipamentos de estudo e sistemas de desenvolvimento. Utiliza os símbolos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 do sistema decimal e as letras A, B, C, D, E, F.

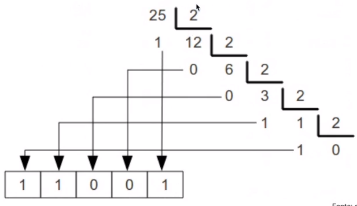
**Equivalências**: A= 10, B= 11, C= 12, D= 13, E= 14 e F= 15.

**Conversões de Base Decimal**

**Decimal > Binário**

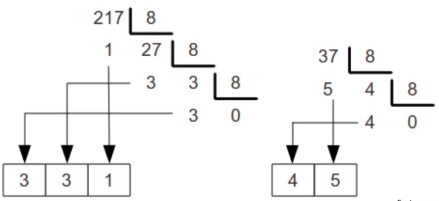
Dividir sucessivamente por 2 o número decimal e os quocientes que vão sendo obtidos, até que o quociente de uma das divisões seja 0.

O resultado é a sequência de baixo para cima de todos os restos obtidos.



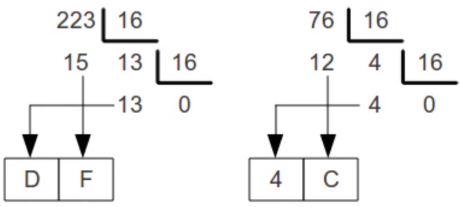
**Decimal > Octal**

Divisões sucessivas por 8. O resultado é a sequência de baixo para cima de todos os restos obtidos (a mesma coisa que binário).



**Decimal > Hexadecimal**

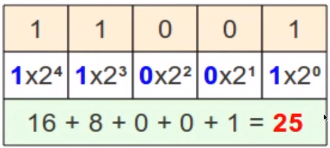
Divisões sucessivas por 16.



**Conversões de base binária, octal e hexadecimal**

**Binário > Decimal**

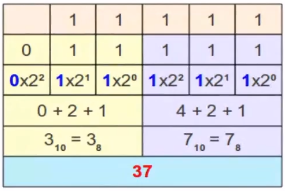
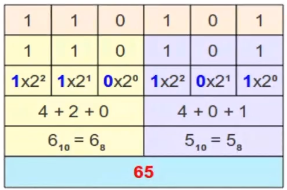
Basta calcular a soma de cada um dos dígitos do número binário multiplicado por 2 (que é a sua base) elevado à posição colunar do número, que, da direita para a esquerda começa com 0.



**Binário > Octal**

Separa-se os dígitos do número binário em grupos de 3 bits da direita para a esquerda. Em seguida transforma-se cada grupo individual de 3 bits em octal. Ao final, une-se os resultados:

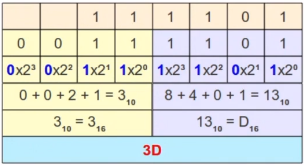
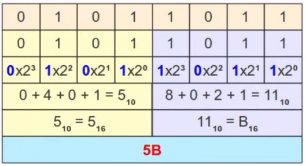
Caso o número de dígitos do número binário não seja múltiplo de 3, completa-se os dígitos à esquerda com zeros (0):



**Binário > Hexadecimal**

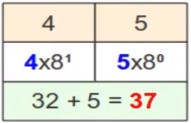
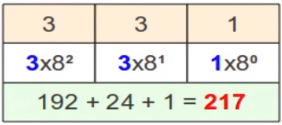
Separa-se o número binário em grupos de 4 bits, da direita para a esquerda. Em seguida, transforma-se cada grupo de 4 bits em hexadecimal. Ao final, simplesmente une-se os resultados em um só:

Caso o número de dígitos do número binário não seja múltiplo de 4, completa-se os dígitos à esquerda com zeros(0):



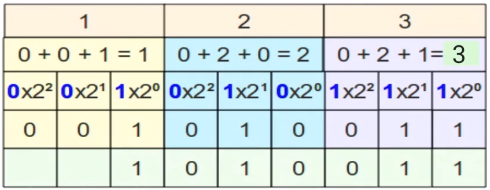
**Octal > Decimal**

Obtida através da soma dos dígitos do número octal multiplicados pela base 8 elevada à posição colunar do dígito, começando em 0 da direita para a esquerda:



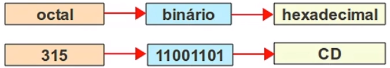
**Octal > Binário**

Decompõem-se o número octal diretamente em binários de 3 digitos. Os zeros mais à esquerda do resultado binário podem ser omitidos:



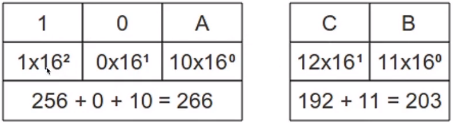
**Octal > Hexadecimal**

Transforma-se primeiro o octal em binário e em seguida o binário em hexadecimal:



**Hexadecimal > Decimal**

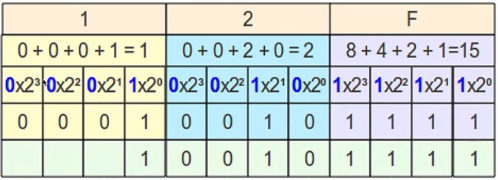
Realizada através da soma dos dígitos hexadecimais multiplicados pela base 16 elevada à posição colunar contando da direita para a esquerda, começando em 0, de forma semelhante à conversão de binários em decimais:



Note que os caracteres que definem os dígitos hexadecimais A, B e C foram substituídos pelos valores equivalentes em decimais 10, 11 e 12 de acordo com a tabela da lição anterios para a realização do cálculo.

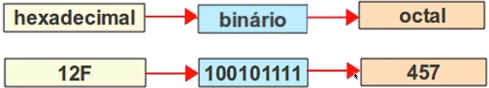
**Hexadecimal > Binário**

Decompõem-se o número hexadecimal diretamente em binários de 4 dígitos. Os zeros mais à esquerda do resultado binário podem ser omitidos:



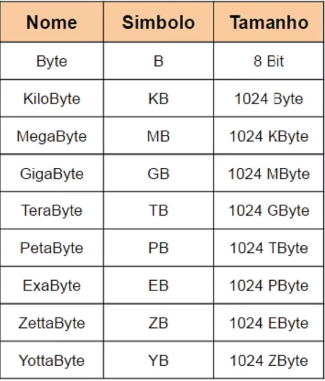
**Hexadecimal > Octal**

Transforma-se primeiro o hexadecimal em binário e em seguida o binário em octal:

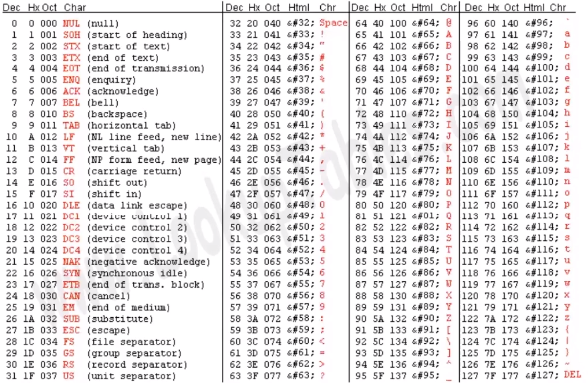


**Resumos**

**Grandezas**



**Tabela ASCII**



**# Conversor de bases:** <https://www.freetool.dev/pt/conversor-base-numerica>

**Algoritmos**

**Algoritmo**

É uma série ordenada de passos não ambíguos, executáveis. Um algoritmo é uma sequência de instruções ou comandos realizados de maneira sistemática com o objetido de resolver um problema ou executar uma tarefa.

Algoritmos são como uma receita de bolo: uma sequência de ações que devem ser executadas até que o objetivo final – o bolo pronto – seja atingido.

**Algoritmo clássico: Cubo Mágico**

Cubo de Rubik, também conhecido como Cubo Mágico, é um quebra-cabeça tridimensional, inventado pelo professor de arquitetura húngaro Ernõ Rubik em 1974

**Algoritmo clássico: Cubo Mágico**

Considere três hastes que servem de suporte a discos de diferentes tamanhos, os menores sempre sobre os maiores. Pode-se mover cada disco para qualquer haste desde que um disco maior nunca fique sobre um disco menor. O objetivo é transferir os três discos para uma outra base.

**Diferentes Estruturas**

Em um algoritmo ou mesmo programa de computador, são várias as estruturas que utilizam para controlar o fluxo de execução dos comandos e funções. Essas estruturas determinam o que será executado, como e quando será executado

**Eficiência de Algoritmos**

**Algoritmos Eficientes**

A análise de algoritmo fornece uma medida objetiva de desempenho proporcional ao tempo de execução do algoritmo.

1° facilidade de compreensão, codificação e depuração;

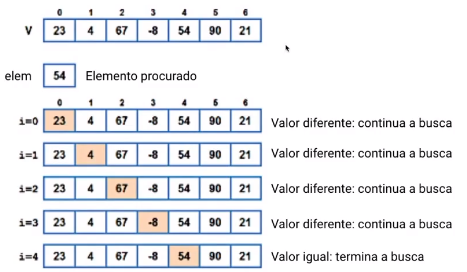
2° eficiência na utilização dos recursos do computador e rapidez, através de seu tempo de execução;

**Buscas e Ordenações**

Se você pensar bem, quase tudo que fazemos em termos de computação é busca de elementos em uma lista ou ordenação de elementos em uma lista! Métodos de **Busca** e **Ordenação** são a essência de muitos problemas e soluções em Computação.

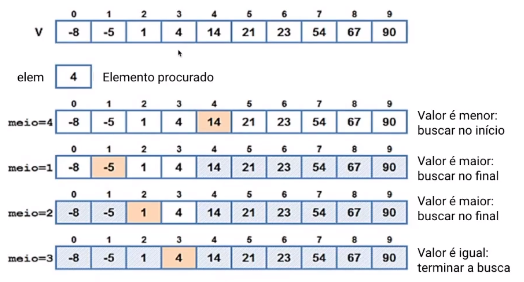
**Busca Sequencial**

A busca sequencial é um algoritmo pouco eficiente para encontrar um item em uma lista grande de elementos. No pior caso, todos os elementos devem ser visitados!



**Busca Binária**

A busca binária é um eficiente algoritmo para encontrar um item em uma lista ordenada de itens. Ela funciona dividindo repetidamente pela metade a porção da lista que deve conter o item, até reduzir as localizações possíveis a apenas uma.

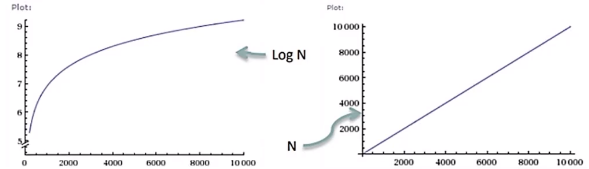


**Busca Sequencial x Busca Binária**

Se a lista não está ordenada, a busca binária não se aplica.

A busca sequencial é muito ineficiente para conjuntos grandes de valores.

Costuma-se dizer que a busca sequencial tem complexidade n e que a busca binária tem complexidade log n, fazendo muito menos comparações, torna-se mais eficiente.



**Linguagens de Programação**

**BASIC**

É uma linguagem historicamente importante que ajudou a popularizar a prática de programação. Alguns dos primeiros computadores pessoais vinham com a linguagem BASIC instalada no hardware convidando novos usuários a começar a programar. Várias derivações de BASIC estão ou estiveram disponíveis, entre elas SmallBasic, VisualBasic, entre outras.

**COBOL**

Muitos sistemas na área bancária foram escritos em Cobol e permanecem em uso até hoje. A formação em COBOL não é comum e por isso mesmo os programadores em COBOL costumam ser muito valorizados.

**PYTHON**

Machine Learning e Extração de Dados são implementados com Python.

**ASSEMBLY**

É a linguagem que leva à alta performance da máquina ainda em padrão legível pelas pessoas. É usada em partes de um programa muito sensíveis à performance. Encontrada em sistemas operacionais e “engine” de jogos, por exemplo.

**C**

Talvez a linguagem de programação mais importante no mundo. Sistemas operacionais tais como Windows, MacOs, iOs, e Android são escritos nela, bem como navegadores e “engine” de jogos. Influenciou dezenas de outras linguagens. Próxima da linguagem Assembly Language permite alta performance do software.

**JAVA**

Foi desenvolvido pela SUN Microsystems no início de 1990. É “orientada a objetos”, escalável e portável. Tem muitas semelhanças com C e C++. É a base de diversos sistemas como Android, possuindo uma comunidade forte e ativa na internet. Muitas empresas tem sistemas desenvolvidos em Java, fazendo com que a procura por profissionais que saibam a linguagem ainda seja expressiva.

**RUBY**

Sintaxe simples e fácil. Inspirada em linguagens como Perl, Smalltalk, Eiffel, Ada e Lis. Muito popular entre as startups, é famosa por ser usada em aplicações mundialmente reconhecidas, como Airbnb, Twitter e Github.

**Javascript**

Pode ser usada tanto no front quanto no back-end, sendo uma das linguagens mais versáteis que existem. É a linguagem majoritária para desenvolvimento Web e dificilmente um programador não terá contato com ela alguma vez na vida no mercado de trabalho.

**PHP**

É a linguagem mais famosa para criar backends de websites. Facebook e WordPress foram, em parte, escritas nela.

**SWIFT**

Desenvolvimento de softwares para a Apple.

**C#**

É portátil e simples. Desenvolvida e mantida pela Microsoft, é uma linguagem muito conhecida e fortíssima no cenário de desenvolvimento de jogos. Baseada nas linguagens C++, Java e Object Pascal.

**C++**

Semelhante à linguagem C.

Essa linguagem foi um esforço de se fazer uma linguagem fácil para a construção de grande projetos e ainda rápida e eficiente.

**Classificação de Linguagens**

**Alto Nível e Baixo Nível**



**Paradigma Imperativo Procedural**

Primeiro paradigma de programação baseado no modelo clássico de Von Neumann;

-Atribuições, sequências de comandos, laços de repetição e comandos condicionais fazem parte deste paradigma.

-Abstração procedural é sua principal característica.

Ex: C, Cobol, Fortran, Pascal, Ada, etc.

**Paradigma Imperativo Orientado a Objetos**

Um programa é constituído de vários objetos que trocam mensagens entre si.

-Objetos de dados e não passivos como no paradigma imperativo;

-Definição de classes de objetos, herança e passagem de mensagens caracterizam este paradigma;

Ex: C++, C#, Java, SmallTalk, etc.

**Paradigma Declarativo Funcional**

O problema é modelado por um conjunto de funções matemáticas, cada uma com um espaço de entrada e um resultado, usada tradicionalmente em IA.

-As funções interagem entre si, utilizando a composição funcional;

Ex: LISP, Haskell, ML, etc.

**Paradigma Declarativo Lógico**

Declarativo – descreve o problema e o que se deseja fazer e não como fazer (programação imperativa e OO).

-Conhecida como baseada em regras;

Ex: Prolog.

**Quanto à Estrutura de Tipos**

-Fracamente tipada: Tipo das variáveis não é definido, podendo modificar-se. Ex: PHP.

-Fortemente tipada: Tipo das variáveis é predefinido e imutável. Ex: Java, Ruby.

-Dinamicamente tipada: Tipo das variáveis é definido em tempo de execução. Ex: Python, Ruby.

-Estaticamente tipada: Tipo das variáveis é definido em tempo de compilação. Ex: Java, C.

**Quanto ao Grau de Abstração**

O grau de abstração funciona como uma escala para linguagens: quanto mais abaixo mais próximo da linguagem de máquina, e quanto mais alto, mais próxima da linguagem dos seres humanos.

-Baixo nível: Possui símbolos que representam o código de máquina propriamente. Ex: Assembly.

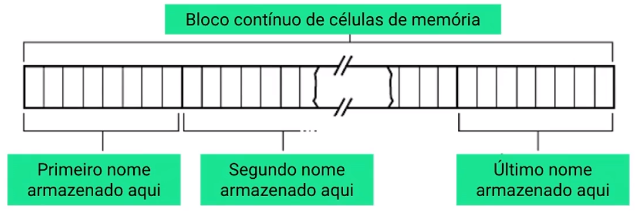
-Médio nível: Possui símbolos que podem ser diretamente traduzíveis para código de máquina, mas também possui símbolos que precisam ser processados por um compilador. Ex: C#.

-Alto nível: Possui símbolos complexos que precisam de interpretação de um compilador antes de serem transformados em linguagem de máquina. Ex: Java, Javascript, Python, Ruby.

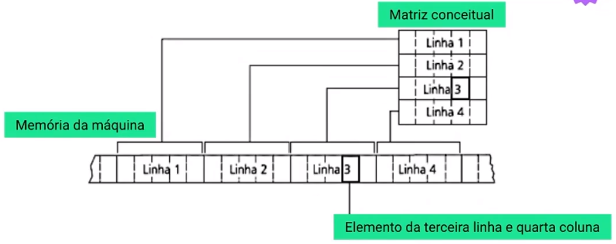
**Abstração de Dados**

A forma como o usuário ou programador lida com os dados não corresponde à forma como os dados estão realmente armazenados na memória do computador ou como eles são realmente.

Chamamos de abstração de dados, estruturas muito úteis que “mascaram” os dados para visualizar e manipular dados de forma a tornar as soluções de uso e programação mais intuitivas.

**Vetor (array) ou lista contígua**

É uma lista linear na qual a ordem lógica dos elementos (a ordem “vista” pelo usuário) é a mesma ordem física (em memória principal) dos elementos. Isto é, elementos vizinhos na lista estarão em posições vizinhas de memória.

**Matriz (4l x 5c) Armazenada Segundo a Sequência de Linhas**

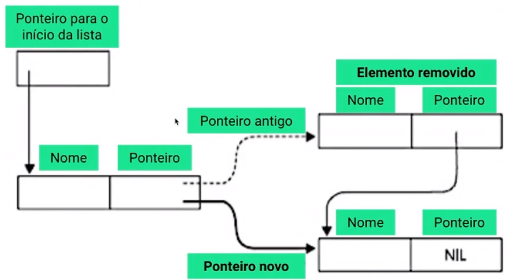
Matriz é uma tabela organizada em linhas e colunas no formato m x n, onde m representa o número de linhas (horizontal) e n o número de colunas (vertical). A função das matrizes é relacionar dados numéricos.

**Listas Ligadas**

É uma estrutura que corresponde a uma sequência lógica de entradas ou nós.

Em uma lista ligada há um ou dois pontos conhecidos de acesso – normalmente o topo da lista (seu primeiro elemento) e, eventualmente, o fim da lista (seu último elemento). Cada nó armazena também a localização do próximo elemento na sequência, ou seja, de seu nó sucessor. Desse modo, armazenamento de uma lista não requer uma área de contígua de memória.

Nesta estrutura os nomes não ocupam uma área contígua de memória, por isso cada elemento da estrutura tem o dado (nome) e um apontador para o próximo nome.

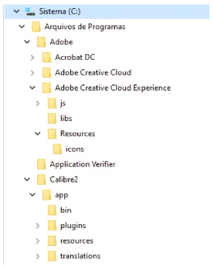


**Lista Ligada: Eliminando um Elemento**

Observe que a remoção de um elemento implica rearranjo de ponteiro e só.

**Lista Ligada: Inserindo um Elemento**

O mesmo acontece na inserção, só mudança de ponteiros.

**Árvores e Exemplos**

**Aplicação**

Na computação, o uso das estruturas em árvore é comum:

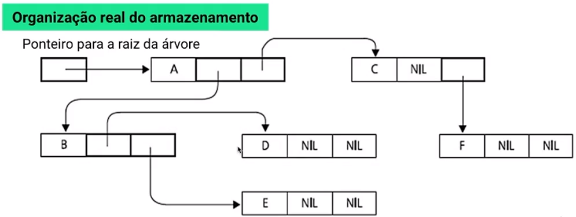
**Árvores**

Em uma árvore, a organização dos elementos se dá de forma hierárquica, existindo um elemento que fica no topo da árvore, chamado de raiz e os elementos subordinados a ele, os nós filhos.

Cada nós filho pode conter zero, um ou mais de um nó filho.

Estruturas em árvore possibilitam algoritmos mais eficientes!

**Árvore Implementada por um Sistema de Listas Ligadas**



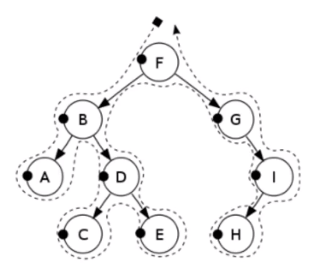
**Árvore de Pesquisa Binaria**

É uma estrutura de dados útil quando precisam ser tomadas decisões bidirecionais em cada ponto de um processo.

Um exemplo: Encontrar todas as repetições numa lista de números. Uma forma para fazer isto é comparar cada número com todos que o precedem.

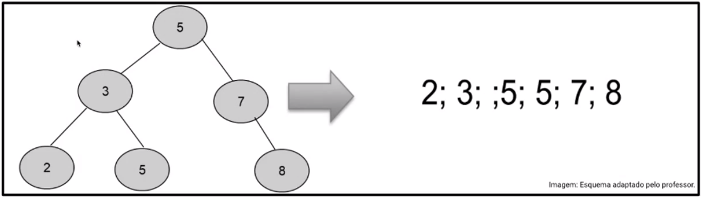
**Percursos em Árvore Binária**

Veja na imagem como algoritmos PreOrdem, InOrdem, PosOrdem geram saídas diferentes!



**Exemplo: Percorrendo uma Árvore em Ordem**

A chave da raiz de uma sub-árvore é impressa entre os valores de sua sub-árvore esquerda e sub-árvore direita.



**Pilha**

São estruturas de dados do tipo LIFO (last-in-first-out), onde o último elemento a ser inserido, será o primeiro a ser retirado.

Assim, uma pilha permite acesso a apenas um item de dados – o último inserido. Para processar o penúltimo item inserido, deve-se remover o último.

São muito importantes quando precisamos armazenar um conjunto de dados para ser utilizado em um determinado software.

**Fila**

São estruturas de dados do tipo FIFO (first-in first-out), onde o primeiro elemento a ser inserido, será o primeiro a ser retirado, ou seja, adiciona-se itens no fim e remove-se do início.

**Engenharia de Software**

Uma disciplina que reúne metodologias, métodos e ferramentas a serem utilizadas desde a percepção de um problema até o momento em que o sistema desenvolvido deixa de ser operacional, visando resolver problemas inerentes a processo de desenvolvimento e ao produto de software.

Visando: - baixar custo.

- produção mais rápida.

- melhoria da qualidade.

**Requisitos**

Correspondem ao conjunto de necessidades do usuário em relação ao sistema. Requisitos podem ser funcionais (o que o sistema deveria fazer) e não funcionais (restrições, isto é, o que o sistema não deveria fazer).

**Extração de Requisitos**

É uma das tarefas mais desafiadoras na área de desenvolvimento de sistemas, envolve várias técnicas tais como entrevistas, formulários, workshops, documentação, análise de cenários, etc. A extração de requisitos pode resultar em um documento de requisitos definido pelo padrão IEEE 830.

**Padrão IEEE830**

#Requisitos Relacionados com Interfaces Externas

Interfaces com o Usuário

Interfaces com Hardware

Interfaces com Outros Sistemas de Software

Interfaces de Comunicação

#Requisitos Funcionais

Requisito Funcional 1\*

Requisito Funcional 2\*

...

#Requisitos de Desemprenho

#Requisitos de Projeto

#Outros Requisitos

**Técnicas de Levantamento de Requisitos: Entrevistas**

É a mais utilizada por ser simples e de baixo custo;

É recomendado que o entrevistador prepare as perguntas antecipadamente;

Pode ter uma longa duração;

É difícil que o entrevistador fique “neutro”, ou seja, não utilize sua experiência para antecipar uma solução e induzir o usuário e aceita-la.

**Protótipos**

É uma aplicação ou sistema em fase de teste.