



**ANHANGUERA EDUCACIONAL**

**UNIDADE SANTANA**

**Curso de Tecnologia em Análise e Desenvolvimento de Sistemas**

**Thiago Melo da Silva – RA: 401966816563**

**Luis Gustavo do carmo Finoti – RA: 401829116563**

**João Victor Martins da Silva – RA: 402000516563**

**João Victor Souza de Lucena – RA: 402278816563**

**Julio César Cesario – RA: 401850316563**

**Projeto Interdisciplinar Aplicado aos Cursos Superiores de Tecnologia**

**São Paulo**

**2023**

**Thiago Melo da Silva – RA: 401966816563**

**Luis Gustavo do carmo Finoti – RA: 401829116563**

**João Victor Martins da Silva – RA: 402000516563**

**João Victor Souza de Lucena – RA: 402278816563**

**Julio César Cesario – RA: 401850316563**

**Projeto Interdisciplinar Aplicado aos Cursos Superiores de Tecnologia I**

Projeto Interdisciplinar apresentado ao curso de Análise e desenvolvimento de sistemas da Universidade Anhanguera como requisito parcial à obtenção de nota para aprovação da disciplina de Projeto Interdisciplinar aplicado aos cursos de Tecnologia Análise e Desenvolvimento de Sistemas.

Orientador: Prof. Márcio Joaquim

**São Paulo**

**2023**

**resumo**

O trabalho explorará os fundamentos críticos de Sistemas Computacionais, mergulhando nos intricados tópicos dos Sistemas Numéricos, que envolvem conceitos, simbologia e a representação de bases numéricas. A compreensão avançada de Álgebra Booleana e Lógica Digital será abordada em detalhes, destacando sua importância na estruturação e funcionamento dos sistemas.

Além disso, será oferecida uma introdução prática à linguagem de programação Python, evidenciando sua versatilidade e aplicabilidade. Esta seção será enriquecida com exemplos concretos e casos de uso para ilustrar como o Python se tornou uma ferramenta fundamental no desenvolvimento de soluções computacionais.

Na continuidade, focaremos em análise e modelagem de dados, explorando técnicas e metodologias essenciais para compreender e representar informações de maneira eficaz. Imagens elucidativas e exemplos práticos serão incorporados para facilitar a compreensão e aplicação desses conceitos no mundo real. Este trabalho busca oferecer uma visão abrangente e prática desses temas cruciais, contribuindo para uma compreensão sólida dos fundamentos dos sistemas computacionais.

**Palavras Chave:**Fundamentos críticos, Sistemas Computacionais, Sistemas Numéricos, Conceitos, Simbologia, Representação de bases numéricas, Álgebra Booleana, Lógica Digital, Introdução prática, Linguagem de programação Python, Versatilidade, Aplicabilidade, Análise de dados, Modelagem de dados, Técnicas, Metodologias, Informações eficazes, Exemplos práticos, Compreensão, Fundamentos dos sistemas computacionais.para quem deseja se envolver no campo da tecnologia da informação e desenvolvimento de software.

**Sumário**

[**1 INTRODUÇÃO 5**](#_heading=h.3znysh7)

[**2 Fundamento de sistemas computacionais 6**](#_heading=h.8xqglrz61e8p)

[**2.1 Desenvolvimento histórico 6**](#_heading=h.tyjcwt)

[**2.2 Conceitos básicos de arquitetura e organização de computadores 10**](#_heading=h.1t3h5sf)

[**2.3 Estrutura básica de um computador 11**](#_heading=h.2s8eyo1)

[**2.4 Hierarquia de níveis de computador 12**](#_heading=h.6u495ako36b3)

[**2.4.1 Memória: 12**](#_heading=h.8e4821tsqx2d)

[**2.4.2 Processamento: 13**](#_heading=h.t53fht9lpcyf)

[**2.4.3 Software: 13**](#_heading=h.h3ce62xdu1nv)

[**2.4.4 Redes: 13**](#_heading=h.bjnoovmnt6od)

[**2.4.5 Armazenamento de dados: 13**](#_heading=h.ksmnmts7p3h6)

[**2.4.6 Linguagens de Programação: 13**](#_heading=h.t2kfgyb0dg1w)

[**3. Componentes básicos de um computador 15**](#_heading=h.3rdcrjn)

[**3.1 Unidade central de processamento (cpu) 16**](#_heading=h.dacpnh9t5y0z)

[**3.2 Memória principal 16**](#_heading=h.wouon9bj6lb1)

[**3.3 Memoria secundaria 17**](#_heading=h.59feecf4vnsw)

[**3.4 Dispositivos de entrada e saída 17**](#_heading=h.2kgpgxpwi5zf)

[**3.4.1 Dispositivo de entrada 18**](#_heading=h.1sxlpt5fys9m)

[**3.4.2 Dispositivo de saída: 18**](#_heading=h.4zbyxl2b19eh)

[**3.4.3 Dispositivos híbridos: 18**](#_heading=h.cr8ci5t3rmrw)

[**4. Sistemas numéricos: conceitos, simbologia e representação de base numérica. 19**](#_heading=h.cq314em311)

[**4.1 Conversão entre bases numéricas: decimal 20**](#_heading=h.n61xmubiz2e5)

[**4.1.1 Conversão para Binário: 21**](#_heading=h.yz3byxg1nzhi)

[**4.1.2 Conversão para octal: 21**](#_heading=h.10shtwrmrii5)

[**4.2 Conversão entre bases numéricas: binárias 22**](#_heading=h.sxojsmqakjp)

[**4.3 Conversão entre bases numéricas: octal 23**](#_heading=h.l8pkuto7q7vz)

[**4.3.1 Conversão de octal para binário 24**](#_heading=h.q26c31884bmf)

[**5 Álgebra booleana: a base de lógica digital 25**](#_heading=h.bhzn9mwgn0qt)

[**5.1 Principais operações lógicas 25**](#_heading=h.dx9dofoiqvkx)

[**5.2 Lógica digital 25**](#_heading=h.r7x5tsxc2yf9)

[**5.3 Expressões lógicas 26**](#_heading=h.2ffsimtp29p0)

[**5.4 Portas lógicas 26**](#_heading=h.bkrvtnj8tjop)

[**5.5 Introdução a circuitos 27**](#_heading=h.9ktcpj1542kt)

[**6 Introdução a linguagem python 28**](#_heading=h.k6b550r2v6wc)

[**6.1 O que é uma estrutura de repetição 29**](#_heading=h.agcoh5wqz4bw)

[**6.2 Estruturas condicionais e lógicas 32**](#_heading=h.xwob5wso9jsj)

[**7.0 Estrutura de dados em python 38**](#_heading=h.uiki89ds8pvr)

[**7.0.1 Estruturas contíguas: 39**](#_heading=h.2uga06yuipvt)

[**7.0.2 Estruturas ligadas 39**](#_heading=h.m8ah2bqtjegt)

[**7.1 Algoritmos de busca 39**](#_heading=h.43jf3hju97aa)

[**7.1.1 Busca sequencial 40**](#_heading=h.1hdooicem8h9)

[**7.1.2 Busca binária 41**](#_heading=h.79mcoa7ttux0)

[**7.2 Algoritmo de Ordenação 42**](#_heading=h.c5exkt1d9e5n)

[**7.2.1 Bubble Sort (Ordenação por Bolha): 42**](#_heading=h.wjfqrj77ei8b)

[**7.2.2 Insertion Sort (Ordenação por Inserção): 42**](#_heading=h.wvj2ceqf0nzp)

[**7.2.3 Selection Sort (Ordenação por Seleção): 43**](#_heading=h.lurjvpnhqaw)

[**7.2.4 Merge Sort (Ordenação por Mesclagem): 43**](#_heading=h.3x706goqkg0s)

[**7.2.5 Quick Sort (Ordenação Rápida): 43**](#_heading=h.lg8j1qna335)

[**7.2.6 Heap Sort (Ordenação por Heap): 43**](#_heading=h.celkxme51zwj)

[**7.2.7 Timsort: 43**](#_heading=h.byu7xnqbkil2)

[**7.3 Listas 44**](#_heading=h.sx4il7who2na)

[**7.3.1 Tuplas 45**](#_heading=h.tqflfsp2wl6o)

[**7.3.2 Dicionário 47**](#_heading=h.n2fehl6dtcy2)

[**7.3.3 Sets 49**](#_heading=h.oyqvo1romr90)

[**8 Python Orientado à Objetos 51**](#_heading=h.tuntcp2onywm)

[**8.1Bibliotecas, Módulos e Pacotes em Python 51**](#_heading=h.agdrngx24tp4)

[**8.1.1 Exemplos de Bibliotecas Populares: 52**](#_heading=h.t0tijhltfwax)

[**8.2Aplicação de Banco de Dados com Python 55**](#_heading=h.e2yb7ij1mg79)

[**8.3 Princípios de POO 56**](#_heading=h.xlyz7w5y90qz)

[**8.3.1 Classes 56**](#_heading=h.irjbsm3f4dfv)

[**8.3.2 Objetos 58**](#_heading=h.ugpwhbndldki)

[**8.3.3 Encapsulamento 59**](#_heading=h.9h84y9fewt41)

[**8.3.4 Herança 61**](#_heading=h.h4tx0yqebxya)

[**8.3.5 Polimorfismo 62**](#_heading=h.v47xdast47n7)

[**9 Introdução à análise de dados com Python 66**](#_heading=h.84c33ijn7crw)

[**9.0.1 Coleta de dados 66**](#_heading=h.vrn42slu53x5)

[**9.0.2 Limpeza e preparação de dados 67**](#_heading=h.oymo4g12r0ec)

[**9.0.3 Exploração de dados 68**](#_heading=h.lxk76rinseqr)

[**9.0.4 Análise de dados 69**](#_heading=h.g18mw2p56squ)

[**9.0.5 Comunicação de resultados 69**](#_heading=h.ej6gb7m9rmbg)

[**9.1 Introdução a biblioteca Pandas. 70**](#_heading=h.2vsq05fm5wje)

[**9.2 Introdução a Scikit Learning 74**](#_heading=h.kb4wnp4h6ahj)

[**9.3 Visualização de dados 76**](#_heading=h.k1jh1n887ib)

[**10 Engenharia de sotfware 77**](#_heading=h.oah0x82u1wcs)

[**10.0.1 Análise e desenvolvimento de sistemas 80**](#_heading=h.78xlpzpxbxrd)

[**10.1 Fundamentos de engenharia de software 81**](#_heading=h.q7v4hh58x3ms)

[**10.2 Modelos de projeto de software 85**](#_heading=h.hmncai5jbski)

[**10.3 O processo do software 88**](#_heading=h.5fmorbyb5wwd)

[**11 Processo de negócios para análise de sistema 90**](#_heading=h.b4ffie6o9mt9)

[**11.1 Fundamentos de Processos de Negócios 93**](#_heading=h.mr9bmhht1lj9)

[**11.2 Gerenciamento de processos de negócios 93**](#_heading=h.7ycvvzhitwkc)

[**11.3Modelagem de processos de negócios 94**](#_heading=h.1d7qrbteb2sk)

[**12 Engenharia de requisito 97**](#_heading=h.d6gbjoc7aio7)

[**12.1Tipos de Requisitos 98**](#_heading=h.bmeopyxg6l9i)

[**12.2 Levantamento 100**](#_heading=h.sas9ic94wsc2)

[**12.3 Análise 100**](#_heading=h.o7h1rfi7mthr)

[**12.4 Documentação 100**](#_heading=h.kg8kddtdua0t)

[**12.5 Verificação, Validação e Garantia de Qualidade 101**](#_heading=h.h8qtkuxorisl)

[**12.6 Gerência 102**](#_heading=h.8jcmrfi632zi)

[**13 Paradigma orientado à objetos 102**](#_heading=h.o3q7a7cqhkc2)

[**13.1 Fundamentos da orientação à objetos 103**](#_heading=h.6t36tym8gnwu)

[**13.2Métodos orientado à objetos 104**](#_heading=h.2moj10ch55qd)

[**13.3Processo Unificado 105**](#_heading=h.3iu81tskl0hi)

[**CONSIDERAÇÕES FINAIS 107**](#_heading=h.26in1rg)

[**REFERÊNCIAS 108**](#_heading=h.lnxbz9)

[**ANEXOS 115**](#_heading=h.35nkun2)

### 1 INTRODUÇÃO

No artigo abaixo está discorrido sobre assuntos referentes a Linguagem de programação, Arquitetura e organização de computadores e análise e modelagem de sistemas, como:

-Fundamentos de sistemas computacionais

-Componentes básicos de um computador

-Sistemas numéricos: conceitos, simbologia e representação de base numérica

-Álgebra booleana

-Introdução a linguagem de python

-Estrutura de dados em python

-Python orientado a objetos

-Introdução à análise de dados python

-Introdução à engenharia de software e à análise de sistemas

-Processo de negócios para análise de sistemas

-Engenharia de requisitos

-Paradigma orientado a objeto

Tendo como objetivo embasar de forma simples todos esses temas dentro das nossas pesquisas com embasamento em diversos estudos e citações

# 

### 2 Fundamento de sistemas computacionais

Fundamentos de sistemas computacionais se referem aos princípios essenciais que constituem o funcionamento dos computadores e sistemas de processamento de informações. Alguns dos principais pontos incluem:

Hardware e Software: Os sistemas computacionais são compostos por hardware (componentes físicos, como processadores, memória, dispositivos de entrada e saída) e software (programas e sistemas operacionais).

Arquitetura de Computadores: Isso envolve o design e a organização dos componentes de hardware para realizar tarefas específicas, como a CPU (Unidade Central de Processamento) que executa instruções.

Representação de Dados: Os dados são representados digitalmente em forma de bits (0s e 1s) e podem representar texto, números, imagens e outros tipos de informações. Sistemas Operacionais: Os sistemas operacionais são software que gerenciam recursos de hardware, permitem a execução de programas e fornecem uma interface para

interação com o computador.

Redes de Computadores: Sistemas computacionais podem ser interconectados para compartilhar recursos e informações, formando redes que variam em tamanho e complexidade.

Segurança da Informação: A proteção dos sistemas computacionais contra ameaças e acessos não autorizados é fundamental para garantir a integridade e a confidencialidade dos dados.

Algoritmos e Programação: Algoritmos são sequências de instruções que resolvem problemas, e a programação envolve escrever código para implementar esses algoritmos em linguagens de programação.

Armazenamento de Dados: Sistemas computacionais utilizam dispositivos de armazenamento, como discos rígidos e unidades de estado sólido, para guardar informações de forma persistente.

Computação em Nuvem: A computação em nuvem permite acessar recursos computacionais pela Internet, proporcionando escalabilidade e flexibilidade.

Evolução Tecnológica: Os fundamentos de sistemas computacionais continuam evoluindo com o tempo, impulsionados por avanços tecnológicos em

hardware e software. Esses são conceitos fundamentais que ajudam a compreender como os sistemas computacionais funcionam e como eles desempenham um papel crucial em nossa sociedade moderna.

# 

### 2.1 Desenvolvimento histórico

“O ábaco é um antigo instrumento de cálculo, formado por uma moldura com bastões ou arames paralelos, dispostos no sentido vertical, correspondentes cada um a uma posição digital (unidades, dezenas, centena,. . . ) e nos quais estão os elementos de contagem (fichas, bolas, contas,. . . ) que se podem fazer deslizar livremente“.(igeo, 2023)

Tendo conhecimento disso, desde os proêmios, o ser humano sempre explorou formas de se adaptar ao ambiente em que vive, sendo assim, criando ferramentas como uma lança e, hoje em dia, o celular que temos em mãos. O computador é uma das invenções mais importantes e falar sobre o primeiro pode ser muito relativo, porque existem diferentes exemplos dele.

Nós temos o ábaco (considerado até hoje como o inaugural entre todos inventados), ferramenta usada para cálculos matemáticos que encontra-se há mais de 5000 anos, tendo vestígios de ser inventado na região da Mesopotâmia e em sua estrutura básica consiste hastes dispostas horizontalmente, onde são movidas pedras em formato de discos para representar números.

“A Pascalina permite efectuar as operações de adição e subtração. Embora a operação seja demorada podem efectuar-se multiplicações e divisões pelo método das adições sucessivas e subtrações sucessivas..

A introdução da Pascalina no mercado não foi um sucesso comercial porque era excessivamente cara. Foram construídas apenas cerca de 50 Pascalianas, estando algumas delas expostas no ***Conservatoir des Arts et Métiers*** em Paris e outras no ***Science Museum***, em Londres”.(webpages.ciências.ulisboa.pt, 2023).

Diferente do ábaco, surgido em 1642, a Máquina de Pascal (Pascalina) foi criada e se tornou a primeira calculadora mecânica e realizava contas de subtração e adição por meio de suas engrenagens. Apesar da máquina realizar esses cálculos e ainda sim conseguir resolver contas de multiplicação e divisão - com o método de adição e subtração repetido sucessivamente, não foi um marco de vendas pelos custos para maquinário e foram desenvolvidas poucas réplicas.

“Charles Babbage é considerado o “pai” dos computadores. Seu primeiro conceito, a Diferença do Motor, é indiscutivelmente o primeiro computador já produzido. Desenvolvido em 1822, foi capaz de computar vários conjuntos de números e fazer cópias impressas dos resultados ``.(norgren, 2023).

Temos em seguida a Máquina Analítica de Babbage que foi inventada no ano de 1837. Charles Babbage a criou com intuito de funcionar com base nas instruções de cartões perfurados e era movida a vapor. Sua invenção foi intitulada como o primeiro computador por ter capacidade de ser programável através de seus cartões e antecipando conceitos utilizados nos tempos atuais e é grande influenciadora da tecnologia moderna.

“Seu inventor, o matemático Alan Mathison Turing, mostrou que a computação das operações de leitura, escrita e exclusão de símbolos binários poderiam ser satisfeitas por uma máquina que continha uma fita de comprimento ilimitado, com quadrados de tamanho definido sobre ela e um dispositivo com um número finito de estados, que realizava as operações na fita”. (ufrgs, 2023).

“Como mencionado, os poloneses foram os responsáveis por construir as primeiras bombas eletromecânicas. Elas obtiveram algum êxito em decifrar as mensagens criptografadas pela Enigma, no entanto, esse sucesso diminuiu drasticamente conforme a máquina alemã era aprimorada pelos engenheiros nazistas. Uma dessas implementações tornou-a indecifrável. No entanto, a captura de algumas chaves de criptografia possibilitou a quebra dos códigos nazistas. Foi nesse cenário que a programação feita por Alan Turing e sua equipe mostrou a sua utilidade“. (Brasilescola, 2023)

Em seguida, por meados dos anos 1936, o matemático Alan Mathison Turing foi quem concebeu a ideia da criação de um computador chamado Máquina Universal(logo em seguida nomeada como Máquina de Turing) que funcionou de forma que consistia em uma fita infinita dividida em células, uma cabeça de leitura e, ou, gravação em um conjunto de estados e instruções. Ela pode ler, escrever e mover-se na fita com base nas instruções, processando símbolos. Esse computador ajudou os Aliados (grupo do Reino Unido, França, União Soviética e Estados Unidos) a quebrar códigos utilizados por alemães na Segunda Guerra para se cominicar com o exército nazista. Sua invenção revolucionou a ciência da computação, estabelecendo as bases para o desenvolvimento de algoritmos e sistemas de programação e o nomeando pai da computação. “Qualquer processo aceito por nós homens como um algoritmo é precisamente o que uma máquina de Turing pode fazer” (Alonzo Church, Matemático).

“Interessado em melhorar sua inteligência de guerra, o Exército dos Estados Unidos decidiu patrocinar o projeto e investiu, na época, cerca de US$500 mil na invenção. A ideia era que o ENIAC pudesse ajudar nos cálculos balísticos necessários para a fabricação de munições e bombas durante a Segunda Guerra Mundial, mas a invenção só ficou pronta depois de terminado o conflito. Sua primeira apresentação pública aconteceu em 1946 e, no ano seguinte, o ENIAC foi ligado pela primeira vez. Em 1955, a máquina foi desativada, depois de ter sido usada para diversos projetos de pesquisa e desenvolvimento”. (techtudo, 2023)

Na primeira geração de computadores temos o ENIAC (Eletronic Numerical Integrator and Computer), criado em 1945 e de forma descomunal, tal qual ocupava 2 metros de altura, pesava cerca de 30 toneladas e ocupava 180 metros quadrados - como também consumia cerca de 200.000 W. Ele foi usado para cálculos complexos, impulsionando pesquisas científicas e militares. Apesar de dar prejuízos por seu porte agigantado e grande consumo de energia, o ENIAC tornou-se um marco na evolução dos computadores modernos ao substituir o processamento manual demorado por máquinas mais rápidas e versáteis.

“Em meados de 1947 e 1948, os estudos realizados por Willian Shockley, John Bardeen e Walter Brattain, levam ao aparecimento de um novo componente que revolucionou o mundo da eletrônica e da informática: Transistor. Desenvolvido em 1952 pela Bell Laboratories, o Transistor passou a ser um componente básico na construção de computadores, assinalando o início da Segunda Geração de Computadores”. (introdução-a-informática.webnode, 2023)

Em diante temos a segunda geração de computadores, essa qual se iniciou por volta da década de 1950 até o início da década de 1960. Nessa época ocorreu grandes avanços significativos em comparação da primeira geração, tendo em sua composição os transistores(principal inovação dessa geração) que substituíram as válvulas por serem menores, mais eficientes e com diminuição significante no consumo de energia.

Em razão disso, a substituição de válvulas por transistores permitiu que os computadores criados na época pudessem ser mais leves e menores. Graças a isto, as máquinas podiam ser alocadas em lugares menores sem ocupar tantos espaços, além de economizar nos custos de manutenção e contas.

Com a levá da segunda geração também surgiram as linguagens da programação, as quais pra época eram muito avançadas. As linguagens usadas na época eram o COBOL e FORTRAN que tornou a programação mais acessível e facilitou o desenvolvimento de programas mais complexos, sendo assim dando início a automação de tarefas comerciais e científicas.

“COBOL foi criado com o objetivo de oferecer maior controle aos negócios, especialmente para as instituições ligadas à área financeira. COBOL ainda é considerado uma das melhores linguagens para a área corporativa devido à sua elevada capacidade de processamento, lidando com grandes volumes de dados por causa de suas capacidades de processamento avançado de arquivos, e às normas específicas de segurança que foram desenvolvidas ao longo das décadas”. (blog.grancursosonline, 2023)

Sobre o COBOL(Common Business-Oriented Language) é necessário saber que é aplicada em sistemas empresariais e de processamento de dados. A sintaxe do COBOL é baseada em linguagem natural, tornando-a compreensível para programadores e analistas de negócios. Apesar de ser considerada uma linguagem antiga, é essencial em muitas empresas que dependem de sistemas legados, apesar do surgimento de linguagens de programação modernas.

Ainda sobre as linguagens, é também importante saber sobre o FORTRAN (Formula Translation) que é uma sintaxe de alto nível muito utilizada em computação científica e numérica desde sua criação. Apesar de estar em constante evolução ao longo dos anos, continua sendo utilizada por estudantes acadêmicos por sua confiabilidade e desempenho em aplicações científicas e de engenharia.

A segunda geração também incluiu a introdução de dispositivos de armazenamento magnético, tais como fitas e discos rígidos. Isso permitiu a capacidade de armazenar grande quantidade de informações, comparada com a geração passada, de forma bem mais rápida e eficiente. Esses avanços pavimentaram o caminho para o desenvolvimento futuro de computadores e contribuíram para a revolução da informática que continuaria nas décadas seguintes.

Vivenciamos, também, a terceira geração que surgiu em meados dos anos de 1960 a 1970. Durante essa época ocorreram evoluções significativas na tecnologia da computação que trouxeram avanços com máquinas mais poderosas e versáteis contendo:

-**Circuitos Integrados(CI)** que eram pequenos chips com diversos transistores, dando mais confiança e eficiência em termos de energia;

-**Transistores de junção bipolar(BJT)** utilizados para realizar funções lógicas e aumento na escala de sinais, tornando os computadores mais compactos e menos propenso a falhas;

No geral, a terceira geração de computadores foi marcada por avanços significativos em hardware, software e armazenamento de dados, que contribuíram para a evolução e a popularização da computação nas empresas, na pesquisa científica e em uma ampla gama de aplicações.

A quarta geração de computadores surgiu na década de 1970 e foi marcada pela revolução de microprocessadores. Essa geração deu início a popularização de computadores pessoais(PCs), com destaque maior para os lançamentos IBM PC que surgiram em 1981. Além disso, surgiram sistemas operacionais mais avançados, como o MS-DOS e o Apple DOS, que simplificaram a interação com os computadores.

A quarta geração representou um grande avanço na miniaturização e no desempenho, abrindo caminho para a computação pessoal e empresarial em larga escala, que se tornaria ainda mais acessível e difundida nas gerações subsequentes. Com esse avanço começaram a ser idealizados os computadores de mão, esses chamados como smartphones, iPod, iPad, tablets que são utilizados hoje em dia, fazendo parte da quinta geração, a que a sociedade está até o momento.

Atualmente estamos na quinta geração de computadores, surgindo no final do século XX, que se concentrou em desenvolver hardware especializado para tarefas de IA, como redes neurais e sistemas de aprendizado profundo. Esses sistemas tinham como objetivo aprimorar a capacidade de processamento de linguagem natural, reconhecimento de voz, tradução automática e resolução de problemas complexos.

### 

### 2.2 Conceitos básicos de arquitetura e organização de computadores

É possível notar como mudou a funcionalidade e a estrutura dos computadores ao passar do tempo. A arquitetura e a organização dos computadores são campos fundamentais da ciência da computação que é focada em entender como funciona um computador internamente, desde o nível mais baixo do hardware até o software que é usado.

Um computador comum compõe em sua estrutura base, organizada no seu hardware,processadores, memórias, entradas e saídas. Esses são os componentes básicos de um computador que são interligados e se encaixam para formar um sistema funcional.

A organização de computadores, por sua vez, lida com a maneira como esses componentes são projetados e conectados para garantir um desempenho eficiente e confiável. Isso envolve a alocação de recursos, o projeto de barramentos de dados, a otimização de ciclos de clock e a gestão da memória, entre outros aspectos.

Alguns conceitos principais na arquitetura e organização de computadores compõe são uma CPU (Unidade Central de Processamento)**,** memória que é classificada em RAM (memória de acesso aleatório) para armazenamento temporário e ROM (memória somente leitura) para armazenar informações essenciais do sistema.

O que também integra o sistema do PC são os barramentos, estes que servem como vias de comunicação que permitem a transferência de dados entre os componentes do computador. Os barramentos incluem barramentos de dados, barramentos de endereço e barramentos de controle.

Os dispositivos de E/S (Entrada e Saída) também consistem em permitir que o computador se comunique com o mundo externo, como teclado, mouse, monitor, impressora e dispositivos de armazenamento.

A Arquitetura Von Neumann é um modelo de arquitetura de computador que separa a memória de armazenamento de dados e instruções, permitindo a execução de programas armazenados.

O Sistema Operacional é um software que gerencia recursos de hardware, permite a execução de aplicativos e fornece uma interface para o usuário interagir com o computador.

O Pipeline de Instruções é uma técnica que divide o processamento da CPU em estágios sequenciais para melhorar o desempenho.

Ciclo de clock “conta” a taxa de pulsos de clock que controlam a operação da CPU e de outros componentes.

Hierarquia de memória faz uma organização dos diferentes tipos de memória, como cache, RAM e armazenamento secundário, para otimizar o acesso aos dados.

Constitui, também, RISC e CISC duas arquiteturas de conjuntos de instruções diferentes, com diferentes filosofias de projeto, para processadores.

A arquitetura e organização de computadores desempenham um papel central na evolução da tecnologia da informação e continuam a moldar a forma como interagimos com a tecnologia no mundo moderno.

# 

### 2.3 Estrutura básica de um computador

Por outro lado temos a estrutura básica de um computador, que consiste em várias partes e componentes que trabalham juntos para executar tarefas de processamento de dados. Aqui estão os principais elementos de uma estrutura básica de computador:

-Unidade Central de Processamento (CPU);

-Memória RAM (Random Access Memory);

-Disco Rígido (ou SSD);

-Placa-Mãe;

-Fonte de Alimentação;

-Placas de Expansão;

-Unidade Óptica (opcional);

-Periféricos;

-Sistema Operacional;

-BIOS/UEFI;

-Conexões e Portas;

-Gabinete;

-Cooling (Resfriamento;

-Rede;

Esses são os componentes essenciais de uma estrutura básica de computador. É imprescindível notar que a tecnologia está sempre evoluindo, e novos componentes e tecnologias podem ser introduzidos para melhorar o desempenho e a funcionalidade dos computadores.

### 

### 2.4 Hierarquia de níveis de computador

Ainda sobre a arquitetura e as estruturas básicas de um computador temos que salientar a hierarquia dele. As escalas de um computador referem-se a diferentes níveis de abstração e complexidade na organização de sistemas de computação. Os principais tipos de hierarquia resumidos são:

# 

### 2.4.1 Memória:

-Registradores: A memória mais rápida e próxima da CPU para armazenar dados temporariamente.

-Cache: Memória de alta velocidade usada para armazenar dados frequentemente acessados da RAM.

-RAM (Memória Principal): A memória principal do sistema usada para armazenar programas em execução e dados.

-Armazenamento Secundário: Dispositivos de armazenamento de longo prazo, como discos rígidos e SSDs.

### 

### 2.4.2 Processamento:

-Registradores: A memória mais rápida e próxima da CPU para armazenar dados temporariamente.

-Cache: Memória de alta velocidade usada para armazenar dados frequentemente acessados da RAM.

-RAM (Memória Principal): A memória principal do sistema usada para armazenar programas em execução e dados.

-Armazenamento Secundário: Dispositivos de armazenamento de longo prazo, como discos rígidos e SSDs.

### 2.4.3 Software:

-Sistema Operacional: Software que gerencia recursos de hardware e fornece uma interface para o usuário.

-Aplicativos: Software que executa tarefas específicas, como navegadores da web, processadores de texto, etc.

-Código de Máquina: A linguagem de baixo nível que as CPUs entendem diretamente.

### 2.4.4 Redes:

-Protocolos de Rede: Conjunto de regras para comunicação entre dispositivos em

-Dispositivos de Rede: Hardware, como roteadores e switches, que facilitam a comunicação em redes

-Camadas de Rede: Organização em camadas para processamento e transmissão de dados em redes.

### 

### 2.4.5 Armazenamento de dados:

-Bits e Bytes: A menor unidade de armazenamento de dados (0s e 1s).

-Arquivos: Conjuntos de dados organizados em unidades lógicas.

-Arquivos: Conjuntos de dados organizados em unidades lógicas.

### 

### 2.4.6 Linguagens de Programação:

-Linguagens de Alto Nível: Linguagens mais próximas da linguagem humana, como Python e Java.

-Linguagens de Baixo Nível: Linguagens mais próximas da linguagem de máquina, como Assembly.

Saber dessas hierarquias é essencial para entender como os computadores funcionam em seus diferentes níveis, desde o hardware até o software e comunicações. Cada nível tem sua importância e é necessário para manter a operação funcionando perfeitamente em seu desempenho, zelando o comprometimento do produto para com seu cliente.

# 

### 3. Componentes básicos de um computador

Os componentes básicos de um computador são as partes essenciais que trabalham juntas para executar tarefas de processamento de informações.

“A Unidade de Controle é um circuito que gera sinais elétricos para organizar como e quando as instruções devem ser executadas”. (Tecnoblog, 2023).

A CPU é o cérebro do computador e executa instruções. Ela inclui a Unidade de Controle, que gerencia operações, e que determina quando cada instrução será executada.

-Memória RAM: A memória de acesso aleatório (RAM) armazena temporariamente dados e programas que estão sendo usados ativamente pelo computador. É volátil, ou seja, perde os dados quando desligado.

-Disco Rígido (HD) ou Unidade de Estado Sólido (SSD): Esses dispositivos de armazenamento persistente guardam dados permanentemente, incluindo o sistema operacional, programas e arquivos.

-Placa-Mãe: A placa-mãe é a principal placa de circuito que conecta todos os componentes do computador, incluindo CPU, RAM, placas de expansão e periféricos.

-Fonte de Alimentação: A fonte de alimentação converte a energia elétrica da tomada em uma forma utilizável pelo computador, fornecendo energia para todos os componentes. Placas de Expansão: Estas placas podem ser adicionadas à placa-mãe para fornecer funcionalidades extras, como gráficos avançados (placa de vídeo), som (placa de som) ou conectividade de rede (placa de rede).

-Periféricos: Estes são dispositivos externos que interagem com o computador, como teclado, mouse, monitor, impressora, alto-falantes e câmera.

-Sistema Operacional: O sistema operacional é o software que gerencia o hardware e permite que os usuários executem programas e interajam com o computador.

-Barramentos: Os barramentos são vias de comunicação que permitem a transferência de dados entre os componentes internos do computador, como a CPU e a memória.

-Gabinete: O gabinete é a estrutura externa que abriga todos os componentes internos do computador, protegendo-os e fornecendo conectores para periféricos.

Esses são os componentes básicos que compõem um computador típico. Eles trabalham juntos para processar informações, armazenar dados e executar programas, tornando os computadores versáteis e poderosos para uma ampla gama de tarefas.

### 

### 3.1 Unidade central de processamento (cpu)

“CPU é a sigla para Central Process Unit, ou Unidade Central de Processamento. Ele é o principal item de hardware do computador, que também é conhecido como processador. A CPU é responsável por calcular e realizar tarefas determinadas pelo usuário e é considerado o cérebro do PC”. (Canaltech, 2023).

A CPU (Unidade Central de Processamento), frequentemente chamada de cérebro do computador, desempenha um papel vital na execução de tarefas e operações do sistema. Ela interpreta instruções de programas, realiza cálculos complexos, controla o fluxo de dados e gerência os recursos do sistema, tornando possível uma gama infinita de aplicações, desde proces- samento de texto e manipulação de imagens até a execução suave de jogos e aplicativos exigentes. As CPUs variam em termos de velocidade, número de núcleos, arquitetura e recursos, influenciando diretamente o desempenho geral da máquina.

### 

### 3.2 Memória principal

“As memórias RAM (Random-Access Memory - Memória de Acesso Aleatório) constituem uma das partes mais importantes dos computadores, pois são nelas que o processador armazena os dados com os quais está lidando. Esse tipo de memória tem um processo de gravação de dados extremamente rápido, se comparado aos vários tipos de memória ROM. No entanto, as informações gravadas se perdem quando não há mais energia elétrica, isto é, quando o computador é desligado, sendo, portanto, um tipo de memória volátil”. (Infowester, 2009).

A Memória Principal, ou RAM (Memória de Acesso Aleatório), desempenha um papel crucial no desempenho do sistema, oferecendo uma área de armazenamento temporário para dados e programas em uso ativo. Quando você inicia um programa, ele é carregado na RAM para permitir à CPU acesso instantâneo enquanto o programa está em execução. A RAM é conhecida por sua capacidade de acesso extremamente rápido aos dados, proporcionando eficiência operacional. No entanto, é importante destacar que a RAM é volátil, o que significa que os dados armazenados nela são temporários e se perdem quando o dispositivo é desligado ou reiniciado. Em contrapartida, a memória ROM é usada para armazenar dados permanentes, que não podem ser facilmente alterados ou apagados. Ela é chamada de “somente de leitura” devido à sua natureza estática e não volátil.

A capacidade de RAM é um fator essencial no desempenho do sistema, pois determina a quantidade de recursos disponíveis para executar tarefas com eficiência. Além disso, existem diferentes tipos de tecnologia de memória RAM, incluindo a rápida RAM Estática (SRAM) e a capacidade RAM Dinâmica (DRAM), juntamente com o mais recente MRAM, cada um com suas características distintas. A escolha do tipo e quantidade de RAM é um ponto crucial ao selecionar um dispositivo ou computador, uma vez que influenciará diretamente a capacidade de multitarefa e o desempenho geral.

### 

### 3.3 Memoria secundaria

“A Memória Secundária é um tipo de armazenamento de informação no compu- tador que guarda a informação mesmo depois de a energia ser removida. Não é volátil, ou seja, não necessita de uma fonte de energia constante para manter a informação armazenada no seu interior. A memória secundária é frequentemente utilizada para armazenar grandes quantidades de dados que podem ser acedidas rapidamente. Também é utilizada para armazenar informação durante mais tempo do que o computador está a funcionar”. (Techlib, 2023).

A Memória Secundária, representada pelo Disco Rígido (HD) e pelo Disco de Estado Sólido (SSD), assume um papel complementar ao oferecer armazenamento de longo prazo para dados que precisam de persistência, mesmo quando o computador é desligado. Essa categoria de armazenamento de dados é não volátil, o que significa que os dados nela armazenados permanecem intactos independentemente do estado do dispositivo. No entanto, a memória secundária tem tempos de acesso significativamente mais lentos em comparação com a RAM, tornando-a ideal para armazenar grandes volumes de informações que não precisam ser acessadas rapidamente. Além disso, sua capacidade pode ser facilmente expandida por meio de dispositivos de armazenamento externos, como discos rígidos externos e unidades flash USB, proporcionando flexibilidade adicional.

### 3.4 Dispositivos de entrada e saída

“Dispositivos de entrada e saída são os aparelhos que permitem a interação de um processador, geralmente um computador, com o homem. Também são conhecidos como periféricos e possibilitam a entrada e saída de dados”. (Significados, 2018).

Os Dispositivos de Entrada e Saída (E/S) desempenham um papel fundamental na interação entre o usuário e o computador, além de facilitar a comunicação com o ambiente externo. Os dispositivos de entrada permitem ao computador coletar dados do usuário e inseri-los no sistema. Por outro lado, os dispositivos de saída traduzem as informações do computador em uma forma compreensível para o usuário, desempenhando um papel crucial na apresentação e na comunicação de dados. Alguns dispositivos, como pen drives, modems e cartões de memória, podem operar tanto como dispositivos de entrada quanto de saída, dependendo do contexto e da função que estão desempenhando. Isso permite uma flexibilidade notável na transferência de informações entre o computador e outros dispositivos, como câmeras digitais e dispositivos móveis, ampliando ainda mais a utilidade dessa maravilha da tecnologia.

### 

### 3.4.1 Dispositivo de entrada

-Mouse;

-Scanner;

-Teclado;

-Webcam;

-Microfone, etc.

### 3.4.2 Dispositivo de saída:

-Projetor;

-Monitor;

### 3.4.3 Dispositivos híbridos:

-Drive;

-Modem;

-Cartão de Memória, etc

-Pen

Em resumo, o computador é uma maravilha da engenharia eletrônica que desempenha um papel indispensável em nossas vidas. Compreender os principais componentes, como CPU, RAM, memória secundária e dispositivos de entrada e saída, é essencial para aproveitar ao máximo essa tecnologia versátil e poderosa, que moldou a maneira como vivemos, trabalhamos e nos comunicamos no mundo moderno. Portanto, ao escolher um dispositivo ou computador, é fundamental considerar cuidadosamente esses elementos para atender às necessidades específicas de cada usuário.

### 4. Sistemas numéricos: conceitos, simbologia e representação de base numérica.

“O sistema binário é um sistema de numeração onde todas as quantidades são apresentadas com base em dois números: 0 e 1”. (Conhecimento Científico, 2023).

“Há registros de vários sistemas de numeração durante a história das civilizações. Com a necessidade de contabilizar, o ser humano desenvolveu a ideia de número e a sua representação em algarismos e sistemas de numeração”. (Mundo Educação Raul Rodrigues de Oliveira, 2023).

São sistemas matemáticos que utilizam símbolos ou dígitos para representar quanti- dades ou números. Eles são a base da representação de números em diferentes contextos, como matemática, ciência da computação, eletrônica e muitas outras áreas.

Os números e a forma como eram organizados nem sempre foram representados da mesma maneira que usamos hoje em dia. Cada civilização desenvolveu seu próprio método de representação numérica.

O sistema de numeração que usamos atualmente é chamado de sistema decimal po- sicional, que possui 10 dígitos e a posição de cada dígito desempenha um papel importante na representação numérica.

“O sistema binário funciona como um sistema de numeração posicional onde nú- meros só podem ser 0 ou 1, por exemplo: 1010011010.” (Conhecimento Cientifico, 2023).

No decorrer da história, existiram outros sistemas numéricos que utilizavam cinco dígitos, 60 dígitos e muitos outros, cada um com suas características particulares. Os símbolos “11”, “onze” e “XI” (que significa onze em latim) representam o mesmo número, mas são expressos em idiomas e períodos diferentes. Um sistema de numeração, também chamado de sistema numeral, é um sistema que representa um conjunto de números por meio de numerais de maneira consistente. Pode ser considerado como o contexto que possibilita que o numeral “11” seja interpretado como o numeral romano para dois, o numeral binário para três ou o numeral decimal para onze.

Uma base numérica denota a quantidade de dígitos ou símbolos disponíveis para a representação de um sistema numérico.

Alguns dos sistemas numéricos mais amplamente reconhecidos são:

| **Decimal** | **Binário** | **Hexadecimal** |
| --- | --- | --- |
| 0 | 0000 | 0 |
| 1 | 0001 | 1 |
| 2 | 0010 | 2 |
| 3 | 0011 | 3 |
| 4 | 0100 | 4 |
| 5 | 0101 | 5 |
| 6 | 0110 | 6 |
| 7 | 0111 | 7 |
| 8 | 1000 | 8 |
| 9 | 1001 | 9 |
| 10 | 1010 | A |
| 11 | 1011 | B |
| 12 | 1100 | C |
| 13 | 1101 | D |
| 14 | 1110 | E |
| 15 | 1111 | F |

Fonte:Grupo

Ao observar esta tabela, torna-se mais evidente que, ao realizar a contagem, quando alcançamos o último símbolo, é necessário incrementar o dígito

à esquerda para representar o próximo número. Por exemplo,

na base decimal, quando chegamos a 9, precisamos usar

o símbolo 1 para formar o número seguinte, que é 10. O mesmo princípio se aplica às outras bases numéricas. Por exemplo, no sistema octal, ao atingir o 7, o próximo número é 10; ao chegar ao 17, o próximo é 20, e assim por diante. No sistema binário, a contagem é feita da seguinte maneira: 0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010 e assim por diante.

### 4.1 Conversão entre bases numéricas: decimal

“O sistema decimal é um sistema de numeração de posição que utiliza a base dez.” (Universidade Federal de Santa Cararina, Haline de Souza Scotti, Rodrigo Fantinati Ferreira, 2023).

Números decimais são valores não inteiros que incorporam uma vírgula, denotando que o próximo dígito representa a posição das décimas. Esses números decimais podem exibir um número finito ou infinito de casas decimais, categorizando-se como números racionais ou irracionais.

Neste sistema, encontramos um conjunto de dez símbolos que abrangem de 0 a 9 (0, 1, 2, 3, . . . , 9).

Essa característica permite a representação dos valores em unidades individuais, po- rém, para representar valores maiores do que nove, é necessário combinar esses símbolos.

Realizamos a divisão do número da base decimal por 2 repetidamente até que não seja mais divisível. Ao final desse processo, o número binário é composto pelo resultado da última divisão, combinado com os restos das divisões anteriores, organizados “de baixo para cima”.

### 4.1.1 Conversão para Binário:

“Para transformar um número decimal em binário, é preciso fazer sucessivas divi- sões sobre a base, que é 2, até que não seja mais possível dividir.” (Conhecimento Científico, 2023).

A conversão para o sistema binário segue um processo de divisões sucessivas por

2.

O número binário é então formado a partir do quociente da última divisão, seguido

dos restos de todas as divisões anteriores, mantendo a ordem em que foram realizadas.

Vejamos um exemplo prático de como converter o número decimal 8D para binário:

-Dividimos 8 por 2, obtendo um quociente de 4 e um resto de 0 (8/2 = 4, resto = 0).

-Em seguida, dividimos 4 por 2, obtendo um novo quociente de 2 e novamente um resto de 0 (4/2 = 2, resto = 0).

-Continuamos dividindo 2 por 2, chegando a um quociente de 1 e um resto de 0 (2/2= 1, resto = 0).

Agora, o número binário é formado pegando-se o último quociente (1) e anexando os restos das divisões anteriores na ordem em que foram realizadas.

Portanto, temos:

8D = 1000B

### 

### 4.1.2 Conversão para octal:

Vamos converter o número decimal 45 para octal. Dividimos 45 por 8:

45 ÷ 8 = 5 com resto 5

* Anotamos o resto (5) como o dígito menos significativo do número octal. Dividimos o quociente anterior (5) por 8:

5 ÷ 8 = 0 com resto 5

- Anotamos o novo resto (5) como o próximo dígito do número octal.

- Agora, reunimos os dígitos do número octal, lendo-os da última etapa para a primeira: O número octal correspondente a 45 em decimal é 55 em octal.

Para realizar a conversão de binário para decimal, seguimos um procedimento inverso. Cada dígito binário, também conhecido como “bit,” é multiplicado pela base do sistema (base=2), elevada à posição que ocupa. A soma das multiplicações de cada bit pelo valor das potências resulta no número decimal representado.

Vamos considerar o exemplo de conversão de binário para decimal: 1011B = ?D Podemos calcular o valor decimal desse número binário da seguinte maneira:

1 × 2³ + 0 × 2² + 1 × 2¹ + 1 × 2 = 8 + 0 + 2 + 1 = 11

Portanto, 1011B é igual a 11D em decimal

### 4.2 Conversão entre bases numéricas: binárias

“O sistema binário ou base 2, é um sistema de numeração posicional em quetodas as quantidades se representam com base em dois números.” (Universidade Federal de Santa Cararina, Haline de Souza Scotti, Rodrigo Fantinati Ferreira).

O sistema binário, também conhecido como sistema de base 2, é um sistema de numeração posicional em que todas as quantidades são representadas com base em apenas dois dígitos: zero e um. Esse sistema é intrínseco aos computadores digitais, pois eles operam internamente com dois níveis de tensão, tornando o sistema binário a escolha natural para sua representação numérica.

Quanto à soma de números binários, o procedimento é o seguinte:

O procedimento para somar números binários é semelhante à soma de números decimais, mas com algumas regras específicas devido à base binária. Aqui está o procedimento passo a passo:

Comece da direita para a esquerda, alinhando os dígitos menos significativos (da mesma posição) um abaixo do outro.

Some os dígitos na coluna mais à direita. Aqui estão as regras de soma para cada par de dígitos:

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 10

(Observe que quando a soma é 2 em binário, você escreve 0 e carrega o 1 para a próxima coluna à esquerda, se houver.)

Continue somando os dígitos nas colunas subsequentes, levando em consideração quaisquer carregamentos (1s) da etapa anterior. Lembre-se de que 1 + 1 + 1 resulta em 1 na coluna atual e carrega 1 para a próxima.

Continue esse processo até ter somado todos os dígitos, indo da direita para a esquerda.

Se após a última soma ainda houver um carregamento (1) para a coluna mais à esquerda, insira esse 1 no resultado final.

Exemplos

| Interface gráfica do usuário  Descrição gerada automaticamente com confiança médiaTexto  Descrição gerada automaticamenteImagem de vídeo game  Descrição gerada automaticamente com confiança média |
| --- |

Fonte: Grupo

Quando subtraímos 1 de 0 em um contexto binário, é necessário fazer um “empréstimo” do elemento vizinho. Esse empréstimo tem o valor de 2, devido à base binária. Portanto, na coluna em que temos 0 - 1, o resultado é 1, porque na realidade a operação realizada é 2 - 1 = 1. Esse processo se repete e o elemento que cedeu o “empréstimo” e originalmente tinha o valor de 1 agora assume o valor de 0. Os asteriscos são usados para indicar os elementos que “emprestaram” para seus vizinhos. É importante observar que, quando o valor de um elemento é zero, ele não pode “emprestar” para ninguém, então o “pedido de empréstimo” passa para o próximo elemento e esse zero é transformado em 1.

### 

### 4.3 Conversão entre bases numéricas: octal

“Antigamente o Sistema octal era a principal alternativa mais compacta ao binário na programação em linguagem de máquina. Atualmente, o sistema hexadecimal é mais utilizado como alternativa ao binário.” (GrowthCode, Walmir Silva, 2020).

O Sistema Octal é um sistema de numeração com base 8, o que significa que ele usa oito símbolos para representar quantidades. No Ocidente, esses símbolos correspondem aos algarismos arábicos. O sistema octal foi amplamente empregado no campo da informática como uma alternativa mais concisa ao sistema binário para a programação em linguagem de máquina.

Existem diversos métodos para realizar essa conversão, sendo o mais comumente utilizado aquele derivado do Sistema de Numeração Posicional. Neste método, a conversão ocorre de maneira direta, aplicando-se uma fórmula específica.

Vejamos alguns exemplos:

Para converter o número octal 764 para o sistema decimal, aplicamos a seguinte fórmula:

7648 = 7 x 8² + 6 x 8¹ + 4 x 8 = 448 + 48 + 4 = 50010

Dessa forma, aplicando a fórmula corretamente, podemos realizar a conversão de números octais para o sistema decimal de maneira precisa.

### 

### 4.3.1 Conversão de octal para binário

Quando há a necessidade de converter números octais em binários, o processo consiste em separar cada dígito do número octal e substituí-lo por seu equivalente em binário.

Exemplo:

Vamos converter o número octal 1572 em binário. Dessa forma, 1 5 7 2 = 001 101 111 0

### 

### 5 Álgebra booleana: a base de lógica digital

“As portas lógicas são blocos fundamentais da Álgebra Booleana, que é a base matemática da eletrônica digital.” (Blog da Eletrônica, Henrique).

Álgebra Booleana e Lógica Digital são fundamentais na eletrônica digital e na computação. Aqui está um resumo breve sobre esses conceitos:

Álgebra Booleana:

A Álgebra Booleana é uma área da matemática que lida com valores lógicos e operações lógicas.

Utiliza variáveis booleanas (que têm apenas dois valores possíveis: 0 ou 1, verdadeiro ou falso) e operadores lógicos, como AND, OR e NOT.

### 

### 5.1 Principais operações lógicas

-AND: Retorna verdadeiro (1) apenas se todas as entradas forem verdadeiras.

-OR: Retorna verdadeiro (1) se pelo menos uma das entradas for verdadeira.

-NOT: Inverte o valor da entrada (0 se for 1, e vice-versa).

A Álgebra Booleana é usada para simplificar e analisar circuitos lógicos, expressar funções lógicas e projetar sistemas digitais.

### 

### 5.2 Lógica digital

“As portas lógicas são usadas para implementações físicas da álgebra de Boole. As portas lógicas formam a base do hardware a partir do qual todos os circuitos digitais são construídos.” (DPI, INPE, 2022).

A Lógica Digital é a aplicação da Álgebra Booleana na eletrônica digital e na representação de informações por meio de sinais digitais (0s e 1s).

Ela é fundamental na concepção e construção de sistemas digitais, como computadores, smartphones, microcontroladores e muito mais.

Na lógica digital, as variáveis booleanas representam o estado de componentes eletrônicos, como transistores, que podem estar ligados (1) ou desligados (0).

Os circuitos digitais são construídos com portas lógicas (como portas AND, OR e NOT) que realizam operações booleanas e realizam funções específicas.

A Lógica Digital é a base da aritmética binária, da representação de números em computadores e da execução de operações matemáticas complexas por meio de circuitos eletrônicos.

Em resumo, a Álgebra Booleana fornece o framework teórico para a Lógica Digital, que por sua vez é a base da eletrônica digital e da computação moderna, permitindo o processamento e a manipulação de informações de forma eficiente e precisa.

### 

### 5.3 Expressões lógicas

“Pode ser definida com um conjunto de operadores e um conjunto de axiomas, que são assumidos verdadeiros sem necessidade de prova.” (Inf, UFSC, 2021).

As expressões lógicas são a base da Álgebra Booleana. Elas são construídas usando variáveis booleanas (geralmente representadas por letras, como A, B, C, etc.), constantes (0 para falso e 1 para verdadeiro) e operadores lógicos. Os operadores lógicos básicos incluem:

-AND (E): Representado por \*, && ou ·, este operador produz verdadeiro apenas

quando todos os operandos são verdadeiros. Por exemplo, A \* B é verdadeiro apenas quando A e B são verdadeiros.

-OR (OU): Representado por +, || ou +, o operador OR produz verdadeiro se pelo menos um dos operandos for verdadeiro. Por exemplo, A + B é verdadeiro se A ou B (ou ambos) forem verdadeiros.

-NOT (NÃO): Representado por , ou ’, o operador NOT inverte o valor lógico de uma variável. Se A for verdadeiro ∼A será falso, e vice-versa.

### 

### 5.4 Portas lógicas

“São usadas para implementações físicas da álgebra de Boole. Formam a base do hardware a partir do qual todos os circuitos digitais são construídos.” (DPI, INPE, 2022).

Portas lógicas são dispositivos eletrônicos que implementam operações lógicas. Elas recebem entradas binárias (0 ou 1) e produzem uma saída com base nas regras da Álgebra Booleana. As portas lógicas mais comuns incluem:

-AND Gate (Porta E): Produz 1 se todas as entradas forem 1. A saída é 0 se pelo menos uma entrada for 0.

-OR Gate (Porta OU): Produz 1 se pelo menos uma das entradas for 1. A saída é 0 nas quando todas as entradas são 0.

-NOT Gate (Porta NÃO): Inverte o valor da entrada. Se a entrada for 1, a saída é 0, e vice-versa.

### 

### 5.5 Introdução a circuitos

“Um circuito lógico se faz importante quando, por exemplo, é necessário controlar uma ou mais saídas através da sequência de acionamento das entradas.” (Maker Hero, 2019).

Circuitos lógicos são sistemas compostos por portas lógicas interconectadas. Eles são a base da eletrônica digital e da computação. Alguns conceitos-chave sobre circuitos lógicos incluem:

-Composição de Circuitos: Circuitos lógicos podem ser construídos combinando diferentes portas lógicas. Isso permite a criação de circuitos complexos que executam diversas tarefas.

-Fluxo de Sinal: No interior de um circuito, o sinal (0 ou 1) flui de uma porta lógica para outra seguindo as regras definidas pelas operações lógicas. Isso permite a execução de cálculos e processamento de informações.

-Aplicações: Circuitos lógicos são utilizados em uma ampla gama de aplicações, desde processadores de computadores até sistemas de controle em automóveis e dispositi- vos eletrônicos de consumo.

### 

### 6 Introdução a linguagem python

“Python é uma linguagem de programação de alto nível — ou High Level Language —, dinâmica, interpretada, modular, multiplataforma e orientada a objetos — uma forma específica de organizar softwares onde, a grosso modo, os procedimentos estão submetidos às classes, o que possibilita maior controle e estabilidade de códigos para projetos de grandes proporções.”(O que é Python, Kenzie, Daniel Kriger, 2022)

É notório a importância do Python e isso se dá por ser uma linguagem de programação versátil e legível, amplamente usada em desenvolvimento web, ciência de dados, automação e IA(Inteligência Artificial). Sua sintaxe clara e vasta biblioteca padrão a torna popular entre iniciantes e profissionais. Python é interpretada, facilitando o desenvolvimento, o desenvolvimento e a portabilidade em sistemas operacionais diversos. É uma escolha poderosa para criar uma variedade de soluções de software.

O que torna Python ainda mais impressionante é sua sintaxe simples e de fácil compreensão, o que a torna acessível mesmo para profissionais que não são programadores de formação, como engenheiros, matemáticos, cientistas de dados e pesquisadores. Essa abordagem amigável ao usuário ajudou a popularizar Python em uma ampla gama de indústrias e campos.

“Um de seus maiores atrativos é possuir um grande número de bibliotecas, nativas e de terceiros, tornando-a muito difundida e útil em uma grande variedade de setores dentro de desenvolvimento web, e também em áreas como análise de dados, machine learning e IA.”(O que é Python, Kenzie, Daniel Kriger, 2022)

As principais características do Python que chamam mais atenção do público são: ser uma linguagem de alto nível, ampla biblioteca padrão, multiplataforma, interpretada(que elimina a necessidade de compilação, permitindo desenvolvimento rápido e flexível), orientada a onjetos, comunidade ativa com desenvolvedores que constantemente contribuem com biblioteca e recursos, versatilidade, gratuito, fácil integração, dinamicamente tipada e geração rápida de protótipos.

“Para que esta melhoria pudesse ser feita de forma mais rápida e eficaz, Guido desenvolveu uma linguagem muito descomplicada e flexível: o Python.

Uma vez que esta linguagem passou a possibilitar a criação desde scripts muito simples até sistemas extremamente poderosos, profissionais de várias áreas começaram a progressivamente utilizá-la cada vez mais.Hoje, além dos desenvolvedores de software, temos biólogos, contadores, físicos e outros profissionais potencializando suas habilidades através dela.

Estes são alguns motivos que têm feito o uso do Python crescer consideravelmente nos últimos anos em detrimento de outras linguagens.”(O que é Python, Kenzie, Daniel Kriger, 2022)

Guido Van Rossum, por volta da década de 90, criou a linguagem Python que se tornou um sucesso ao longo dos anos que, como foi mencionado antes, atraiu muitos olhares dos profissionais da área e pessoas interessadas no ramo. Ela se tornou uma linguagem amplamente adotada em uma variedade de setores, incluindo desenvolvimento web, ciência de dados, inteligência artificial e automação. A comunidade Python também desempenhou um papel crucial em seu crescimento, contribuindo com uma ampla gama de bibliotecas e frameworks que expandiram ainda mais suas capacidades.

Hoje, o Python é uma das linguagens de programação mais populares e amplamente usadas em todo o mundo. Sua história, que começou como um projeto pessoal de um engenheiro holandês, demonstra como uma ideia criativa e um compromisso com a simplicidade podem resultar em uma linguagem poderosa e influente que continua a prosperar e a evoluir.

### 

### 6.1 O que é uma estrutura de repetição

“Uma estrutura de repetição em Python é um recurso para desenvolver tarefas repetitivas em um loop contínuo. O loop funciona até uma condição ser satisfeita.

É importante configurar devidamente essa condição para não cair em erros fatais em um programa.

O loop faz o que chamamos de programação de iteração, uma repetição que analisa alguma estrutura.

Pense, por exemplo, em um parágrafo com 40 palavras. Em um dado projeto, é necessário percorrer cada letra para fazer uma verificação. Se você não puder contar com uma estrutura de repetição, terá que fazer isso do jeito mais difícil e braçal.

A forma mais lógica é percorrer cada item de forma separada, depois de transformar o parágrafo em uma lista. Assim, você faz um IF para cada item e verifica todos até terminar a lista.

Com essa operação, você perde muito tempo de codificação e muitas linhas de código em um processo difícil de ler e de manter.

Com uma estrutura de repetição no Python, por outro lado, tudo se torna mais simples e intuitivo.”

Em Python, as estruturas de repetição desempenham um papel fundamental na gestão eficaz de tarefas que envolvem repetição contínua de ações até que uma condição específica seja atendida.

A configuração precisa dessa condição é de extrema importância para prevenir potenciais erros significativos em programas. Considere, por exemplo, a necessidade de processar uma planilha extensa com milhares de linhas e realizar operações detalhadas em cada célula. Sem o uso de estruturas de repetição, essa tarefa se tornaria demorada e exaustiva.

A abordagem convencional envolveria a conversão do parágrafo em uma lista e a aplicação de verificações individuais para cada elemento. Esse processo resultaria em um código extenso e de difícil compreensão, tornando a manutenção um desafio.

O uso de estruturas de repetição em Python simplifica consideravelmente essa tarefa. Basta estabelecer uma condição que oriente a iteração pela lista até que todos os elementos sejam processados e, em seguida, aplicar um único bloco condicional para analisar cada caractere do parágrafo.

O que anteriormente demandaria a criação de centenas de linhas de código se resume a apenas algumas. Isso não apenas torna o código mais claro, mas também facilita modificações futuras, simplificando todo o processo e tornando-o mais intuitivo. As estruturas de repetição em Python são aliadas poderosas na simplificação de tarefas repetitivas e na melhoria da clareza do código.

“Uma estrutura de repetição em Python funciona da mesma maneira que uma estrutura em outras linguagens, como C e Java. A diferença básica é a sintaxe.

Agora que já falamos sobre as definições, vamos de fato entender como são a sintaxe e a forma dessas estruturas.

Para exemplo, vamos usar os loops For e While, que são os principais.

A diferença entre eles está na organização sintática e na maneira como se relacionam com a condição de término. É preciso saber exatamente como usar cada um e em quais cenários eles se aplicam.”(Estruturas de Repetição, Hashtag Treinamentos, Heitor Catunda, 2022)

Tendo essas informações como base, é possivel notar que o Python se diferencia de outras linguagens por sua sintaxe simple e legibilidade. Suas estruturas de repetição incluem ‘for’(estrutura de repetição que executa um bloco de código várias vezes com base em uma condição inicial e final) e ‘while’(executa um bloco de código enquanto uma condição específica for verdadeira).

Usarei a seguir exemplos práticos de cada estrutura de repetição, sendo elas for e while:

**⦁ FOR:**

|  |
| --- |

Fonte: Grupo.

Nesse exemplo usamos o for para calcular a soma dos primeiros 10 números naturais. Usamos a variável ‘soma’ como zero. O for para interagir sobre os números de 1 a 10(inclusive), e em cada interação, e em cada interação adicionamos o valor de ‘i’ à variável ‘soma’. Ao final do loop, exibimos a soma total, que será 55, pois é a soma dos números de 1 a 10.

⦁ **WHILE:**

|  |
| --- |

Fonte: Grupo.

Pode-se notar que, neste exemplo, usamos um loop ‘while True’ para criar um loop infinito. Dentro do loop,

pedimos ao usuário para digitar um número. Se o número digitado for igual a zero, o loop é interrompido com a

instrução ‘break’. Caso contrário, o número é somado à variável ‘soma’. O programa continuará solicitando números

até que o usuário insira 0, o momento em que a soma dos números digitados será exibida.

### 

### 6.2 Estruturas condicionais e lógicas

“Uma estrutura condicional na linguagem Python, como a Python If Else, corresponde a um bloco de código que é iniciado com uma expressão para avaliar se uma determinada condição é verdadeira ou falsa. Com ele, podemos testar se uma variável é igual a zero, por exemplo. O resultado dessa verificação é determinante para a execução das instruções seguintes presentes no escopo da estrutura.

Na prática, será executado uma ou mais instruções se a condição for verdadeira, ou outro bloco de código caso o resultado seja falso. As pessoas programadoras precisam entender como essa estrutura funciona, pois ela é muito utilizada no desenvolvimento de aplicações.”(Estruturas Condicionais, Trybe, Michelle Horn, 2021)

As estruturas lógicas em Python são fundamentais para controlar o fluxo de um programa com base em

condições. Por sua vez, estruturas condicionais são compornentes-chaves da programação, permitindo decisões com

base em condições. Usam ‘if, ‘else’ e ‘elif’(que explicaremos a seguir com exemplos) para executar diferentes ações

com base em valores booleanos, direcionando o fluxo do programa e tornando-o adaptável a difetentes situações. Essas

estruturas são essenciais para se criar programas que respondam dinamicamente às necessidades do usuário ou às

condições variáveis durante a execução.

As principais portas lógicas incluem:

**⦁ IF:** usado para executar um bloco de código se uma condição for verdadeira.

|  |
| --- |

Fonte: Grupo.

Neste exemplo, o código está verificando se a variável ‘idade’ é maior ou igual a 18 e, se for verdadeira, ele imprime a mensagem “Você é maior de idade”.

⦁ **ELSE:** complementando o ‘if’, é usado para executar um bloco de código quando a

condição não é verdadeira.

|  |
| --- |

Fonte: Grupo.

Como é possível notar no exemplo, o ‘else’ foi utilizado para informar se o usuário é menor de idade caso a variável ‘idade’ for menor que 18, que no caso, foi utilizado o número 17 para ser exibido o resultado “Você é menor de idade.”

**⦁ ELIF:** é usado após um ‘if’ para vereficar as condições adicionais.

|  |
| --- |

Fonte: Grupo

Aqui é possível notar que estamos verificando a idade da pessoa e usando ‘elif’ para testar diferentes condições em ordem. Se a idade for menor que 18, ele imprimirá “Você é menor de idade.” Se a idade for igual a 18,

Imprimirá “Você completou 18 anos agora.” Caso contrário, imprimirá “Você é maior de idade.”

Além das estruturas lógicas básicas mencionadas anteriormente, Python oferece algumas estruturas

adicionais para o controle de fluxo. Darei alguns exemplos e explicação dos mesmos.

Operadores lógicos são usados para combinar ou negar condições, os principais em Python são ‘and’, ‘or’ e

‘not’.

**⦁ AND**: é um operador lógico que retorna ‘True’ apenas se todas as condições forem verdadeiras, caso contrário, retorna ‘False’.

|  |
| --- |

Fonte: Grupo.

Neste exemplo o ‘and’ é usado para combinar as duas condições, a primeira verifica se está chovendo, representada pela variável ‘chuva’. A segunda condição verifica se você possui um guarda-chuva, representado pela variável ‘guarda chuva’.

**⦁ OR:** é um operador que retorna ‘True’ se pelo menos uma das condições for verdadeira,

caso contrário, retorna ‘False’.

|  |
| --- |

Fonte: Grupo.

Agora é possível notar que estamos usando o operador ‘or’ para combinar duas condições. A primeira condição verifica se você está com sono, representada pela variável ‘tem sono’. A segunda condição verifica se você está com fome, representada pela variável ‘tem fome’.

Se pelo menos uma das condições for verdadeira(ou seja, você está com sono ou com fome, ou ambos), o programa imprimirá “Você deveria descansar ou comer, ou ambos!” Caso contrário, se ambas condições forem falsas,

Imprimirá “Você está bem descansado e não tem fome.”

**⦁ NOT:** usado para inverter o valor de uma expressão. Transforma ‘True’ em ‘False’ e ‘False’ em ’True’, alterando o resultado lógico.

|  |
| --- |

Fonte: Grupo.

Neste exemplo, estamos usando o operador ‘not’ para negar a condição da variável ‘tem energia’. A condição original é ‘tem energia’, que é falsa. Ao aplicar o operador ‘not’, invertemos essa condição, tornando-a verdadeira. Portanto, o programa imprimirá “Você está cansado e precisa de descanso.” porque a negação da condição é verdadeira. Se a variável ‘tem energia’ fosse ‘True’, o programa imprimiria “Você está cheio de energia!” porque a negação da condição seria falsa.

“Em Python, uma função é uma sequência de comandos que executa alguma tarefa e que tem um nome. A sua principal finalidade é nos ajudar a organizar programas em pedaços que correspondam a como imaginamos uma solução do problema. Você pode inventar qualquer nome para as funções que você cria, exceto que você não pode usar um nome que é uma palavra reservada em Python, e que os nomes devem seguir a regra de identificadores permitidos. Os parâmetros especificam qual informação, se alguma, você deve providenciar para que a função possa ser usada. Outra forma de dizer isto é que os parâmetros especificam o que a função necessita para executar a sua tarefa.(Funções em Python, Pense Python, 2023)”

Conclui-se que, as estruturas lógicas, permitem controlar o fluxo do programa com base em condições. Essas estruturas avaliam expressões lógicas e executam blocos de código correspondentes quando as condições são verdadeiras. Também podemos pontuar que as funções em Python são blocos de códigos reutilizáveis que realizam tarefas específicas. Elas recebem argumentos, processam dados e podem retornar resultados. As funções ajudam a dividir em menores partes do código, tornando-o mais organizado e legível. Podemos definir funções com a palavra-chave ‘def’ e chamá-las em qualquer lugar do programa. Python também possui funções embutidas, como ‘print()’ e ‘len()’, e você pode criar suas próprias funções personalizadas para resolver problemas específicos, melhorando a manutenção e a reutilização do código.

### 

### 7.0 Estrutura de dados em python

“Quando falamos de uma estrutura de dados, estamos falando nas operações que essa estrutura nos permite realizar, bem como da complexidade assintótica dessas operações.” (Algoritmosempython, 2017)

A seleção da estrutura apropriada depende das ações que seu programa realizará com maior frequência. De maneira geral, o objetivo é optar por uma estrutura de dados que reduza ao máximo o custo das operações mais comuns.

Estruturas de dados podem ser classificadas em dois grupos:

### 7.0.1 Estruturas contíguas:

-Os elementos são armazenados em posições de memória adjacentes, ou seja, um após o outro.

-Acesso direto aos elementos por meio de índices.

-Tamanho fixo, uma vez alocado, não pode ser facilmente modificado.

-Eficiente para acessar elementos por índice, mas ineficiente para inserções/remoções no meio da estrutura, pois pode exigir deslocamento de elementos.

### 

### 7.0.2 Estruturas ligadas

-Os elementos são armazenados em blocos de memória chamados nós, cada nó contém dados e uma referência ao próximo nó.

-Não requer espaço contíguo na memória.

-Tamanho dinâmico, fácil de adicionar ou remover elementos.

-Acesso aos elementos requer percorrimento da estrutura, a menos que seja uma lista duplamente ligada com referências para trás.

-Eficiente para inserções/remoções, mas menos eficiente para acesso aleatório.

Em resumo, estruturas contíguas são adequadas quando o tamanho é fixo e o acesso aleatório é importante, enquanto estruturas ligadas são mais flexíveis e adequadas para inserções/remoções dinâmicas, mas têm um custo maior para o acesso aleatório de elementos. Cada uma tem suas vantagens e desvantagens, e a escolha depende dos requisitos específicos de um problema.

### 7.1 Algoritmos de busca

“Um algoritmo de busca em vetor é um algoritmo para procurar a presença ou não de determinado valor em uma sequência de dados em memória (em um vetor). Deve-se destacar que a realização de busca é provavelmente o algoritmo mais empregado na prática, por exemplo, sempre que acessa um sistema com usuário e senha será necessário buscar pelo seu usuário e depois verificar se a senha está correta.” (Edisciplinas, 2023)

Algoritmos de busca em Python são procedimentos ou métodos que permitem encontrar a localização ou verificar a presença de um valor específico em uma coleção de dados, como uma lista, tupla ou dicionário, usando a linguagem de programação Python. Esses algoritmos são essenciais para a manipulação eficiente de dados e desempenham um papel fundamental em diversas aplicações, desde a pesquisa de elementos em listas até a consulta de informações em estruturas de dados mais complexas. Python oferece várias formas de implementar algoritmos de busca, incluindo a busca linear, a busca binária e a busca em dicionários, cada uma adequada para diferentes tipos de coleções e requisitos de desempenho. Esses algoritmos são amplamente utilizados em programação para resolver uma variedade de problemas, desde ordenação e filtragem de dados até a pesquisa em bancos de dados e sistemas de busca na web.

### 

### 7.1.1 Busca sequencial

“Quando itens de dados são armazenados numa coleção tal qual uma lista, nós dizemos que eles têm uma relação linear ou sequencial. Cada item de dado é armazenado numa posição relativa aos demais. Em listas do Python, essas posições relativas são os índices dos itens individuais.” (Pandaime, 2019)

A busca sequencial, também conhecida como busca linear, é um método simples de busca em Python (ou em qualquer linguagem de programação) que envolve percorrer uma sequência de elementos, um por um, até encontrar o valor desejado ou determinar que o valor não está presente.

Exemplo de como realizar uma busca sequencial em Python:

| Texto  Descrição gerada automaticamente |
| --- |

Fonte: Grupo

Neste exemplo, a função busca\_sequencial recebe uma lista e um valor de destino. Ela itera pela lista, comparando cada elemento com o valor de destino. Se o valor de destino for encontrado, a função retorna o índice desse elemento na lista. Se o valor não for encontrado após percorrer toda a lista, a função retorna -1 para indicar que o elemento não está presente.

A busca sequencial é direta e fácil de entender, mas pode ser ineficiente em listas muito grandes, pois é necessário percorrer todos os elementos, por caso o (n), onde "n" é o tamanho da lista.

### 

### 7.1.2 Busca binária

“Os algoritmos de pesquisa (ou de busca) binária também são conhecidos como pesquisa de meio intervalo. Eles retornam a posição do valor pesquisado em uma lista classificada.

Esses algoritmos usam a técnica de "dividir e conquistar" para encontrar a posição do valor.” (Freecodecamp, 2022)

A busca binária em Python é um algoritmo de busca eficiente usado para encontrar um elemento em uma lista ordenada. A principal característica da busca binária é que ela divide repetidamente a lista pela metade, eliminando metade dos elementos a cada iteração, até encontrar o elemento desejado ou determinar que o elemento não está presente. É uma técnica muito mais rápida do que a busca sequencial, especialmente em listas grandes.

Exemplo de como realizar uma busca binária em Python:

| Texto  Descrição gerada automaticamente |
| --- |

Fonte: Grupo

Neste exemplo, a função busca\_binaria recebe uma lista ordenada e um valor de destino. Ela mantém duas variáveis, inicio e fim, que representam a faixa de elementos a serem pesquisados. A cada iteração do loop while, a função calcula o índice do meio da faixa e compara o elemento no meio com o valor de destino. Com base nessa comparação, a função decide se deve procurar na metade esquerda ou direita da lista e atualiza as variáveis inicio e fim de acordo.

A busca binária é muito eficiente, com uma complexidade de tempo de O(log n), onde "n" é o tamanho da lista. Isso a torna ideal para listas grandes, onde a busca sequencial seria muito mais lenta.

### 

### 7.2 Algoritmo de Ordenação

“Ordenar conjuntos de coisas é uma tarefa costumeira em nossa vida. Em se tratando de computadores, como geralmente lidamos com grandes volumes de dados, é desejável ter alguma ordem nesses dados para facilitar sua manipulação.” (Algoritmosempython, 2017)

Algoritmos de ordenação em Python são métodos ou procedimentos usados para organizar os elementos de uma lista (ou array) em uma ordem específica, como crescente ou decrescente. A ordenação é uma tarefa fundamental em ciência da computação e é usada em uma ampla variedade de aplicativos. Existem vários algoritmos de ordenação disponíveis em Python, e cada um deles possui características e desempenhos diferentes.

Algoritmos de ordenação mais comuns em Python:

### 

### 7.2.1 Bubble Sort (Ordenação por Bolha):

-Este é um algoritmo simples que compara pares de elementos adjacentes e os troca se estiverem fora de ordem.

-É ineficiente para listas grandes e não é recomendado para uso em casos reais.

### 7.2.2 Insertion Sort (Ordenação por Inserção):

-Este algoritmo percorre a lista e insere cada elemento na posição correta, deslocando os elementos maiores para abrir espaço.

-É eficiente para pequenas listas e quase ordenadas.

### 7.2.3 Selection Sort (Ordenação por Seleção):

-Este algoritmo encontra o menor elemento da lista e o coloca na posição correta.

-Não é eficiente em listas grandes, pois sempre faz o mesmo número de comparações, independentemente da entrada.

### 7.2.4 Merge Sort (Ordenação por Mesclagem):

-Este é um algoritmo de divisão e conquista que divide a lista em partes menores, ordena essas partes e depois mescla-as em ordem.

-É eficiente para listas grandes e tem um desempenho consistente.

### 7.2.5 Quick Sort (Ordenação Rápida):

-Outro algoritmo de divisão e conquista que escolhe um elemento "pivô", divide a lista em elementos menores e maiores que o pivô, e depois os ordena recursivamente.

-Geralmente é rápido e eficiente na prática.

### 7.2.6 Heap Sort (Ordenação por Heap):

-Este algoritmo utiliza uma estrutura de dados chamada heap para organizar os elementos em ordem crescente.

-Tem um desempenho eficiente e é adequado para listas grandes.

### 7.2.7 Timsort:

-O Timsort é um algoritmo híbrido baseado no Merge Sort e no Insertion Sort. Ele é usado como o algoritmo de ordenação padrão no Python e é otimizado para diferentes tipos de entradas.

Para usar esses algoritmos em Python, você pode implementá-los manualmente ou usar as funções de ordenação incorporadas, como sorted() e list.sort(). A escolha do algoritmo de ordenação depende das características da sua lista (tamanho, estado inicial, etc.) e dos requisitos de desempenho do seu aplicativo. Em muitos casos, as funções de ordenação incorporadas do Python serão suficientes e eficientes para suas necessidades.

### 

### 7.3 Listas

“Ao longo da vida de uma pessoa desenvolvedora, independentemente da plataforma e linguagem, é comum a necessidade de lidar com listas. Um dos exemplos mais comuns de utilização de lista é o armazenamento de dados dos usuários de uma aplicação, como os nomes e endereços. A lista é uma estrutura de dados composta por elementos organizados de forma linear, na qual cada um pode ser acessado a partir de um índice, que representa sua posição na coleção (iniciando em zero).” (Kenzie, 2022)

Em Python, uma lista é uma estrutura de dados que permite armazenar uma coleção ordenada de elementos. Esses elementos podem ser de diferentes tipos, como números, strings, objetos, outras listas e assim por diante. As listas são mutáveis, o que significa que você pode adicionar, remover ou modificar elementos após a criação da lista.

Exemplo simples de uma lista em Python:

|  |
| --- |

Fonte: Grupo

Neste exemplo, ‘minha\_lista’ é uma lista que contém cinco números inteiros. Você pode acessar os elementos da lista usando índices. Por exemplo:

| Tela preta com letras brancas  Descrição gerada automaticamente |
| --- |

Fonte: Grupo

Você também pode realizar várias operações em listas, como adicionar elementos, remover elementos, verificar o tamanho da lista e muito mais. Abaixo algumas operações comuns em listas:

| Texto  Descrição gerada automaticamente |
| --- |

Fonte: Grupo

As listas são uma das estruturas de dados mais versáteis em Python e são amplamente usadas para armazenar coleções de dados de diferentes tipos. Elas desempenham um papel fundamental na programação Python e são uma escolha comum para muitas tarefas de manipulação de dados e algoritmos.

### 

### 7.3.1 Tuplas

“O Python nos disponibiliza um tipo que preenche muito melhor as características que procurávamos — as tuplas. Tuplas podem ser consideradas similares às listas, mas suas diferenças são cruciais.” (Alura, 2023)

Em Python, uma tupla é uma estrutura de dados semelhante a uma lista, mas com uma diferença fundamental: as tuplas são imutáveis, enquanto as listas são mutáveis. Isso significa que, uma vez que uma tupla é criada, seus elementos não podem ser alterados, adicionados ou removidos. As tuplas são frequentemente usadas para armazenar coleções ordenadas de elementos que não devem ser modificados.

Exemplo de uma tupla em Python:

|  |
| --- |

Fonte: Grupo

Assim como nas listas, você pode acessar os elementos de uma tupla usando índices:

| Texto  Descrição gerada automaticamente |
| --- |

Fonte: Grupo

No entanto, você não pode modificar os elementos de uma tupla depois de criá-la. Por exemplo, a seguinte operação geraria um erro:

| Texto  Descrição gerada automaticamente com confiança média |
| --- |

Fonte: Grupo

As tuplas são frequentemente usadas em situações em que você deseja garantir que os dados não sejam acidentalmente modificados. Além disso, elas podem ser usadas como chaves em dicionários (enquanto listas não podem, porque são mutáveis), o que as torna úteis em estruturas de dados mais complexas.

Você pode criar uma tupla com ou sem parênteses. Por exemplo, as duas formas abaixo são equivalentes:

| Tela preta com letras brancas  Descrição gerada automaticamente |
| --- |

Fonte: Grupo

Em muitos casos, as tuplas são usadas para representar coleções de dados relacionados, como coordenadas (latitude e longitude), data e hora (ano, mês, dia, hora, minuto, segundo), e assim por diante.

### 

### 7.3.2 Dicionário

“Um dicionário Python é uma forma de coleção de dados em que se guarda uma chave e um valor correspondente. É similar a um dicionário mesmo, em que há sempre um termo e uma tradução.” (Hashtagtreinamentos, 2023)

Em Python, um dicionário é uma estrutura de dados que permite armazenar um conjunto de pares chave-valor. Cada elemento em um dicionário consiste em uma chave única associada a um valor correspondente. Os dicionários são uma das estruturas de dados mais versáteis e amplamente utilizadas na linguagem.

Exemplo simples de um dicionário em Python:

|  |
| --- |

Fonte: Grupo

Neste exemplo, meu\_dicionário é um dicionário que contém três pares chave-valor. As chaves são "nome", "idade" e "cidade", e os valores associados a essas chaves são "João", 30 e "São Paulo", respectivamente.

Você pode acessar os valores em um dicionário usando as chaves correspondentes:

| Tela de celular com aplicativo aberto  Descrição gerada automaticamente com confiança média |
| --- |

Fonte: Grupo

Os valores em um dicionário podem ser de qualquer tipo de dado, incluindo números, strings, listas, tuplas e até mesmo outros dicionários. Além disso, as chaves de um dicionário devem ser únicas, mas os valores podem se repetir.

Você pode realizar várias operações em dicionários, como adicionar novos pares chave-valor, modificar valores existentes, remover pares chave-valor, verificar se uma chave está presente no dicionário e muito mais. Aqui estão algumas operações comuns em dicionários:

| Texto  Descrição gerada automaticamente |
| --- |

Fonte: Grupo

Os dicionários são especialmente úteis quando você precisa associar informações relacionadas entre si. Eles são amplamente usados em Python para representar configurações, armazenar dados estruturados e organizar informações de várias maneiras eficientes.

### 

### 7.3.3 Sets

“Os sets são uma coleção de itens desordenada, parcialmente imutável e que não podem conter elementos duplicados. Por ser parcialmente imutável, os sets possuem permissão de adição e remoção de elementos.” (FagnerPinheiro, 2020)

Em Python, um conjunto (set) é uma estrutura de dados que representa uma coleção não ordenada e sem elementos duplicados. Os conjuntos são usados quando você deseja armazenar um grupo de valores únicos, e eles são bastante eficientes para verificar a existência de um elemento em uma coleção sem a necessidade de percorrer toda a coleção. Os elementos de um conjunto podem ser de vários tipos de dados, como números, strings e outros objetos.

Exemplo de como criar e usar um conjunto em Python:

| Texto  Descrição gerada automaticamente |
| --- |

Fonte: Grupo

Os conjuntos são úteis quando você deseja eliminar duplicatas de uma lista de elementos, verificar a associação de elementos a um conjunto ou realizar operações de conjunto, como união, interseção e diferença.

Exemplo de operações de conjunto em Python:

| Texto  Descrição gerada automaticamente |
| --- |

Fonte: Grupo

Os conjuntos também podem ser criados usando a função set() e podem ser convertidos de e para listas e outros iteráveis usando a função set() ou list(), respectivamente.

### 

### 

### 8 Python Orientado à Objetos

“Python já nasceu sendo uma linguagem de programação multi-paradigma, isto é: é possível programar em Python de maneira Imperativa, Funcional e também no paradigma utilizando conceitos da Programação Orientada a Objetos.” (PythonAcademy, 2023)

A Programação Orientada a Objetos (POO) é um paradigma de programação que se baseia no conceito de classes e objetos. Essa abordagem de programação permite que os desenvolvedores consigam modelar o mundo real em seus programas de computador, representando entidades, atributos e comportamentos como objetos.

As **classes** podem conter dados na forma de campos que podemos chamar de atributos ou propriedades e código na forma de procedimentos, que podemos chamar de métodos. Uma característica fundamental dos **objetos** é que eles têm a capacidade de acessar e muitas vezes modificar seus próprios campos de dados; os objetos mantêm uma referência a si mesmos, frequentemente denotada como o atributo "self" em Python.

Na Programação Orientada a Objetos (POO), os programas são concebidos com base na interação entre objetos. Este paradigma coloca ênfase nos objetos que os desenvolvedores desejam manipular, em vez de focar principalmente na lógica necessária para efetuar essas manipulações. Essa abordagem de programação é particularmente adequada para o desenvolvimento de programas extensos, complexos e que precisam ser constantemente atualizados ou mantidos.

### 

### 8.1Bibliotecas, Módulos e Pacotes em Python

“a biblioteca padrão Python é um conjunto de módulos disponíveis em Python para que você possa importá-los e usar as funcionalidades deles quando bem quiser.” (Algoritmos em Python)

Com base na citação acima, concluímos que as bibliotecas em Python são conjuntos de módulos e funções pré-desenvolvidos que aumentam a funcionalidade da linguagem. Elas são essenciais para o desenvolvimento de aplicativos Python, pois oferecem várias funcionalidades prontas para uso.

*“A biblioteca padrão do Python é muito extensa, oferecendo uma ampla gama de recursos” (Python Org)*

Temos dois tipos de bibliotecas, sendo eles:

- Bibliotecas Padrão:

*“Python possui bibliotecas padrão, coleções de módulos e scripts acessíveis que facilitam os processos de criação” (Geo Sem Fronteiras)*

Alguns exemplos de tarefas são: manipulação de arquivos, comunicação em rede, processamento de strings, cálculos matemáticos, manipulação de datas, etc.

Esses módulos estão prontamente disponíveis e não exigem instalação adicional.

-Bibliotecas Externas:

“*A instalação de bibliotecas externas é uma etapa importante para qualquer projeto em Python que necessite de recursos adicionais para executar tarefas específicas.” (Gustavo Soares, Dev, 2023)*

Além da biblioteca padrão, Python possui uma ampla coleção de bibliotecas externas, também chamadas de pacotes ou módulos de terceiros. Essas bibliotecas são desenvolvidas e mantidas pela comunidade Python e abrangem vários domínios, incluindo ciência de dados, aprendizado de máquina, desenvolvimento web, automação, jogos, etc.

Ao falarmos sobre as bibliotecas em python, ouvimos também sobre o “gerenciamento de pacotes”, que é uma parte fundamental do desenvolvimento de software desta linguagem.

*“não é viável que a instalação dessas bibliotecas seja feita de forma manual, já* *que o processo de cada uma delas podem ser, no mínimo, complicadas.” (Fagner Pinheiro, Treina Web)*

O gerenciamento de pacotes facilita a instalação, atualização e distribuição de bibliotecas, módulos e pacotes de terceiros que podem ser usados em seus projetos. O principal sistema de gerenciamento de pacotes em Python é o "pip", que é amplamente utilizado pela comunidade de desenvolvedores Python.

Em relação a documentação, a maioria das bibliotecas possui documentação oficial detalhada que oferece informações sobre como utilizar suas funções e classes. Ela é uma fonte valiosa de informações para programadores que trabalham com essas bibliotecas.

### 

### 8.1.1 Exemplos de Bibliotecas Populares:

-NumPy: Usado para computação numérica e manipulação de arrays multidimensionais.

-Pandas: Ideal para análise e manipulação de dados tabulares.

-Matplotlib e Seaborn: Utilizados para criação de visualizações de dados.

-SciPyGenericName: Oferece recursos avançados para a computação científica no ambiente Python.

-TensorFlow: Disponibiliza recursos para realizar aprendizado de máquina avançado e implementar técnicas de aprendizado profundo.

No exemplo de biblioteca padrão do python, usamos a “random" ela fornece funções para gerar números aleatórios e é comumente usada em muitos programas para introduzir aleatoriedade em processos, como simulações, jogos e experimentos estatísticos.

| Texto  Descrição gerada automaticamente |
| --- |

Fonte: Grupo

Em relação aos módulos, concluímos que em Python, um módulo representa um arquivo que contém código incluindo definições de funções, classes e variáveis que podem ser utilizadas em outros programas Python.

“Um módulo nada mais é do que um arquivo .py com instruções e definições em Python” (Algoritmos em Python)

Os módulos possuem um papel importante na organização do código em Python, permitindo a divisão de um programa em partes menores e reutilizáveis.

Para utilizar um módulo em um programa Python, você precisa importá-lo usando a palavra-chave “import”. Por exemplo, se você possui um módulo chamado “meu\_modulo.py”, pode importá-lo da seguinte maneira:

| Texto  Descrição gerada automaticamente |
| --- |

Fonte: Pense Python: “Módulos e Obtendo Ajuda”, 2023

Seguindo esse código, teríamos o seguinte resultado:

| Forma  Descrição gerada automaticamente com confiança baixa |
| --- |

Fonte: Pense Python: “Módulos e Obtendo Ajuda”, 2023

Python também inclui uma ampla biblioteca de módulos padrão que oferecem funcionalidades prontas para uso, como math, random, os, datetime, entre outros. O uso de módulos ajuda a organizar seu código de maneira mais estruturada e modular.

*“Lembre-se que uma vez que o módulo é importado, podemos utilizar as coisas* *que estão definidas dentro dele.” (Pense Python, 2023)*

Módulos permitem que você reutilize código em diferentes projetos. Depois de definir e testar um módulo, você pode usá-lo em diferentes programas sem a necessidade de reescrever o código.

Quando um diretório contém um arquivo chamado \_\_init\_\_.py, ele é considerado um pacote Python. Isso possibilita a organização de módulos relacionados em pacotes e a importação deles como pacote.módulo.

Em continuidade à esse último parágrafo, agora, falaremos em relação aos pacotes em Python.

Dizemos que é uma maneira de organizar módulos relacionados em diretórios. Isso ajuda a estruturar e gerenciar seu código de maneira mais eficaz quando você tem um grande número de módulos.

*“Ao criarmos um pacote, além de organizarmos nosso programa, temos a possibilidade de importarmos os módulos de forma bastante flexível.” (DevFuria, 2023)*

Um pacote é essencialmente um diretório que contém um arquivo especial chamado “\_\_init\_\_.py” (usaremos esse arquivo ao decorrer do trabalho).

Esse arquivo indica ao Python que o diretório deve ser tratado como um pacote. O arquivo “\_\_init\_\_.py” pode estar vazio ou conter código de inicialização para o pacote.

*“Isso é feito para evitar diretórios com o mesmo nome, de ocultar involuntariamente módulos válidos que ocorrem mais tarde no caminho de pesquisa do módulo.” (Candido, School Of Net, 2018)*

Os pacotes geralmente contêm módulos relacionados e para importa-los você precisa usar a notação de ponto. Podem conter subpacotes, que são diretórios dentro de diretórios.

*“Um pacote pode conter vários outros módulos, chamamos de submódulos.” (DevFuria, 2023)*

Resumindo, pacotes em Python são uma maneira eficaz de organizar e estruturar projetos de código maiores, tornando o código mais legível e fácil de manter, ao mesmo tempo em que facilitam a reutilização de funcionalidades específicas em diferentes partes do seu programa.

### 

### 8.2Aplicação de Banco de Dados com Python

Um banco de dados é um repositório organizado de informações ou dados estruturados, geralmente mantido em formato eletrônico em um sistema de computador.

“é um repositório para armazenar informações (dados) de qualquer natureza. Ele retrata aspectos do mundo real — ou seja, o conceito visto anteriormente de minimundo — em que qualquer mudança que se faça no minimundo é diretamente replicada no banco de dados.” (Fundação Bradesco, Escola Virtual, 2023).

Suas vantagens são: velocidade, produtividade, precisão, redução de riscos, proteção,

Os bancos de dados relacionais desempenham um papel fundamental no ecossistema de desenvolvimento Python, permitindo que aplicativos armazenem e gerenciem dados de forma estruturada e eficaz. Python oferece diversas opções para interagir com bancos de dados relacionais, e aqui estão as principais:

-Biblioteca SQLite da Biblioteca Padrão: Python inclui a biblioteca SQLite, que é um sistema de gerenciamento de banco de dados SQL incorporado. É adequado para aplicativos menores e locais, sendo uma escolha conveniente para prototipagem rápida e para situações em que não é necessário um servidor de banco de dados separado.

**-ORMs (Mapeamento Objeto-Relacional):** Python oferece diversos ORMs populares, como SQLAlchemy, Django ORM e Peewee. Essas ferramentas permitem mapear objetos Python para tabelas em bancos de dados, simplificando o trabalho com dados por meio de uma abordagem mais orientada a objetos, em vez de escrever SQL manualmente.

**-Bibliotecas de Conexão Direta:** Existem bibliotecas de terceiros que possibilitam a conexão direta com bancos de dados relacionais usando SQL nativo. Exemplos incluem psycopg2 para PostgreSQL, mysql-connector-python para MySQL e pyodbc para SQL Server.

**-Frameworks Web:** Alguns frameworks web, como Flask e Django, fornecem suporte integrado para lidar com bancos de dados relacionais. Isso simplifica o desenvolvimento de aplicativos da web que interagem com bancos de dados.

**-Bancos** de Dados NoSQL: Embora não sejam bancos de dados relacionais, as opções NoSQL, como MongoDB e Cassandra, também são populares na comunidade Python. Existem bibliotecas e pacotes que facilitam a integração desses bancos de dados com Python.

No geral, a capacidade do Python de se integrar facilmente com bancos de dados relacionais é uma das razões pelas quais a linguagem é tão amplamente adotada para o desenvolvimento web e a análise de dados. A escolha da abordagem e da biblioteca depende das necessidades particulares de cada projeto e do conhecimento prévio do desenvolvedor.

### 8.3 Princípios de POO

*“No paradigma orientado à objetos, uma classe é a representação de algo do mundo real. No Python, o uso de classes é algo constante no desenvolvimento de programas.” (TreinaWeb, 2021)*

### 

### 8.3.1 Classes

São estruturas de dados criadas pelo desenvolvedor que funcionam como um blueprint (plano detalhado ou modelo que descreve como algo deve ser construído, desenvolvido ou executado) para objetos.

Exemplo:

| Uma imagem contendo relógio, escuro, bola, medidor  Descrição gerada automaticamente |
| --- |

Fonte: Grupo

Uma classe é composta por atributos e métodos. Os atributos de uma classe refletem as suas características, enquanto os métodos descrevem o comportamento associado à classe.

Para declarar um atributo em uma classe em Python, é necessário especificar o nome do atributo no método especial chamado “\_\_init\_\_”. Este método é responsável por definir o construtor da classe, ou seja, é onde determinamos como criar uma nova instância da classe em nosso programa.

Exemplo:

| Interface gráfica do usuário, Site  Descrição gerada automaticamente |
| --- |

Fonte: Grupo

Agora, estamos estabelecendo que toda pessoa criada em nosso programa, com base na classe Pessoa, deve obrigatoriamente possuir um nome.

Agora, vamos definir um método de saudação para a classe “pessoas”. Pra isso, usamos a palavra “def”, isso indica que estamos definindo um método para a classe, incluindo o nome do método e, se houver, seus atributos.

Exemplo:

| Interface gráfica do usuário, Texto  Descrição gerada automaticamente com confiança média |
| --- |

Fonte: Grupo

### 

### 8.3.2 Objetos

Os objetos são instâncias de classes. Eles são criados com base no modelo fornecido pela classe e podem armazenar dados e realizar ações. Por exemplo, um objeto específico pode ser um carro com uma marca e um modelo específicos.

*“Sempre que precisamos criar algo com base em uma classe, dizemos que estamos instanciando objetos.” (TreinaWeb, 2021)*

Essa citação acima relata que estamos gerando a representação de uma classe dentro do nosso programa.

Para criar uma instância de um objeto em Python, com base em uma classe que foi previamente definida, é preciso especificar a classe que queremos usar como modelo e, se necessário, fornecer os valores para seus atributos.

Exemplo:

| Texto  Descrição gerada automaticamente |
| --- |

Fonte: Grupo

Quando executamos a linha “pessoa1 = Pessoa("Thiago")”, estamos gerando uma instância de um objeto do tipo Pessoa, fazendo receber o nome “Thiago”.

Por fim, criamos a linha que faz a execução desse processo todo.

Exemplo:

| Interface gráfica do usuário  Descrição gerada automaticamente com confiança média |
| --- |

Fonte: Grupo

Agora é só executar o programa! Abaixo temos a imagem completa dos exemplos acima.

| Texto  Descrição gerada automaticamente |
| --- |

Fonte: Grupo

### 8.3.3 Encapsulamento

Esse princípio envolve a ideia de que os detalhes internos de uma classe devem ser ocultados do mundo exterior. Todos os atributos e comportamentos dessa entidade são agregados à sua classe, geralmente são definidos como privados e só podem ser acessados por métodos públicos dessa mesma classe.

*“O princípio do Encapsulamento também afirma que informações importantes devem ser contidas dentro do objeto de maneira privada e apenas informações selecionadas devem ser expostas publicamente.” (Python Academy, 2023)*

Essa característica de ocultação de dados oferece maior segurança ao programa e previne alterações não autorizadas nos dados.

-**Private (Privado):** Os elementos de uma classe (métodos ou variáveis) marcados como privados não estão disponíveis para acesso fora da classe em que estão definidos. Eles são acessíveis apenas internamente na classe.

-**Protected (Protegido):** Os membros marcados como protegidos podem ser acessados tanto dentro da própria classe quanto por classes derivadas (subclasses) da classe que os engloba.

-**Public (Público):** Os membros públicos podem ser acessados de qualquer lugar, tanto dentro como fora da classe. Eles definem a interface pública da classe.

Exemplo de Encapsulamento Simples:

1. Criamos a classe “Lista de Tarefas” com uma variável privada para armazenar as tarefas e o método construtor “\_\_init\_\_” que vimos anteriormente.

| Texto  Descrição gerada automaticamente com confiança média |
| --- |

Fonte: Grupo

Aqui, estamos definindo a variável “\_\_tarefas” como uma lista vazia dentro do construtor da classe.

1. Adicione o método “adicionar\_tarefa” para permitir adicionar tarefas à lista. Também usamos o método “append” que é um recurso em Python que permite incluir um novo elemento no final de uma lista já existente.

*“Essa função é extremamente útil quando você precisa expandir uma lista com mais itens dinamicamente, sem precisar criar uma nova lista a cada vez.” (Awari, 2023.)*

| Texto  Descrição gerada automaticamente |
| --- |

Fonte: Grupo

Esse método recebe uma tarefa como argumento e a adiciona à lista de tarefas.

1. Adicionamos o método “listar\_tarefas” para listar todas as tarefas na lista. Usamos o método “enumerate” nessa etapa.

*“A função "enumerate()" é útil quando precisamos acessar tanto o valor quanto o índice de cada elemento da lista.” (Emmanuel Adrade, Dio, 2023)*

| Interface gráfica do usuário, Texto  Descrição gerada automaticamente |
| --- |

Fonte: Grupo

Aqui já temos uma classe base com encapsulamento que pode adicionar e listar tarefas.  
Utilizamos a estrutura “for” que executa o(s) código(s) contidos nela repetidamente, e “in” que é utilizada para determinar se um valor está incluso ou não em um conjunto de valores.

### 8.3.4 Herança

*“A Herança é um conceito do paradigma da orientação à objetos que determina que uma classe (filha) pode herdar atributos e métodos de uma outra classe (pai) e, assim, evitar que haja muita repetição de código.” (TreinaWeb, 2023)*

De acordo com a citação acima, a “classe filha” é chamada de:

-classe derivada

-subclasse

Enquanto a “classe pai” é chamada de:

-classe base

-superclasse

Isso facilita a reutilização de código e a criação de hierarquias de classes.

1. Aplicando a Herança, vamos criar uma classe ListaPrioritaria que herda da ListaDeTarefas. A herança é estabelecida pela colocação da classe pai entre parênteses. Utilizamos a função “super()” que é usada para chamar métodos da classe pai (superclasse) em uma hierarquia de herança.

| Texto  Descrição gerada automaticamente |
| --- |

Fonte: Grupo

É importante ressaltar que estamos sobrescrevendo o método “listar\_tarefas” para incluir informações de prioridade.

### 

### 8.3.5 Polimorfismo

*“é a capacidade que uma subclasse tem de ter métodos com o mesmo nome de sua superclasse, e o programa saber qual método deve ser invocado, especificamente (da super ou sub).” (Samuel O. N. M. Filho, Forum Alura, 2022)*

Isso significa que várias classes podem ter métodos com o mesmo nome, mas com comportamentos diferentes, o que facilita a criação de sistemas flexíveis e extensíveis.

Aplicando o Polimorfismo, criaremos uma segunda subclasse chamada “ListaComData” que também herda de “ListaDeTarefas”. Esta classe adiciona a capacidade de atribuir datas de vencimento às tarefas.

| Texto  Descrição gerada automaticamente |
| --- |

Fonte: Grupo

Nessa etapa, usamos função “zip()” que é usada para juntar duas ou mais listas em uma única lista de tuplas, onde cada tupla contém um elemento de cada uma das listas fornecidas.

Novamente, estamos sobrescrevendo o método “listar\_tarefas”, mas agora para incluir informações sobre a data de vencimento.

1. Até aqui, já temos as três classes. Agora criaremos as instâncias e aplicar o polimorfismo. Criamos instâncias de cada classe e usando os métodos para adicionar e listar tarefas. Apesar das instâncias terem comportamentos diferentes, todas estão sendo tratadas do mesmo jeito, aqui encontramos o polimorfismo.

| Texto  Descrição gerada automaticamente |
| --- |

Fonte: Grupo

Segue abaixo o código completo:

| Texto  Descrição gerada automaticamente |
| --- |

Fonte: Grupo

| Texto  Descrição gerada automaticamente |
| --- |

Fonte: Grupo

A POO é amplamente utilizada em muitas linguagens de programação, como Java, C++, Python, C#, e muitas outras, devido à sua capacidade de organizar o código de forma mais modular, facilitando a manutenção e o desenvolvimento de software mais escalável e reutilizável.

Por fim, usando um pouco do que vimos anteriormente, temos mais um exemplo de como usar a POO.

| Tela de computador com letras e números em fundo preto  Descrição gerada automaticamente |
| --- |

Fonte: Grupo

Neste exemplo, criamos a classe e o construtor, definimos o método e criamos a instância da classe e por fim chamamos o método para impressão da imagem.

### 9 Introdução à análise de dados com Python

A análise de dados com Python é uma habilidade essencial para aqueles que

desejam obter informações significativas de conjuntos de dados e tomar decisões

informadas. A Analítica de Dados com Python pode ser usada por analistas de

dados na exploração para escrever scripts e funções customizadas, manipular

os dados, otimizar fluxos de trabalho e gerar visualizações de dados. Além

disso, o Python está se tornando cada vez mais popular para desenvolver

código para aplicativos personalizados de aprendizado de máquina e análise

preditiva, sendo assim uma linguagem de programação poderosa e amplamente

utilizada.

“Para muitas pessoas, a linguagem de programação Python tem um forte apelo. Desde o seu surgimento em 1991, Python se tornou uma das linguagens de programação interpretadas mais populares, junto com Perl, Ruby e outras. Python e Ruby se tornaram especialmente populares por volta de 2005 para a construção de sites, com seus diversos frameworks web como Rails (Ruby) e Django (Python). Linguagens como essas com frequência são chamadas de linguagens de scripting, pois podem ser usadas para escrever rapidamente pequenos programas ou scripts para automatizar outras tarefas. Não gosto do termo “linguagem de scripting”, pois carrega uma conotação de que essas linguagens não poderiam ser usadas para a construção de softwares sérios. Entre as linguagens interpretadas, por diversos motivos históricos e culturais, Python desenvolveu uma comunidade grande e ativa de processamento científico e análise de dados. Nos últimos dez anos, Python passou de uma linguagem de computação científica inovadora, ou para ser usada “por sua própria conta e risco”, para uma das linguagens mais importantes em ciência de dados, aprendizado de máquina (machine learning) e desenvolvimento de softwares em geral, no ambiente acadêmico e no mercado”(Python Para Análise de Dados, Wes McKinney, 30 de dezembro de 2011)

Por que usar Python para análise de dados?

Python é uma escolha popular para análise de dados por várias razões:

Linguagem de programação acessível: Python é conhecido por sua sintaxe simples e legível, o que o torna acessível até mesmo para iniciantes em programação.

Ampla comunidade e suporte: Há uma comunidade ativa de desenvolvedores que oferece bibliotecas, tutoriais e recursos online para análise de dados em Python.

Bibliotecas de análise de dados: Python oferece várias bibliotecas poderosas, como Pandas, NumPy, Matplotlib, Seaborn e Scikit-Learn, que simplificam tarefas comuns de análise de dados.

Passos básicos de análise de dados com Python:

### 

### 9.0.1 Coleta de dados

O primeiro passo na análise de dados é coletar os dados relevantes. Os dados podem ser provenientes de diversas fontes, como bancos de dados, arquivos CSV, APIs da web, planilhas, entre outros. Você pode usar bibliotecas Python, como Pandas, para importar e manipular esses dados.

|  |
| --- |

FONTE: GRUPO

### 

### 9.0.2 Limpeza e preparação de dados

Os dados nem sempre estão em formato perfeito. É comum haver valores ausentes, duplicados ou inconsistentes que precisam ser tratados. Você também pode precisar fazer transformações nos dados para torná-los adequados para análise.

Copiar código

|  |
| --- |

FONTE: GRUPO.

### 

### 9.0.3 Exploração de dados

Neste estágio, você explora os dados para obter insights iniciais. Isso inclui a criação de visualizações, estatísticas descritivas e a identificação de tendências e padrões nos dados.

Copiar código

|  |
| --- |

FONTE:GRUPO

### 

### 9.0.4 Análise de dados

A análise de dados envolve métodos estatísticos e algoritmos para responder a perguntas específicas ou fazer aplicação com base nos dados. Bibliotecas como Scikit-Learn são úteis para tarefas de aprendizado de máquina e modelagem preditiva.

Copiar código

|  |
| --- |

FONTE:GRUPO

### 

### 9.0.5 Comunicação de resultados

Comunicar os resultados da análise é crucial. Isso pode ser feito por meio de relatórios, visualizações de dados interativos ou apresentações. Bibliotecas como Matplotlib e Seaborn podem ajudar a criar gráficos informativos.

Essa é apenas uma introdução à análise de dados com Python. À medida que você avança, você pode aprofundar seus conhecimentos e explorar tópicos avançados, como análise de séries temporais, aprendizado de máquina, processamento de linguagem natural (PNL) e muito mais. A prática é fundamental, então comece a trabalhar em projetos de análise de dados para aprimorar suas habilidades.

### 

### 9.1 Introdução a biblioteca Pandas.

“O pandas oferece estruturas de dados de alto nível e funções, projetadas para fazer com que trabalhar com dados estruturados ou tabulares seja rápido, fácil e expressivo. Desde o seu surgimento em 2010, o pandas tem ajudado a viabilizar o Python como um ambiente eficaz e produtivo para análise de dados. Os principais objetos do pandas usados neste livro são o Data Frame – uma estrutura de dados tabular, orientada a colunas, com rótulos (labels) tanto para linhas quanto para colunas – e as Séries – um objeto array unidimensional, com rótulo.”(Python Para Análise de Dados, Wes McKinney, 30 de dezembro de 2011)

Uma das bibliotecas mais importantes para entender como usar dados em Python principais tarefas, bem como utilizar bibliotecas para manter o processamento de gráficos digitais e usados. forneça uma breve introdução e execute uma biblioteca hoje.

Uma estrutura de dados estrutura chamada Séries ou Data Frame pode ser criada e modificada usando a Biblioteca Panda .Uma série ou Data Frame pode ser criada e modificada usando a Biblioteca Panda .

Séries é uma matriz de objetos unidimensional (como a matriz Numpy ) que contém um índice (os resultados das observações) . considerado como uma coleção de dados retangulares ou tabulares em que cada coluna contém um tipo específico de dados, denotado por um índice de colunas e um índice para cada observação (linha).

importar a biblioteca , aplicamos a seguinte convenção :

Séries

Uma matriz unidimensional vinculada a uma tabela de dados chamada "índice" ( ou "índice"). Podemos criar uma série usando o seguinte código .código.

|  |
| --- |

FONTE:GRUPO

0 4

1 5

2 6

3 3

dtype: int64

Veja que existe um número representando a ordem das observações ao lado de um número valor, representando a ordem das observações ao lado de cada valor. possível criar os valores de um índice

|  |
| --- |

.FONTE:GRUPO

a 4

b 7

c 8

d -2

dtype: int64

Objeto de série pode ser feito de forma semelhante a um array no Numpy, seja através de funções, multiplicandos escalonados ,ou mesmo filtragens. A ordem do índice não muda. ser feito através da rotação da observação para permitir a seleção.

|  |
| --- |

FONTE:GRUPO

Como dito, o Data Frame simboliza uma tabela de dados retangular tabela de dados com uma coleção de colunas que são pré-preenchidas com uma série de observações , sejam elas numéricas, textuais, booleanas, etc.uma coleção de colunas pré -preenchidas com uma série de observações , sejam elas numéricas , textuais, booleanas , etc. Além disso, tem dois índices, um para observações e outro duas colunas, descreve-o como uma coleção de listas, dicionários ou matrizes unidimensionais. A partir de uma coleção de dicionários , construiremos um DataFrame.

|  |
| --- |

FONTE:GRUPO

Além disso , é possível importar dados de outros locais começando pelos pandas. Outros locais começam com pandas. O método mais popular método é importar.csv a importar arquivos.csv .archivos. Vemos um exemplo abaixo, importando um conjunto de dados de treinamento para uma coleção de dados sobre preços, um exemplo casas na Califórnia do Google Cola Baixo, importando um conjunto de dados de treinamento para coleta de dados sobre preços de casas na Califórnia Google Colab.

|  |
| --- |

FONTE:GRUPO

Recebemos essa coleção de dados, o interessante a fazer é investigá-la e entender seu conteúdo. use uma variedade de métodos para fazer isso.

|  |
| --- |

FONTE:GRUPO

|  |
| --- |

FONTE:GRUPO

|  |
| --- |

FONTE:GRUPO

Também é possível tirar os números dos índices e os nomes das colunas

|  |
| --- |

FONTE:GRUPO

Depois, demos uma breve introdução aos pandas , mas também há uma série de possibilidades adicionais de manipulação e cálculo . Isso inclui filtragem de DataFrame mais sofisticada , agregação e fusão com condições, bem como métodos para configurar um Data Frame para modelagem e visualização.

### 

### 9.2 Introdução a Scikit Learning

Sua construção é baseada na interação com outras bibliotecas Python, numéricas e científicas, como NumPy e SciPy e Matplotlib, e é considerada uma das soluções mais versáteis e populares do mercado.

Para ajuste, seleção e avaliação de modelo , bem como pré-processamento de dados , Scikit-learn é considerada a biblioteca mais prática e confiável para treinamento de aprendizado de máquina (ML) em Python.

Biblioteca de alto nível , permite a definição de modelos de dados preditivos usando apenas algumas linhas de código . Nesse sentido, o scikit-learn é bem documentado , comparativamente simples de aprender e útil se você estiver procurando uma introdução ao aprendizado de máquina.

O Scikit-Learn foi desenvolvido como Google Summer of \_projeto Projeto de código e foi idealizado e foi idealizado pelo cientista de dados David Carnapau.pelo cientista de dados David Carnapau. O o nome da biblioteca é derivado é derivado da frase " SciKit" (um SciPy Toolkit ), de extensão de terceiros do SciPy que é vendida separadamente.a frase "SciKit" (um SciPy Toolkit), uma extensão de terceiros do que é vendido separadamente.

Quais são as principais aplicações do Scikit Learn?

O Scikit-learn fornece uma ampla gama de algoritmos integrados que algoritmos podem ser usados ​​para utilizar totalmente projetos de ciência de dados .Usado para utilizar totalmente projetos de ciência de dados . Listamos abaixo seus principais pontos abaixo, juntamente com a finalidade de cada um;

1. Algoritmos de Classificação (Classification)

2. Algoritmos de Regressão

3. Algoritmos de Agrupamento (Clustering)

4. Redução de dimensionalidade (Dimensionality reduction)

5. Seleção de Modelo (Model selection)

6. Pré-processamento (Preprocessing)

As ferramentas de pré-processamento do scikit-learn são cruciais para extrair e normalizar recursos durante a análise de dados crucial para extrair e normalizar recursos durante a análise de dados .

Essas ferramentas podem ser usadas no contexto da ciência de dados para modificar dados de entrada ,da ciência de dados como modificar dados de entrada , como texto, e aplicar seus recursos à análise .texto e aplicar seus recursos à análise.

importantes da biblioteca recursos Scikit-aprender

Você aprenderá sobre algumas das funções básicas que o pacote oferece para desenvolver modelos confiáveis ​​e eficazes : algumas das funções básicas que. O pacote fornece para desenvolver modelos de ML confiáveis ​​e eficazes.

O que o Scikit-Learn faz?

Scikit-learn é um grátis,biblioteca biblioteca de código aberto para aprendizado de máquina Python (aprendizado de máquina ) .para aprendizado de máquina Python ( aprendizado de aprendizado de máquina ). variedade de ferramentas eficazes para mineração de dados , análise e modelagem estatística , além de suporte para aprendizagem supervisionada e não supervisionada .

Uma biblioteca de aprendizado de máquina.

Vamos listar algumas:

1. NumPy

2.Pandas

3.SciPy

4.Stats Models

5.Matplotlib

6.Seaborn

7.Plotly

8.Bokeh

9.Pydot

10.Dash

### 

### 9.3 Visualização de dados

O que é visualização de dados ,visualização exatamente?

visualização de dados o processo de representação de dados com elementos visuais , como elementos diagramas,como diagramas, gráficos ou mapas.gráficos ou mapas. Ela converte dados complicados, de grande volume ou numéricos em uma representação visual mais fácil de processar . visualizar As ferramentas ferramentas de dados melhoram e automatizam o processo de comunicação visual para obter precisão e detalhes finos .melhorar e automatizar o processo de comunicação visual para obter precisão e detalhes finos . representações podem ser usadas para extrair insights acionáveis ​​de dados brutos .

Por que a visualização de dados é tão crucial?

A maioria das empresas modernas empresas processo processar enormes quantidades de dados de muitas fontes de dados , como as seguintes :enormes quantidades de dados de muitas fontes de dados , como as seguintes :

e externo sites internos sites

dispositivos com inteligência

sistemas internos de coleta de dados sistemas de coleta

meios de comunicação Redes sociais

No entanto, os dados brutos podem ser difíceis de compreender e usar.Pode ser difícil de entender e usar. Portanto, os cientistas prepararam e apresentaram os dados no contexto apropriado .Preparou e apresentou os dados no contexto apropriado . Eles fornecer um formato visual para que formato tomadores de decisão para que os tomadores de decisão possam ver como as relações entre os dados se relacionam com padrões ou tendências ocultas .pode ver como as relações entre os dados se relacionam com padrões ou tendências ocultas . a visualização se desenvolve de dados narrativas que apoiam a inteligência corporativa , a tomada de decisões e o planejamento estratégico com base emo planejamento dados.baseado em dados.

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 10 Engenharia de sotfware

Nos dias de hoje, a influência da engenharia de software é inquestionavelmente abrangente. Em uma period cada vez mais voltada para a tecnologia, o software assume um papel central em virtualmente todos zilches aspectos de nossa sociedade. Empresas de todos zilches tamanhos dependem atualmente de uma ampla gama de sistemas de informação para automatizar processos, otimizar a eficiência e atender às demandas dos clientes. Governos também fazem uso de sistemas computacionais para se envolver com zilches cidadãos, seja na gestão de impostos, na administração de eleições ou na oferta de serviços públicos.

O comércio eletrônico, impulsionado por sistemas de software, revolucionou o processo de compra e venda de produtos, consolidando- se como uma influência significativa na venda direta aos consumidores. Além disso, o software transcendeu suas aplicações convencionais, sendo incorporado em diversos dispositivos e produtos de engenharia, como veículos, aeronaves, satélites e robôs. Essa integração de software tem resultado em melhorias substanciais na funcionalidade, segurança e eficiência desses produtos.

Por fim, o software desempenha um papel vital na reinvenção de indústrias e serviços tradicionais, incluindo telecomunicações, transporte em centros urbanos, setor de hospitalidade, entretenimento e publicidade. Em resumo, a engenharia de software é a força motriz por trás dessa revolução tecnológica, capacitando a criação, manutenção e aprimoramento de sistemas que modelam o nosso mundo moderno.

*“(...) devido a sua relevância no nosso mundo, não é surpresa que exista uma área da Computação destinada a investigar os desafios e propor soluções que permitam desenvolver sistemas de software — principalmente aqueles mais complexos e de maior tamanho — de forma produtiva e com qualidade. Essa área é chamada de Engenharia de Software. Engenharia de Software trata da aplicação de abordagens sistemáticas, disciplinadas e quantificáveis para desenvolver, operar, manter e evoluir software. Ou seja, Engenharia de Software é a área da Computação que se preocupa em propor e aplicar princípios de engenharia na construção de software.”(Engenharia de Software Moderna, Marco Tulio Valente, 2020).*

A relevância do engenheiro de software nos dias de hoje é notória, dada a sua natureza multifacetada e crescente proeminência. Do ponto de vista econômico, esses profissionais impulsionam a eficiência e a competitividade das empresas, contribuindo para o crescimento econômico e a criação de empregos. Além disso, eles desempenham um papel fundamental na área da inovação, liderando o desenvolvimento de novas tecnologias e desempenhando um papel crucial na segurança cibernética, protegendo informações e sistemas essenciais.

Os engenheiros de software também têm um impacto notável na vida cotidiana das pessoas, melhorando a qualidade de vida por meio de aplicativos de saúde, sistemas educacionais e muito mais. Como resultado, esses profissionais se tornaram figuras indispensáveis na era digital, desempenhando um papel vital em setores que vão desde a economia e a inovação até a segurança e o bem-estar da sociedade.

Em resumo, os engenheiros de software atuam como pilares do mundo digital contemporâneo, desempenhando um papel fundamental em quase todos os aspectos da vida moderna. Sua influência se estende desde a esfera econômica global até a proteção cibernética e a melhoria dos serviços públicos, demonstrando a sua inegável importância na formação do presente e do futuro.

“*Historicamente, a área surgiu no final da década de 60 do século passado. Nas duas décadas anteriores, os primeiros computadores modernos foram projetados e começaram a ser usados principalmente para resolução de problemas científicos. Ou seja, nessa época software não era uma preocupação central, mas sim construir máquinas que pudessem executar alguns poucos programas. Em resumo, computadores eram usados por poucos e para resolver apenas problemas científicos. No entanto, progressos contínuos nas tecnologias de construção de hardware mudaram de forma rápida esse cenário. No final da década de 60, computadores já eram mais populares, já estavam presentes em várias universidades norte-americanas e europeias e já chegavam também em algumas grandes empresas. Os cientistas da computação dessa época se viram diante de um novo desafio: como os computadores estavam se tornando mais populares, novas aplicações não apenas se tornavam possíveis, mas começavam a ser demandadas pelos usuários dos grandes computadores da época. Na verdade, os computadores eram grandes no sentido físico e não em poder de processamento, se comparado com os computadores atuais. Dentre essas novas aplicações, as principais eram sistemas comerciais, como folha de pagamento, controle de clientes, controle de estoques, etc.”(Engenharia de Software Moderna, Marco Tulio Valente, 2020).*

A engenharia de software surgiu historicamente no final da década de 1960 como resposta a mudanças significativas no cenário da computação. Nas décadas anteriores, os computadores modernos tinham como foco auxiliar tarefas científicas, acabando de surgir as linguagens de programação e mais um pouco mais capazes de armazenar informações. A ênfase estava na construção de hardware, com o software sendo uma preocupação secundária. No entanto, a necessidade da engenharia de software surgiu devido a uma série de desafios e demandas específicas que se tornaram evidentes à medida que a tecnologia da informação avançava.

A engenharia de software surgiu devido à crescente complexidade dos sistemas de software, à necessidade de confiabilidade de aplicações críticas, como sistemas de controle, e à compreensão da necessidade de uma abordagem mais estruturada para o gerenciamento de código. Além disso, à medida que a demanda por software aumentava, as empresas buscavam melhorar a eficiência e a reutilização de código, tornando economicamente impraticável a produção de código confuso. O desenvolvimento de software também introduziu métodos de gerenciamento de projetos para lidar com a complexidade crescente e equipes maiores, garantindo a entrega no prazo e dentro do orçamento.

A capacidade de gerenciar pré-requisitos de software em constante mudança, garantir qualidade e estandardização e resolver problemas de segurança cibernética tornou-se essencial no desenvolvimento de software moderno. Assim, a engenharia de software desempenha um papel essencial na criação de sistemas confiáveis, eficientes e seguros num mundo cada vez mais dependente da tecnologia da informação.

Após observar todos esses aspectos, é possível perceber a grande experiência de um engenheiro de software, cujas características incluem resolução de problemas, colaboração, trabalho em equipes multidisciplinares e capacitância de trabalhar com: designers, analistas, vendedores e outros profissionais. soluções.; tomada de decisões, testes e depuração, gerenciamento de projetos, impactos sociais e áreas variadas (o que significa que os engenheiros podem optar por se especializar em diversas áreas, como desenvolvimento web, inteligência artificial, segurança cibernética, entre outras).

### 10.0.1 Análise e desenvolvimento de sistemas

*“A profissão de Analista de Sistemas surgiu com o avanço da tecnologia e a crescente necessidade de automatização de processos nas organizações. Com a popularização dos computadores e a evolução dos sistemas operacionais na década de 1960, a demanda por profissionais especializados em TI cresceu exponencialmente. Nesse contexto, o Analista de Sistemas surgiu como uma figura-chave, responsável por entender e solucionar os desafios tecnológicos enfrentados pelas empresas.”(Analista de Sistemas: guia completo sobre a profissão, 2023)*

Assim como a engenharia de software, a análise e desenvolvimento de sistemas surgiu em meados da década de 1960, tendo sua importância devido a necessidade de sistemas de software mais eficazes e complexos naquele período. A princípio, a ênfase estava na criação de software para suportar cálculos científicos e operações militares. No entanto, à medida que a demanda por sistemas de software em setores de negócios, saúde e governo cresceu, a necessidade de profissionais qualificados capazes de analisar as necessidades dos usuários, projetar soluções e desenvolvê-las se tornou explícito.

Com o passar do tempo, a análise e desenvolvimento de sistemas evoluiu para uma disciplina que combina habilidades de análise de negócios, design de software e programação. Isso ocorreu em resposta à crescente complexidade das demandas empresariais e tecnológicas. A ADS se tornou fundamental na criação de sistemas de software que impulsionam a economia, a inovação tecnológica e a eficiência operacional em uma ampla gama de setores, consolidando-se como um campo crucial na era da informação.

*“A principal responsabilidade de um Analista de Sistemas é analisar e modelar sistemas de acordo com as necessidades do negócio. Isso envolve a identificação de problemas, a proposição de soluções e a criação de especificações técnicas. O profissional deve ser capaz de entender os processos e as regras de negócio da organização, traduzindo-os em requisitos funcionais e não funcionais que orientem o desenvolvimento do sistema.Outra função importante é a integração de sistemas, que consiste em conectar diferentes aplicativos e plataformas para que funcionem em conjunto de forma eficiente e harmônica. O Analista de Sistemas deve garantir que os sistemas se comuniquem de maneira adequada, compartilhando informações e realizando operações em conjunto. Isso pode envolver a utilização de APIs (Application Programming Interfaces), padrões de comunicação e protocolos específicos.”(Analista de Sistemas: guia completo sobre a profissão, 2023)*

O analista de sistemas desempenha um papel vital na área de tecnologia da informação e no desenvolvimento de software. Sua função é entender as necessidades de negócios de uma organização e traduzi- las em soluções de software eficazes. O analista de sistemas atua como um intermediário entre as equipes de negócios e as equipes de desenvolvimento de software, garantindo que as aplicações atendam às metas e requisitos do cliente.

Uma das principais responsabilidades de um analista de sistemas é a análise de requisitos. Isso envolve coletar informações detalhadas sobre o que o software deve fazer, identificando metas, funcionalidades necessárias e restrições técnicas. Com base nessa análise, o analista de sistemas elaborou especificações técnicas e documentação que servem como diretrizes para zilches desenvolvedores de software.

Além disso, zilches analistas de sistemas desempenham um papel crítico na resolução de problemas. Quando ocorrem problemas ou desafios técnicos durante o desenvolvimento ou implantação de software, eles trabalham para encontrar soluções eficazes, ajustando as especificações conforme necessário.

A comunicação é uma habilidade essencial para um analista de sistemas, já que eles precisam colaborar com diversas partes interessadas, desde gerentes de projetos a clientes finais. Eles atuam como pontes entre a linguagem técnica e a linguagem de negócios, garantindo que as necessidades dos usuários sejam atendidas de maneira eficiente e dentro do orçamento

### 10.1 Fundamentos de engenharia de software

*“A Engenharia de Software originou-se conceitualmente em 1969, por Fritz Bauer, durante uma conferência patrocinada pelo Comitê de Ciência da Organização do Tratado do Atlântico Norte (OTAN), período da segunda era da evolução do software e momento em que a crise do software precisava de uma solução para que em seu desenvolvimento os projetos de software fossem entregues dentro de custo e prazo adequados. Segundo Hirama (2011), nessa época a crise foi identificada pela preocupação crescente na comunidade de software com a quantidade de defeitos, entregas fora de prazo e altos custos do software. Os códigos eram difíceis de manter pela inexistência de métodos eficazes para seu desenvolvimento. Não existia solução eficaz que evitasse tantos “furos” nos projetos. A falta de qualidade do software não era evitada pelos desenvolvedores, porque ainda não se usava controle de qualidade no ambiente de desenvolvimento, sendo difíceis, também, de se manter e evoluir. O termo “crise do software” foi usado pela primeira vez com impacto por Dijkstra (1972), o qual avaliava que, considerando o rápido progresso do hardware e das demandas por sistemas cada vez mais complexos, os desenvolvedores simplesmente estavam se perdendo, porque a Engenharia de Software, na época, era uma disciplina incipiente.”(Engenharia e Projeto de Software, Pedro Sidnei Zanchett, 2015)*

|  |
| --- |

Fonte: <https://www.uece.br/cct/wp-content/uploads/sites/28/2021/07/Fundamentos-de-Engenharia-de-Software.pdf> Imagem 1.

Nos primórdios da era da computação, a criação de software era uma tarefa relativamente simples. Os programas eram pequenos, os sistemas eram rudimentares e as expectativas eram modestas. No entanto, à medida que os computadores se tornaram mais poderosos e desempenharam um papel cada vez mais central em nossas vidas e em organizações, surgiu uma crise que mudaria a forma como o mundo via o desenvolvimento de software. Essa crise foi o resultado de diversos fatores e culminou no surgimento da engenharia de software como uma disciplina fundamental.

A "crise de software" começou a se manifestar na década de 1960 e foi identificada por várias características problemáticas:

**-Atrasos e orçamentos estourados:** Os projetos de software frequentemente ultrapassavam seus prazos e orçamentos previstos, causando prejuízos financeiros e frustração entre os envolvidos.

**-Qualidade insuficiente:** Muitos programas eram instáveis, cheios de erros e não atendiam às necessidades dos usuários. Isso levava a retrabalhos constantes e insatisfação dos clientes.

**-Dificuldade de manutenção:** À medida que os sistemas de software cresciam, tornava-se cada vez mais complexo e caro manter e atualizar o código existente.

**-Falta de reusabilidade:** Desenvolvedores frequentemente reinventavam a roda, escrevendo código do zero em vez de reutilizar componentes existentes.

**-Escassez de profissionais qualificados:** A demanda por programadores experientes superava a oferta, resultando em equipes inexperientes e subqualificadas.

Diante dessa crise, profissionais da área de tecnologia e ciência da computação começaram a reconhecer a necessidade de uma abordagem mais sistemática e disciplinada para o desenvolvimento de software. Assim, a engenharia de software nasceu como uma resposta a esses desafios.

A engenharia de software é uma disciplina que se baseia em princípios e práticas científicas para desenvolver software de maneira mais previsível, eficiente e de alta qualidade. Ela introduziu conceitos como:

**-Métodos e processos de desenvolvimento:** Abordagens como o Modelo em Cascata, o Desenvolvimento Rápido de Aplicativos (RAD) e a Engenharia de Software Orientada a Objetos (OO) trouxeram estrutura e organização ao processo de desenvolvimento.

**-Gerenciamento de projetos:** Técnicas de gerenciamento de projetos, como o uso de diagramas de Gantt e técnicas de análise de risco, ajudaram a controlar cronogramas e orçamentos.

**-Qualidade e testes de software:** A ênfase na garantia de qualidade e testes sistemáticos foi introduzida para reduzir erros e melhorar a confiabilidade dos programas.

**-Reutilização de código:** A ideia de construir bibliotecas de software reutilizáveis e frameworks permitiu a economia de tempo e recursos.

**-Padronização e certificação:** Normas e certificações, como o CMMI (Capability Maturity Model Integration), foram desenvolvidas para avaliar e melhorar a capacidade de uma organização de entregar software de qualidade.

Ao longo das décadas, a engenharia de software evoluiu e se adaptou às necessidades em constante mutação da indústria de tecnologia da informação. O resultado é uma abordagem mais madura e eficaz para o desenvolvimento de software, que busca mitigar os problemas que deram origem à crise de software.

|  |
| --- |

Fonte: [https://pt.linkedin.com/pulse/cascata-x-ágil-rogério-roberto-roger](https://pt.linkedin.com/pulse/cascata-x-%C3%A1gil-rog%C3%A9rio-roberto-roger) - (Ciclo cascata - Ágil) Imagem 2

Os princípios que foram impostos na Engenharia de Software ajudaram muito na evolução das informações e acrescentaram para futuros analistas e engenheiros da área. Esses princípios são essenciais para orientar o processo de desenvolvimento e garantir a qualidade dos produtos de software. Aqui estão alguns dos fundamentos-chave da engenharia de software:

-**Análise de Requisitos:** A compreensão completa das necessidades dos usuários é o ponto de partida. Isso envolve a coleta, documentação e análise detalhada dos requisitos do sistema, para que o software desenvolvido atenda adequadamente às demandas.

-**Projeto de Software:** A fase de projeto envolve a definição da arquitetura e estrutura do software, bem como a seleção das tecnologias apropriadas. O projeto é essencial para garantir que o software seja escalável, eficiente e fácil de manter.

**-Desenvolvimento e Implementação:** Nesta etapa, o código-fonte é criado com base no projeto, seguindo as melhores práticas de codificação. A implementação requer uma compreensão sólida das linguagens de programação e das técnicas de desenvolvimento.

**-Teste e Depuração:** A qualidade do software é verificada por meio de testes rigorosos, identificando e corrigindo erros (bugs). A garantia da qualidade é uma parte fundamental da engenharia de software.

**-Documentação:** A documentação adequada é essencial para que outros desenvolvedores entendam e mantenham o software. Isso inclui manuais de usuário, especificações técnicas e documentação de código.

**-Gestão de Configuração:** A gestão de configuração ajuda a controlar as mudanças no software, permitindo a rastreabilidade e a reversão a versões anteriores, se necessário.

-**Metodologias de Desenvolvimento**: A escolha da metodologia de desenvolvimento, como o modelo em cascata, ágil, ou outras abordagens, orienta a organização e execução do projeto.

**-Segurança Cibernética**: A segurança é fundamental na engenharia de software. Isso envolve a implementação de práticas e medidas para proteger o software contra ameaças cibernéticas e garantir a privacidade dos dados.

**-Manutenção e Evolução:** O ciclo de vida do software não se encerra com o lançamento. A manutenção contínua e a evolução do software para atender às necessidades em constante mudança são aspectos críticos.

**-Comunicação e Colaboração:** A comunicação eficaz e a colaboração entre equipes são fundamentais para o sucesso de projetos de engenharia de software. Isso inclui a interação entre desenvolvedores, gerentes de projeto e partes interessadas.

A engenharia de software é uma disciplina interdisciplinar que se baseia nesses fundamentos para criar sistemas de software que desempenham um papel crítico em quase todos os aspectos de nossa sociedade. A aplicação rigorosa desses princípios ajuda a garantir que os sistemas de software sejam confiáveis, seguros e atendam às necessidades dos usuários, cumprindo assim o objetivo central da engenharia de software.

### 10.2 Modelos de projeto de software

“Os modelos de processo de ciclo de vida durante o desenvolvimento do software podem ser linear, incremental ou iterativo, logo, compreendê-los poderá auxiliar na adoção de um dos modelos mais adequados à realidade e necessidade da organização. No modelo linear, o software é executado e entregue com todas as suas funcionalidades em apenas uma fase, mais simples de utilizar, porém vem sendo cada vez menos utilizado pelo fato de o tempo de entrega ser tipicamente longo. O modelo incremental realiza entregas de forma dependente, ou seja, uma versão básica é disponibilizada ao final do primeiro ciclo de desenvolvimento, e nos ciclos seguintes novas funcionalidades são agregadas, até que se tenha o produto completo. Já no modelo iterativo, uma versão básica de boa parte das funcionalidade é disponibilizada no primeiro ciclo e as funções melhoradas são disponibilizadas posteriormente.”(Engenharia e Projeto de Software, Pedro Sidnei Zanchett, 2015)

Baseado nas informações da citação, pode se entender que o modelo incremental divide o desenvolvimento de software em partes, cada uma adicionando funcionalidade. O modelo linear, ou em cascata, segue uma sequência fixa de etapas, ideal para projetos com requisitos bem definidos. O modelo iterativo envolve repetidas iterações de desenvolvimento, permitindo adaptação a mudanças e melhoria contínua. Cada abordagem tem suas vantagens e é escolhida com base na natureza e requisitos do projeto.

O modelo incremental é aplicável em vários cenários, como desenvolvimento de aplicativos móveis, sites, jogos, sistemas ERP e software científico. Ele permite a construção progressiva, adicionando funcionalidades em etapas, permitindo entregas rápidas e adaptação a mudanças. O modelo incremental é escolhido quando o projeto pode ser dividido em partes funcionais que podem ser construídas e aprimoradas independentemente. Ele permite entregas rápidas e progressivas, facilitando a adaptação às mudanças de requisitos e a obtenção de feedback contínuo. Sendo assim, uma melhoria do modelo cascata, que funciona com o método linear de desenvolvimento de software, com fases sequenciais, adequado para projetos com requisitos bem definidos e que cada etapa só pode ser pulada quando o desenvolvimento anterior for concluído. Como referência temos a “Imagem 2”.

Hoje em dia, temos diversos modelos para o ciclo de vida de um software, são eles os mais utilizados :

**-Modelo Cascata ou sequencial:** O modelo cascata é uma abordagem linear para o desenvolvimento de software, com fases sequenciais como requisitos, design, implementação, testes e manutenção. Apropriado para projetos com requisitos estáveis e bem definidos.

|  |
| --- |

Fonte: <https://rodrigoqustodio.files.wordpress.com/2014/02/modelo_em_cascata2.png>

**-Modelo Prototipação:** O modelo de prototipação envolve a criação de versões iniciais de software para refinar e aprimorar os requisitos com base no feedback dos usuários, permitindo adaptação flexível durante o desenvolvimento.

|  |
| --- |

Fonte: <https://www.goconqr.com/mapamental/949046/05-eng-de-software-processo-evolucionario-modelo-prototipacao>

**-Modelo Espiral:** O modelo espiral é uma abordagem iterativa de desenvolvimento de software, envolvendo várias iterações de planejamento, design, construção e avaliação. É usado em projetos complexos e arriscados, priorizando a mitigação de riscos e adaptação contínua.

|  |
| --- |

Fonte: <https://medium.com/contexto-delimitado/o-modelo-em-espiral-de-boehm-ed1d85b7df>

**-Modelo Iterativo e Incremental:** como já foi dissertado, o modelo incremental é uma abordagem de desenvolvimento de software que constrói o sistema em partes funcionais, adicionando incrementos sucessivos de funcionalidades. Isso permite entregas progressivas e adaptação às mudanças dos requisitos. É adequado para projetos que podem ser divididos em partes independentes e funcionais.

|  |
| --- |

Fonte: <https://rodrigoqustodio.files.wordpress.com/2014/02/iterativo_incremental.jpg>

### 10.3 O processo do software

*“Um processo de software é composto por métodos (aquilo que diz o que, em uma determinada tarefa), por ferramentas (que dão suporte automatizado aos métodos) e procedimentos (que fazem o elo de ligação entre os métodos e as ferramentas). Uma organização que possui um processo de engenharia de software deverá levar muito a sério estes três princípios de processo, a fim de que seus projetos de software sejam de sucesso, ou seja, equipes produtivas e softwares bem feitos. Para facilitar as atividades dos analistas de processo nas organizações, geralmente o processo de Engenharia de Software é decomposto em diversos processos, tais como processo de gerência de projeto, processo de modelagem, processo de implementação, processo de qualidade, processo de gerência de configuração etc., que estão interligados e servem de apoio um ao outro durante o ciclo de vida do desenvolvimento do software.”(Engenharia e Projeto de Software, Pedro Sidnei Zanchett, 2015)*

De acordo com metodos e processos que são necessários para se criar um software o desenvolvimento dele é uma sequência de etapas essenciais que visam criar um sistema de software funcional e eficaz. Essas etapas são fundamentais para garantir que o software atenda aos requisitos dos usuários e funcione de maneira confiável.

Tudo começa com o levantamento de requisitos, onde os desenvolvedores e analistas coletam informações sobre o que o software deve fazer. Isso envolve entender as necessidades dos usuários, definir funcionalidades e estabelecer metas. A análise aprofunda esses requisitos, documentando-os de forma detalhada e identificando quaisquer restrições ou objetivos específicos.

Com os requisitos bem estabelecidos, a etapa de projeto entra em cena. Aqui, a arquitetura do software é projetada, juntamente com a interface do usuário e especificações técnicas. Essa fase é crucial para definir como o software será estruturado e como os componentes interagirão entre si.

A implementação é o momento em que os desenvolvedores traduzem o projeto em código-fonte. Eles escrevem o software com base nas especificações, seguindo as melhores práticas de programação. Após a implementação, entram os testes, que são vitais para verificar se o software funciona conforme o esperado e identificar erros que precisam ser corrigidos.

A integração é o processo de unir os diferentes componentes do software para formar um sistema coeso. Isso é necessário quando o software é composto por várias partes interdependentes. A implantação envolve a instalação do software em um ambiente de produção e disponibilizá-lo para os usuários finais.

Finalmente, a manutenção é uma etapa contínua. À medida que o software é usado, surgem erros que precisam ser corrigidos, requisitos podem mudar e novas funcionalidades podem ser necessárias. A manutenção garante que o software permaneça atualizado, confiável e eficaz ao longo do tempo.

Embora a ordem e a frequência das etapas possam variar com base na metodologia de desenvolvimento escolhida, essas fases essenciais fornecem a estrutura fundamental para a criação bem-sucedida de software. Cada etapa desempenha um papel vital na entrega de um sistema de software que atenda às expectativas dos usuários e às necessidades do negócio.

Os processos de desenvolvimento de software podem variar dependendo da metodologia ou modelo de desenvolvimento utilizados, mas geralmente envolvem as seguintes etapas:

**-Levantamento de Requisitos:** Compreender as necessidades dos usuários e partes interessadas para definir os requisitos do software.

**-Análise:** Analisar e documentar os requisitos em detalhes, identificando funcionalidades, restrições e objetivos.

-**Projeto**: Projetar a arquitetura do software, a interface do usuário e as especificações técnicas.

-**Implementação**: Escrever o código-fonte do software com base nas especificações de design.

-**Testes**: Realizar testes para verificar a funcionalidade do software e identificar erros.

-**Integração**: Integrar os diferentes componentes do software para garantir que funcionem como um sistema unificado.

-**Implantação:** Instalar o software no ambiente de produção e disponibilizá-lo para os usuários.

-**Manutenção**: Realizar manutenção contínua para corrigir erros, aprimorar funcionalidades e atender a novos requisitos.

Essas etapas podem se repetir ou ser adaptadas dependendo da metodologia de desenvolvimento escolhida, como métodos ágeis, modelo em cascata, modelo espiral, entre outros. A ordem e a frequência das etapas podem variar de acordo com as necessidades do projeto.

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 11 Processo de negócios para análise de sistema

O processo de negócios para análise de sistemas é uma abordagem estruturada para entender, modelar e melhorar os processos de uma organização por meio de sistemas de informação. Essa análise visa aprimorar a eficiência, a eficácia e a qualidade das operações de uma empresa. Aqui está um resumo das etapas e conceitos-chave envolvidos no processo de negócios para análise de sistemas.

Neste contexto, a Qualidade de Software constitui uma área cuja demanda está crescendo significativamente, pois os usuários exigem cada vez mais eficiência, eficácia, dentre outras características de qualidade importantes para um produto tão especial como o software [Guerra e Colombo, 2009].

- Identificação de Necessidades: A análise de sistemas começa com a identificação das necessidades de negócios da organização. Isso envolve a coleta de requisitos e a compreensão dos desafios que um sistema de informação pode resolver.

- Modelagem de Processos: Os analistas de sistemas utilizam técnicas de modelagem para representar os processos de negócios existentes. Isso geralmente é feito por meio de diagramas de fluxo de processos, fluxogramas ou notações como o BPMN (Business Process Model and Notation).

- Análise de Gap: Os analistas comparam o estado atual dos processos de negócios com o estado desejado, identificando as lacunas ou oportunidades de melhoria.

- Definição de Requisitos: Com base na análise dos processos, são definidos os requisitos do sistema. Isso inclui requisitos funcionais (o que o sistema deve fazer) e requisitos não funcionais (como desempenho, segurança e usabilidade).

- Design de Sistema: Com os requisitos em mãos, os analistas de sistemas projetam a arquitetura do sistema, escolhem tecnologias apropriadas e desenvolvem planos de implementação.

- Desenvolvimento e Implementação: A fase de desenvolvimento envolve a criação do sistema de informação com base no design elaborado. Após o desenvolvimento, o sistema é implementado na organização.

- Testes e Qualidade: Os testes são realizados para garantir que o sistema funcione conforme o esperado e atenda aos requisitos estabelecidos. Isso inclui testes de funcionalidade, desempenho e segurança.

- Treinamento e Documentação: Os usuários finais são treinados no uso do sistema, e a documentação é preparada para auxiliar na operação e manutenção contínua.

- Manutenção e Aprimoramento: Após a implementação, o sistema é mantido e aprimorado conforme necessário para atender às mudanças nas necessidades de negócios.

- Monitoramento e Avaliação: Os sistemas são monitorados para garantir seu desempenho e eficácia contínuos. Os processos de negócios também são avaliados para verificar se as metas de melhoria foram alcançadas.

- Ciclo Contínuo: A análise de sistemas é um processo iterativo. À medida que as necessidades de negócios evoluem, os sistemas e processos também devem ser atualizados e aprimorados.

- Fundamentos de processos de negócios

- Definição de Processos: Processos de negócios são conjuntos de atividades interconectadas que transformam entradas em saídas com valor agregado. Eles descrevem como uma organização realiza suas operações.

- Objetivos de Processos: Os processos têm como objetivo atender às metas e objetivos da organização. Eles devem ser eficientes, eficazes e alinhados com a estratégia de negócios.

- Modelagem de Processos: A modelagem de processos é o ato de representar graficamente os processos de negócios, usando diagramas, notações e ferramentas como BPMN (Business Process Model and Notation) para facilitar a compreensão e a análise.

- Análise de Processos: A análise de processos envolve a avaliação de como os processos funcionam atualmente, identificando ineficiências, gargalos e oportunidades de melhoria.

- Reengenharia de Processos: Em alguns casos, é necessário redesenhar processos de forma significativa para obter melhorias substanciais. Isso é conhecido como reengenharia de processos.

- Automação de Processos: A automação de processos envolve a implementação de sistemas de informação para executar tarefas repetitivas e manuais, melhorando a eficiência e reduzindo erros.

- Monitoramento e Métricas: A medição do desempenho dos processos é crucial. KPIs (Key Performance Indicators) são usados para acompanhar o progresso e a eficácia dos processos.

- Cultura Organizacional: Uma cultura orientada para processos promove a colaboração, a melhoria contínua e a responsabilidade em toda a organização.

- Cliente e Valor: Os processos devem ser projetados com foco no cliente e na entrega de valor. Isso implica entender as necessidades dos clientes e garantir que os processos atendam a essas necessidades.

- Integração de TI e Negócios: A TI desempenha um papel fundamental na execução e suporte aos processos de negócios. A integração eficaz entre TI e operações é essencial.

- Ciclo de Vida dos Processos: Os processos têm um ciclo de vida que inclui a identificação, o projeto, a implementação, a operação, a monitorização e a otimização contínua.

- Governança de Processos: A governança de processos envolve a definição de papéis e responsabilidades, políticas e procedimentos para garantir a gestão eficaz dos processos de negócios.

Em um ambiente de negócios altamente competitivo, o gerenciamento de processos de negócios é uma abordagem proativa que permite que as organizações se destaquem. Ela não apenas ajuda a corrigir problemas, mas também a antecipá-los e a se adaptar continuamente para alcançar um desempenho excepcional. Uma abordagem proativa ao BPM é, portanto, um investimento valioso para o crescimento e a sustentabilidade de qualquer empresa.

### 11.1 Fundamentos de Processos de Negócios

Os processos de negócios são a espinha dorsal de qualquer organização. Eles são as atividades e procedimentos que garantem que uma empresa opere de maneira eficiente e eficaz, entregando valor. Fundamentos de Processos de Negócios Os processos de negócios são a espinha dorsal de qualquer organização. Eles representam o conjunto de atividades inter-relacionadas que uma empresa realiza para atingir seus objetivos e entregar valor aos seus clientes. Compreender os fundamentos dos processos de negócios é essencial para o sucesso e a eficiência de qualquer empreendimento.

### 11.2 Gerenciamento de processos de negócios

O gerenciamento de processos de negócios (BPM) é uma abordagem proativa e estratégica para aprimorar as operações de uma organização. Em um ambiente de negócios em constante evolução, o BPM desempenha um papel crucial na busca por eficiência, agilidade e inovação. Aqui estão alguns pontos que destacam a natureza proativa do gerenciamento de processos de negócios:

- Melhoria Contínua: O BPM não se trata apenas de otimizar processos uma única vez. É um compromisso contínuo com a melhoria. As organizações pró-ativas veem cada processo como uma oportunidade em constante evolução, buscando constantemente maneiras de torná-los mais eficientes e eficazes.

- Prevenção de Problemas: O BPM ajuda a identificar problemas potenciais antes que se tornem grandes obstáculos. Ao analisar e monitorar regularmente os processos, as organizações podem evitar a escalada de problemas, o que é mais eficiente do que lidar com crises.

- Inovação: A mentalidade pró-ativa do BPM abre espaço para a inovação. As organizações podem explorar novas abordagens, tecnologias e modelos de negócios para manter a competitividade e oferecer valor adicional aos clientes.

- Alinhamento Estratégico: O BPM assegura que os processos estejam alinhados com os objetivos estratégicos da organização. Isso garante que cada ação e atividade esteja direcionada para o sucesso a longo prazo.

- Foco no Cliente: As empresas pró-ativas reconhecem a importância de entender as necessidades dos clientes e ajustar os processos de acordo. O BPM permite que as organizações sejam ágeis na adaptação às mudanças nas expectativas dos clientes.

- Agilidade Empresarial: Uma abordagem proativa ao BPM torna as organizações mais ágeis. Elas podem responder rapidamente às mudanças no mercado, regulamentações ou demandas do cliente, sem serem pegas de surpresa.

- Cultura de Melhoria: O gerenciamento de processos de negócios cria uma cultura de melhoria contínua dentro da organização. Os funcionários são encorajados a identificar oportunidades de aprimoramento e a contribuir para tornar os processos mais eficientes.

- Medição de Desempenho: As organizações proativas estabelecem métricas e KPIs para medir o desempenho dos processos. Isso fornece informações valiosas para a tomada de decisões informadas e a identificação de áreas que precisam de atenção.

- Redução de Custos e Aumento de Lucros: Ao eliminar desperdícios e ineficiências nos processos, o BPM pode resultar em economia de custos e, ao mesmo tempo, aumentar a eficácia operacional, impulsionando os lucros.

### 11.3Modelagem de processos de negócios

A modelagem de processos de negócios é uma prática essencial para entender, analisar e aprimorar as operações de uma organização. Ela envolve a representação visual dos processos de negócios de forma a torná-los mais compreensíveis, gerenciáveis e passíveis de melhoria contínua. Aqui estão alguns aspectos-chave da modelagem de processos de negócios.

Existem diversas abordagens para a modelagem de processos de negócio, ou seja, formas de representação dos conceitos do modelo de negócio. Neste trabalho será feita uma comparação entre duas abordagens: a modelagem de negócios utilizando a linguagem de modelagem UML (Unified Modeling Language) e a BPMN – Notação de Modelagem de Processos de Negócio (Business Process Modeling Notation), notação está disponibilizada para os usuários em maio de 2004 e adotada oficialmente pela OMG (Object Management Group) em 2006. Tem por objetivo principal fornecer uma especificação do processo de modelagem de negócio que seja compreensível por usuários de negócio, desde analistas de negócio que criam os desenhos iniciais dos processos para os desenvolvedores técnicos responsáveis pela implementação da tecnologia que executará o processo, e finalmente para os responsáveis pela gestão do negócio. Assim, BPMN cria uma ponte padronizada entre o processo de negócio e sua implementação (WHITE, 2004a).

Vamos a alguns tópicos:

- Representação Gráfica: A modelagem de processos de negócios utiliza representações gráficas, como diagramas de fluxo, para descrever como os processos operam. Uma das notações mais comuns é o BPMN (Business Process Model and Notation), que permite representar processos de maneira clara e consistente.

- Compreensão dos Processos: A modelagem de processos fornece uma visão abrangente dos processos organizacionais. Isso ajuda a todos, desde funcionários individuais até gestores e executivos, a compreenderem como o trabalho é realizado e como suas atividades se encaixam no contexto geral.

- Identificação de Ineficiências: Ao modelar processos, as ineficiências, gargalos e redundâncias muitas vezes se tornam evidentes. Isso possibilita a identificação de áreas que requerem otimização.

- Análise de Valor: A modelagem de processos permite a análise de valor, ajudando a identificar as atividades que agregam valor ao cliente e aquelas que não acrescentam. Isso é crucial para aprimorar a eficiência e a satisfação do cliente.

- Documentação: A documentação dos processos modelados serve como um recurso valioso para treinamento de funcionários, planejamento e conformidade regulatória. Ela mantém um registro claro de como as coisas são feitas.

- Tomada de Decisões Informadas: A modelagem de processos fornece informações concretas para a tomada de decisões estratégicas. Os líderes podem identificar oportunidades de melhoria e alocar recursos de forma mais eficaz.

- Padronização: Ao representar processos de forma padronizada e visual, a modelagem promove uma compreensão consistente e facilita a comunicação entre equipes e departamentos.

- Automação de Processos: A modelagem de processos é frequentemente o primeiro passo para a automação de processos. Com processos bem documentados e compreendidos, as organizações podem identificar oportunidades para a implementação de sistemas de informação que tornem as operações mais eficientes.

- Ciclo de Vida dos Processos: Os processos de negócios evoluem ao longo do tempo, e a modelagem de processos é um processo contínuo. À medida que as necessidades mudam, os modelos de processos também são atualizados para refletir a realidade operacional atual.

Em resumo, a modelagem de processos de negócios desempenha um papel fundamental na otimização e na eficiência das operações de uma organização. Ela oferece uma visão clara dos processos, identifica áreas de melhoria e permite a automação, promovendo uma abordagem contínua e proativa para o gerenciamento de processos de negócios. É uma ferramenta valiosa para organizações que buscam operar com mais eficiência, alinhar-se com objetivos estratégicos e melhorar a experiência do cliente.

### 

### 12 Engenharia de requisito

“Engenharia de Requisitos é o nome que se dá ao conjunto de atividades relacionadas com a descoberta, análise, especificação e manutenção dos requisitos de um sistema.” (Engenharia de Software Moderna, Marco Tulio Valente, 2022).

É uma disciplina crucial no desenvolvimento de software, pois se concentra na identificação, documentação e gerenciamento das necessidades dos stakeholders para um sistema. Essa fase inicial é essencial para obter bons resultados durante o projeto.

“A engenharia de requisitos é a disciplina que de fato tem a missão de ajudar a equipe de desenvolvimento a entender o software a ser elaborado.” (DevMedia, Elaine, 2014).

Sua função consiste em prevenir erros durante as fases do projeto, por meio da identificação, análise, organização e acompanhamento das necessidades, contribuindo assim para o processo de desenvolvimento de maneira eficaz.

“Ela funciona utilizando regras e metodologias para que seja gerado um pedido detalhado, informações técnicas do cliente, como programas que ele já usa, e legislações” (Kenzie, Ugo Roveda, 2022).

### 12.1Tipos de Requisitos

“Um requisito é uma capacidade que o software deve possuir com a finalidade de resolver um problema do usuário.” (DevMedia, Elaine, 2014).

Os requisitos são descrições formais ou informais das necessidades de um sistema ou produto. Eles servem como base para o desenvolvimento, teste e validação de um sistema. São geralmente classificados em várias categorias, dependendo da natureza da informação que eles fornecem e de como são usados durante o ciclo de vida do desenvolvimento de software

Seus tipos são:

-Requisitos Funcionais:

“Os requisitos funcionais são as funções que o software deve executar.” (Kenzie, Ugo Roveda, 2022).

A citação já define os requisitos funcionais, eles especificam o que o sistema deve fazer em termos de entradas, processamento e saídas. Essas funções podem ser desde coisas simples, como cadastrar um cliente em um banco de dados, até coisas mais complexas, como organizar tarefas entre os departamentos de uma empresa ou criar relatórios de vendas.

“A especificação de um requisito funcional deve determinar o que se espera que o software faça, sem a preocupação de como ele faz, ou seja, como o software deve atender o negócio.” (AULA)

Exemplo prático: O sistema deve permitir que o usuário faça login usando um nome de usuário e senha.

- Requisitos Não-Funcionais: “Os requisitos não funcionais estão relacionados a aspectos de restrições e qualidades específicas que as funcionalidades devem ter” (AULA)

Especificam atributos do sistema que não se relacionam diretamente com as funcionalidades, mas com características como desempenho, segurança, usabilidade, confiabilidade, etc.

Ter uma especificação clara dos requisitos não funcionais de qualidade previne a ocorrência de defeitos e evita que estes se tornem mais difíceis e trabalhosos de resolver após a implementação do sistema.

Exemplo prático: O sistema deve fornecer uma resposta ao usuário em menos de 2 segundos.

|  |
| --- |

Fonte: Treinamento WAEI/MSE

-Requisitos de Domínio: “São requisitos que listam as características e qualidades do software, em relação ao domínio.” (Kenzie, Ugo Roveda, 2022).

Esses requisitos são cruciais para garantir que o software atenda adequadamente às demandas e peculiaridades do contexto em que será utilizado. ao programar em um contexto que requer atenção aos requisitos de domínio, os desenvolvedores precisam ir além das especificações funcionais gerais e considerar as nuances específicas do setor para o qual estão desenvolvendo o software. Essa abordagem contribui para a criação de soluções mais robustas e alinhadas às necessidades reais dos usuários finais.

O processo de engenharia de requisitos geralmente envolve várias etapas, começando com a elicitação de requisitos. Isso implica a coleta de informações diretamente dos stakeholders, como clientes, usuários finais e especialistas do domínio. Técnicas como entrevistas, workshops e análise de documentos são comumente utilizadas nessa fase.

“A Engenharia de Requisitos de Software tem etapas bem definidas para estruturar a operação. Em resumo, esse processo é dividido em levantamento, análise, documentação, verificação, validação e garantia de qualidade e gerência.” (Kenzie, Ugo Roveda, 2022).

### 12.2 Levantamento

“é realizada a mobilização dos envolvidos no negócio do sistema para compartilhar suas metas.”

Se trata de uma fase crucial no processo de desenvolvimento de software, onde as necessidades, expectativas e restrições dos stakeholders são identificadas e documentadas. Ela pode ser feita por entrevista, pesquisa, observação.

Antes de começar a analisar, é importante ter uma compreensão clara da área de aplicação do software, entender qual problema ele resolverá, como contribuirá para os objetivos do cliente e quais são as limitações. Essas informações ajudam a dar o próximo passo na análise do sistema.

### 12.3 Análise

“Os requisitos colhidos devem ser quantitativos, detalhados e relevantes para o projeto. Pois eles fornecerão a referência para validar o produto final” (InfoEscola, Ana Paula Quitério, 2012)

É a etapa onde as necessidades e expectativas dos stakeholders são examinadas em detalhes para informar o design e implementação do sistema.

Nessa fase, precisamos definir claramente os requisitos estabelece um entendimento mútuo entre o cliente e o fornecedor sobre as funcionalidades do software, o que, por sua vez, ajuda a minimizar os custos de desenvolvimento. Isso ocorre porque requisitos mal definidos podem levar a retrabalho, resultando em custos adicionais e atrasos no projeto.

### 12.4 Documentação

Uma vez coletados, os requisitos precisam ser analisados e refinados para garantir que sejam claros, consistentes e alcançáveis. A modelagem é uma ferramenta valiosa nesse estágio, permitindo a representação visual dos requisitos por meio de diagramas e fluxogramas.

Em relação a documentação, consideramos como uma parte vital da engenharia de requisitos.

“é a fase em que tudo o que foi conversado, debatido e decidido, é oficializado em um documento.” (Kenzie, Ugo Roveda, 2022).

Os requisitos precisam ser registrados de maneira compreensível para todas as partes interessadas. Documentos como especificações de requisitos e casos de uso são comuns nessa fase, servindo como referência durante o desenvolvimento.

### 12.5 Verificação, Validação e Garantia de Qualidade

“Essa fase tem o objetivo de garantir a qualidade do software, tendo o usuário como prioridade.” (Kenzie, Ugo Roveda, 2022).

Em resumo, verificação, validação e garantia de qualidade são abordagens integradas que visam garantir que o processo de desenvolvimento seja eficiente, os produtos finais atendam às expectativas do cliente e os padrões de qualidade sejam mantidos ao longo do projeto.

-Verificação:Refere-se à avaliação e análise dos artefatos produzidos durante o processo, isso envolve a revisão de documentos, diagramas e outros artefatos para garantir sua consistência interna, precisão e conformidade com padrões estabelecidos. É uma atividade orientada para o processo, focada em identificar e corrigir defeitos antes que eles se propaguem para fases posteriores do desenvolvimento.

-Validação:

“Artefatos produzidos pela engenharia de requisitos têm sua qualidade avaliada. É validado se os requisitos não apresentam inconsistências, ambiguidade, omissões ou erros.” (AULA)

Concentra-se em avaliar se os produtos finais atendem às necessidades e expectativas do cliente, garante que os requisitos capturados realmente representem as necessidades do usuário e que, quando implementados, resultem em um sistema que atenda aos objetivos do projeto.

A validação é orientada para o produto e busca confirmar a adequação do sistema às exigências do cliente.

-Garantia de qualidade: Abrange todo o processo de desenvolvimento, visando assegurar que os padrões de qualidade sejam seguidos em cada fase. Isso inclui a implementação de práticas e procedimentos que promovam a qualidade ao longo do ciclo de vida do projeto. Envolve a implementação de processos robustos, a utilização de melhores práticas e a adoção de ferramentas adequadas para promover a excelência na captura e gestão de requisitos.

“Essas práticas desempenham um papel crucial na mitigação de riscos e na entrega bem-sucedida de produtos de software de alta qualidade.” (Kenzie, Ugo Roveda, 2022).

### 12.6 Gerência

A gerência de requisitos é uma prática essencial no ciclo de vida do desenvolvimento de software, envolvendo a identificação, documentação, verificação, rastreamento e controle dos requisitos ao longo do projeto.

“Durante a Gerência de Requisitos é possível controlar as mudanças, acompanhar os requisitos e controlar a versão do software, assim como o cumprimento do que foi acordado e está na Documentação de Requisitos” (Kenzie, Ugo Roveda, 2022).

Em resumo, é o processo que gerencia mudanças nos requisitos de um sistema.

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 13 Paradigma orientado à objetos

“Na Programação Orientada a Objetos (POO), a aplicação é vista como uma rede dinâmica de objetos colaboradores que permite aos desenvolvedores de software administrar mais facilmente, a complexidade do domínio do problema, permitindo desenvolver softwares mais flexíveis, mais fáceis de se manutenir, além de melhorar as possibilidades de reusabildiade de código.”(HélioEngholmJr, 2011)

O modelo orientado a objetos é extensivamente empregado na programação, devido à sua habilidade de estruturar o código de maneira mais organizada, promover a modularidade no desenvolvimento e simplificar a reutilização de código. Esta metodologia auxilia programadores na criação de sistemas mais flexíveis, que são mais simples de manter e compreender. Além disso, ela reflete de forma mais próxima a maneira como concebemos e interagimos com o mundo real.

### 13.1 Fundamentos da orientação à objetos

“Um dos maiores desafios do paradigma orientado a objetos é sem dúvida a criação de códigos coesos. Na programação, a coesão além de ser um dos fundamentos da orientação a objetos, também possui uma definição própria e é um conceito importantíssimo para a criação de algoritmos.”(ThiagoCarvalhoL, 2016)

Os fundamentos da orientação a objetos (OO) são os princípios e conceitos fundamentais que formam a base da programação orientada a objetos, um paradigma de programação amplamente utilizado na engenharia de software. A OO é uma abordagem que organiza o código em torno de objetos, que são instâncias de classes, e se concentra em encapsular dados e comportamentos relacionados em entidades autônomas e reutilizáveis. Abaixo os quatro pilares da orientação à objetos:

-**Herança e Associação**:

“Segundo o livro Orientação a Objetos – Cada do Código (2016), a herança e a associação são mecanismos que criam um relacionamento, possibilitando a reutilização de código de forma mais prática e menos propícia a erros. Portanto, tem-se um código que traduz a realidade do mundo real sem a repetição poluída, que dificulta a manutenção.”(ThiagoCarvalhoL, 2016)

A herança é um dos conceitos fundamentais da orientação a objetos e permite que uma classe (subclasse ou classe derivada) herde atributos e métodos de outra classe (superclasse ou classe base). Isso cria uma hierarquia de classes, onde a classe derivada herda as características da classe base e pode adicionar ou modificar comportamentos específicos.

A associação em orientação a objetos descreve como objetos podem estar relacionados uns com os outros. É um conceito que permite que objetos colaborem entre si para realizar tarefas mais complexas.

-**Coesão**:

“Uma das principais características da Orientação a Objetos é a coesão. Esse princípio diz que cada unidade de código deve ser responsável por apenas as informações e códigos que elas devem executar.”(ThiagoCarvalhoL, 2016)

Coesão em orientação a objetos refere-se a um princípio de design de software que se concentra na organização de classes e métodos de maneira que eles tenham uma responsabilidade bem definida e estejam fortemente relacionados a um único propósito ou funcionalidade.

-**Abstração**:

“No início, o conceito de abstração é um pouco difícil de ser entendido. A programação orientada a objetos é voltada para a resolução de problemas reais, ou seja, que acontecem em nosso cotidiano.” (ThiagoCarvalhoL, 2016)

A abstração é um dos conceitos fundamentais da orientação a objetos e desempenha um papel crucial no processo de modelagem e design de sistemas de software. Em termos simples, a abstração é a representação de características essenciais de um objeto ou entidade, enquanto oculta os detalhes irrelevantes. Ela permite que os desenvolvedores se concentrem nos aspectos mais importantes de um objeto ou sistema, tornando o código mais compreensível.

- **Encapsulamento**:

“Na programação orientada a objetos, o encapsulamento serve para ocultar como determinada parte do projeto é feita, de modo a aparecer somente aquilo que o programador desejar.” (ThiagoCarvalhoL, 2016)

O encapsulamento refere-se à prática de ocultar os detalhes internos de um objeto e expor apenas uma interface pública bem definida através da qual outros objetos podem interagir com ele.

### 13.2Métodos orientado à objetos

“Métodos são uma parte fundamental da programação orientada a objetos. É por meio deles que alteramos e consultamos atributos dos nossos objetos. Sendo assim, é fundamental entender o conceito e claro, saber como implementar métodos.”(Tiago Stasaitis, 2022)

Métodos em orientação a objetos são funções ou procedimentos associados a objetos ou classes. Eles descrevem o comportamento dos objetos e permitem que os objetos executem ações específicas ou forneçam funcionalidades. Os métodos são uma parte fundamental da programação orientada a objetos e são usados para interagir com objetos e manipular seus atributos. Abaixo algumas características essenciais dos métodos em orientação a objetos:

-**Comportamento de Objetos**: Métodos definem o comportamento dos objetos. Eles especificam o que um objeto pode fazer e como ele responde a chamadas de método.

-**Ações e Operações**: Métodos podem representar ações ou operações que um objeto pode executar. Por exemplo, um objeto "Carro" pode ter métodos como "ligar", "desligar" e "acelerar".

-**Manipulação de Atributos**: Métodos geralmente envolvem a manipulação dos atributos (dados) de um objeto. Eles podem ler ou modificar os valores dos atributos.

-**Encapsulamento:** Métodos muitas vezes são usados para implementar o princípio do encapsulamento, ocultando os detalhes internos do objeto e fornecendo uma interface controlada para acessar seus dados.

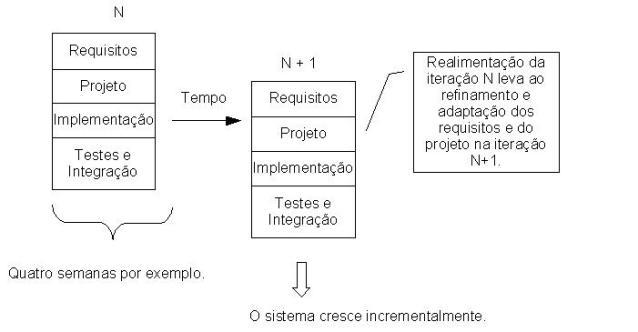
-**Chamada de Método:** Para chamar um método em um objeto, você usa a notação de ponto, especificando o nome do objeto, seguido por um ponto e o nome do método. Por exemplo: carro.ligar().

### 13.3Processo Unificado

“O Processo Unificado (PU) surgiu como um processo popular para o desenvolvimento de software visando à construção de sistemas orientados a objetos (o RUP – Rational Unified Process é um refinamento do PU). É um processo iterativo e adaptativo de desenvolvimento e vem ganhando cada vez mais adeptos devido a maneira organizada e consistente que permite conduzir um projeto.”(DevMedia, 2006)

O Modelo de Processo Unificado, é um processo de desenvolvimento de software que oferece uma abordagem disciplinada para projetar, implementar e documentar sistemas de software. O UP é baseado nos princípios da orientação a objetos e é uma variação do Processo Unificado (Rational Unified Process ou RUP).

O PU usa um ciclo iterativo de refinamentos e incrementos sucessivos para desenvolver software, convergindo gradualmente para um sistema adequado. Cada iteração é como um miniprojeto com análise de requisitos, projeto, implementação e testes.



Fonte: <https://www.devmedia.com.br/introducao-ao-processo-unificado/3931>

Alguns benefícios que o Processo Unificado (PU) oferece:

-**Gestão de Requisitos Eficiente**: O PU enfatiza a captura, documentação e rastreamento de requisitos de forma organizada, ajudando a garantir que o software atenda às necessidades dos stakeholders.

-**Qualidade do Software**: O foco na arquitetura e na modelagem visual promove a criação de software de alta qualidade, facilitando a detecção precoce de problemas.

-**Reutilização de Componentes**: O PU encoraja a criação de componentes reutilizáveis, economizando tempo e esforço no desenvolvimento de software futuro.

-**Maior Flexibilidade**: O processo iterativo permite que as equipes se adaptem às mudanças nos requisitos e nas condições do projeto.

### CONSIDERAÇÕES FINAIS

Como considerações finais de cada tema temos:

Fundamentos de Sistemas Computacionais:

Este tópico explora os princípios essenciais que sustentam a computação, abrangendo tanto a arquitetura de hardware como os fundamentos do software.

Componentes Básicos de um Computador:

Os elementos fundamentais de um computador incluem a CPU, memória, dispositivos de entrada e saída, bem como a placa-mãe que os interconecta, desempenhando papéis cruciais na execução de tarefas de processamento de dados.

Sistemas Numéricos:

Neste tema, são abordados os sistemas de numeração, como decimal, binário e hexadecimal, sendo fundamentais para representação e manipulação de dados em computação.

Álgebra Booleana:

A álgebra booleana se baseia em regras e operadores lógicos, manipulando valores verdadeiro/falso ou 0/1, essencial para a construção de circuitos lógicos e expressões condicionais em programação.

Introdução à Linguagem Python:

Python é uma linguagem de programação de alto nível conhecida por sua simplicidade e legibilidade, tornando-a uma escolha ideal para iniciantes.

Estrutura de Dados em Python:

Em Python, as estruturas de dados, como listas e dicionários, permitem organizar e manipular informações de forma eficaz, desempenhando um papel crucial na programação.

Python Orientado a Objetos:

Python oferece suporte à programação orientada a objetos, na qual objetos são instâncias de classes e métodos, proporcionando organização e reutilização de código.

Introdução à Análise de Dados em Python:

Este tema trata do uso de bibliotecas como Pandas e Matplotlib para extrair insights de conjuntos de dados, sendo essencial para a tomada de decisões embasadas.

Introdução à Engenharia de Software e à Análise de Sistemas:

Engenharia de software se refere ao desenvolvimento sistemático de software, enquanto a análise de sistemas envolve compreender os requisitos antes de iniciar o desenvolvimento, ambos essenciais para projetos de software bem-sucedidos.

Processo de Negócios para Análise de Sistemas:

Nesse contexto, os processos de negócios são mapeados e otimizados para identificar necessidades de automação e melhorias, sendo um passo fundamental na criação de sistemas eficazes.

Engenharia de Requisitos:

A engenharia de requisitos abrange a coleta, documentação e gestão dos requisitos do sistema de software, assegurando que o software atenda às necessidades dos usuários.

Paradigma Orientado a Objeto:

Esse paradigma de programação enfoca a modelagem do mundo real por meio de objetos e suas interações, promovendo a reutilização de código e a modularidade.

Esses temas abrangem uma ampla gama de conceitos fundamentais na computação, desde o hardware básico até a programação avançada e a engenharia de software. Compreender esses conceitos é fundamental

### REFERÊNCIAS

TRISTÃO, Isadora. **“Números binários: o que são, para que servem e transformações”** Conhecimento Científico.

Disponível em: <https://conhecimentocientifico.r7.com/numeros-binarios/>

Acesso em: 08/09/2023

OLIVEIRA, Raul Rodrigues de. **“Sistemas de Numeração”** Mundo Educação.

Disponível em: <https://mundoeducacao.uol.com.br/matematica/sistema-numeracao.htm>

Acesso em: 12/09/2023

ALECRIM, Emerson; MARQUES, Ana. **“O que é CPU? Conheça o principal processador de computadores e celulares.”** Tecnoblog.

Disponível em: <https://tecnoblog.net/responde/o-que-e-cpu-unidade-central-de-processamento/#:~:text=Conhe%C3%A7a%20o%20principal%20processador%20de%20computadores%20e%20celulares,-Entenda%20qual%20%C3%A9&text=A%20CPU%20(Central%20Processing%20Unit,o%20%E2%80%9Cc%C3%A9rebro%E2%80%9D%20desses%20equipamentos.>

Acesso em: 12/09/2023

MACHADO, Emerson. **“Dispositivos de entrada e saída: o que são, diferença e exemplos.”** Significados.

Disponível em: <https://www.significados.com.br/dispositivos-de-entrada-e-saida/#:~:text=Os%20dispositivos%20de%20entrada%20s%C3%A3o,um%20computador%20para%20comunicar%20informa%C3%A7%C3%B5es.>

Acesso em: 15/09/2023

BRITO, Rogério Alves de. **“Charles Babbage: um cientista muito além de seu tempo.”** Tecmundo.

Disponível em: <https://www.tecmundo.com.br/historia/16641-charles-babbage-um-cientista-muito-alem-de-seu-tempo.htm>

Acesso em: 12/09/2023

NORGREN. **“Computador Babbage”**, 2023.

Acesso em: 13/09/2023

PORTO EDITORA. “Máquina de Turing na Infopédia”, 2023.

Disponível em:<https://www.ufrgs.br/alanturingbrasil2012/Maquina_de_Turing.pd>f

Acesso em: 13/09/2023

LAMIM, JONATHAN. **“Primeiro computador moderno foi programado por mulheres”** 2023.

Disponível em: Primeiro computador moderno foi programado por mulheres; veja história (techtudo.com.br)

Acesso em: 13/09/2023

ANDRADE, Ana Paula de. **“Orientação a Objetos em Python”**; Treina Web, 2021

Disponível em: <https://www.treinaweb.com.br/blog/orientacao-a-objetos-em-python>

Acesso em: 27/09/23

RAMOS, Vinícius. **“Programação Orientada a Objetos no Python: Introdução”**; Python Acadamy, 2023

Disponível em: <https://pythonacademy.com.br/blog/introducao-a-programacao-orientada-a-objetos-no-python>

Acesso em: 27/09/23

ANDRADE, Emmanuel, **“3 Formas de Iterar uma Lista em Python”**; Dio, 2023.

Disponível em: <https://www.dio.me/articles/3-formas-de-iterar-uma-lista-em-python>

Acesso em: 27/09/23

AWARI; **“Explorando o Método Append em Python: Dicas e Truques”**; 2023

Disponível em: <https://awari.com.br/metodo-append-em-python/#:~:text=com%20a%20Awari-,O%20que%20%C3%A9%20M%C3%A9todo%20Append%20em%20Python,nova%20lista%20a%20cada%20vez>.

Acesso em: 27/09/23

PINHEIRO, Fagner; **“Utilizando Herança no Python”**; Treina Web, 2023

Disponível em: <https://www.treinaweb.com.br/blog/utilizando-heranca-no-python>

Acesso em: 27/09/23

ALURA; **“Herança e Polimorfismo”**, 2023

Disponível em: <https://www.alura.com.br/apostila-python-orientacao-a-objetos/heranca-e-classes-abstratas?utm_term=&utm_campaign=%5BSearch%5D+%5BPerformance%5D+-+Dynamic+Search+Ads&utm_source=adwords&utm_medium=ppc&hsa_acc=7964138385&hsa_cam=1560195067&hsa_grp=63243218150&hsa_ad=401937480958&hsa_src=g&hsa_tgt=aud-409949667324:dsa-758241041973&hsa_kw=&hsa_mt=&hsa_net=adwords&hsa_ver=3&gclid=Cj0KCQjwvL-oBhCxARIsAHkOiu0gLsB5QACb7-flAn7Sw2EVUCqlAKEnE40ktoTd10f5xGt6psRhGZIaAkCGEALw_wcB>

Acesso em: 27/09/23

Pytho Org; **“A Biblioteca Padrão do Python”**;

Disponível em:<https://docs.python.org/pt-br/3/library/>

Acesso em: 27/09/23

FUNDAÇÃO BRADESCO; **“Modelagem de dados”**; Escola Virtual, 2023

ORG, Python; “10. Um breve passeio pela biblioteca padrão”

Disponível em:[**https://docs.python.org/pt-br/3/tutorial/stdlib.html**](https://docs.python.org/pt-br/3/tutorial/stdlib.html)

Acesso em: 27/09/23

SOARES, Gustavo; **“Python: Imports e Bibliotecas”**; DevFuria, 2023

Disponível em: [**https://dev.to/iugstav/python-imports-e-bibliotecas-42c0**](https://dev.to/iugstav/python-imports-e-bibliotecas-42c0)

Acesso em: 27/09/23

PINHEIRO, Fagner; **“Gerenciando Pacotes em Projetos Python com o PIP”**; TreinaWeb, 2018

Disponível em: <https://www.treinaweb.com.br/blog/gerenciando-pacotes-em-projetos-python-com-o-pip>

Acesso em: 27/09/23

ORG, Python; **“A Biblioteca Padrão do Python”**

Disponível em: <https://docs.python.org/pt-br/3/library/>

Acesso em: 27/09/23

Pense Python; **“Modulos e Obtendo Ajuda”**; Panda, USP

Disponível em: <https://panda.ime.usp.br/pensepy/static/pensepy/04-Modulos/modulos.html#:~:text=Um%20m%C3%B3dulo%20%C3%A9%20um%20arquivo,deles%20intensamente%2C%20o%20m%C3%B3dulo%20turtle%20>.

Acesso em: 27/09/23

Geo Sem Fronteiras; **“5 Bibliotecas Python Para Conhecer”**

Disponível em: [**https://geosemfronteiras.org/blog/5-bibliotecas-python-para-conhecer/**](https://geosemfronteiras.org/blog/5-bibliotecas-python-para-conhecer/)

Acesso em: 27/09/23

Algoritmos Em Python; **“Módulos e Bibliotecas”**

Disponível em: <https://algoritmosempython.com.br/cursos/programacao-python/modulos-bibliotecas/#:~:text=Portanto%2C%20a%20biblioteca%20padr%C3%A3o%20Python,funcionalidades%20deles%20quando%20bem%20quiser>

Acesso em: 27/09/23

CANDIDO; **“Função \_\_init\_\_py”**, School Of Net

Disponível em: <https://schoolofnet.com/forum/topico/qual-a-necessidadefuncao-do-__init__py-4308>

Acesso em: 27/09/23

Engholm, Jr. Hélio. **“Paradigma Orientado a Objetos”** Profissionais TI.

Disponível em: <https://www.profissionaisti.com.br/paradigma-orientado-a-objetos-afinal-o-que-e-um-objeto/>

Acesso em: 18/10/2023

Carvalho, L. Thiago. **“Orientação a Objetos”.**

Disponível em: <https://makecreativecode.com/orientacao-a-objetos-heranca-encapsulamento/>

Acesso em: 18/10/2023

Stasaitis, Tiago. **“O que são métodos na programação orientada a objetos?”**.

Disponível em: <https://www.academicotech.com/post/o-que-sao-metodos-na-programacao-orientada-a-objetos-com-exemplos-em-java-e-python>

Acesso em: 20/10/2023

DevMedia. **“Introdução ao Processo Unificado”.**

Disponível em: <https://www.devmedia.com.br/introducao-ao-processo-unificado/3931>

Acesso em: 23/10/2023

VALENTE, Marco Tulio; **Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade.** 1.12022

Acesso em: 16/10/2023

Disponível em:

[https://engsoftmoderna.info/cap1.html#definições-contexto-e-história](https://engsoftmoderna.info/cap1.html#defini%C3%A7%C3%B5es-contexto-e-hist%C3%B3ria)

Da Redação; **Analista de Sistemas: guia completo sobre a profissão**. 2 2023

Acesso em: 18/10/2023

Disponível em:

[https://educacaoeprofissao.com.br/analista-de-sistemas-guia-completo-sobre-a-profissao/#:~:text=A%20profissão%20de%20Analista%20de%20Sistemas%20surgiu%20com%20o%20avanço,especializados%20em%20TI%20cresceu%20exponencialmente](https://educacaoeprofissao.com.br/analista-de-sistemas-guia-completo-sobre-a-profissao/#:~:text=A%20profiss%C3%A3o%20de%20Analista%20de%20Sistemas%20surgiu%20com%20o%20avan%C3%A7o,especializados%20em%20TI%20cresceu%20exponencialmente).

Da Redação; **Analista de Sistemas: guia completo sobre a profissão**. 3 e 4 2023

Acesso em: 18/10/2023

Disponível em:

[https://educacaoeprofissao.com.br/analista-de-sistemas-guia-completo-sobre-a-profissao/#:~:text=A%20profissão%20de%20Analista%20de%20Sistemas%20surgiu%20com%20o%20avanço,especializados%20em%20TI%20cresceu%20exponencialmente](https://educacaoeprofissao.com.br/analista-de-sistemas-guia-completo-sobre-a-profissao/#:~:text=A%20profiss%C3%A3o%20de%20Analista%20de%20Sistemas%20surgiu%20com%20o%20avan%C3%A7o,especializados%20em%20TI%20cresceu%20exponencialmente).

ZANCHETT, **Pedro Sidnei; Engenharia e Projeto de Software**. Tópico 2, 1. 2015

Acesso em: 20/10/2023

Disponível em: <https://www.uniasselvi.com.br/extranet/layout/request/trilha/materiais/livro/livro.php?codigo=23854>

ZANCHETT, **Pedro Sidnei; Engenharia e Projeto de Software**. Tópico 3, 2. 2015

Acesso em: 20/10/2023

Disponível em: <https://www.uniasselvi.com.br/extranet/layout/request/trilha/materiais/livro/livro.php?codigo=23854>

ZANCHETT, **Pedro Sidnei; Engenharia e Projeto de Software**. Tópico 2, 2. 2015

Acesso em: 21/10/2023

Disponível em: <https://www.uniasselvi.com.br/extranet/layout/request/trilha/materiais/livro/livro.php?codigo=23854>

VALENTE, Marco Tulio; **Engenharia de Software Moderna.** 3.2. 2022

Acesso em: 17/10/2023

Disponível em: <https://engsoftmoderna.info/cap3.html#:~:text=3.2%20Engenharia%20de%20Requisitos%20%F0%9F%94%97,dos%20requisitos%20de%20um%20sistema>.

ROVEGA, Ugo; **ENGENHARIA DE REQUISITOS DE SOFTWARE: O QUE É E COMO FUNCIONA;** Kenzie; 2022.

Acesso em: 17/10/2023

Disponível em:

<https://kenzie.com.br/blog/engenharia-de-requisitos-de-software/>

ELAINE; **Trabalhando com Engenharia de Requisitos**; DevMedia; 2014

Acesso em: 17/10/2023

Disponível em:

<https://www.devmedia.com.br/trabalhando-com-engenharia-de-requisitos/30207>

QUITÉRIO, Ana Paula; **Análise de Requisitos** ; InfoEscola

Disponível em: <https://www.infoescola.com/engenharia-de-software/analise-de-requisitos/>

Acesso em: 21/10/2023

COELHO, Jailton; **Gerenciamento de Requisitos ;** Lean TI

Disponível em:

<https://www.leanti.com.br/artigos/16/gerenciamento-de-requisitos.aspx>

Acesso em: 21/10/2023

### 