Aula 01 - Conceitos fundamentais da Programação

Capítulo 1 - Introdução

Você já imaginou como seria a sua vida sem todos os aparelhos e ferramentas que fazem parte do seu cotidiano? Como seria viver na pré-história, como os homens das cavernas? Eu acredito que para muitos, viver sem computadores ou smartphones representaria o fim do mundo! Todas essas coisas que tanto gostamos e que, de certa forma, entraram em nosso cotidiano para nunca mais sair, tornam mais prático e confortável viver nos tempos modernos. Mas você já se perguntou realmente como essas coisas funcionam? Bem, se essa pergunta já passou pela sua cabeça, você está no lugar certo!

Quando pensamos em tecnologias, é crucial mencionar que todo avanço tecnológico é resultado da colaboração de diversas pessoas nos mais variados contextos ao longo da história da humanidade. Ao estudarmos a história, é comum analisarmos os fatos históricos por meio de uma visão geral da sociedade, focando em como nós, seres humanos, nos relacionamos, e muitas vezes esquecemos de levar em consideração como as tecnologias influenciaram a própria humanidade.

É fundamental reconhecer que as inovações tecnológicas não apenas moldaram a forma como vivemos, mas também tiveram um impacto significativo nas interações sociais, no desenvolvimento econômico e na evolução cultural. Ao compreendermos o papel das tecnologias na história, conseguimos enxergar como essas mudanças moldaram não apenas o nosso ambiente, mas também a maneira como nos percebemos como sociedade. A interação entre a tecnologia e a humanidade é um aspecto vital para uma compreensão abrangente da evolução histórica.

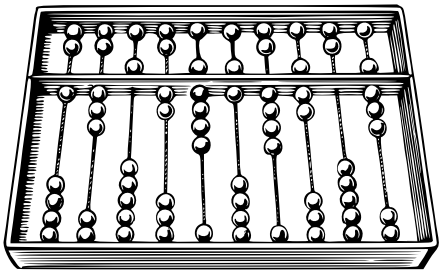
Podemos identificar inúmeras tecnologias que transformaram o mundo ao longo da história, desde a revolução agrícola no período neolítico (8 mil a.C. a 5 mil a.C.), passando pelos estudos e formulação da mecânica newtoniana, a descoberta do elétron, dos fenômenos eletromagnéticos, até o isolamento da penicilina, e chegando à mecânica quântica e relativística,

entre outras invenções e descobertas técnicas e científicas. Toda a ciência que conhecemos hoje é fundamentada em invenções primordiais e antiquíssimas, sendo a comunicação e, por conseguinte, o uso e domínio da linguagem, uma das mais essenciais.

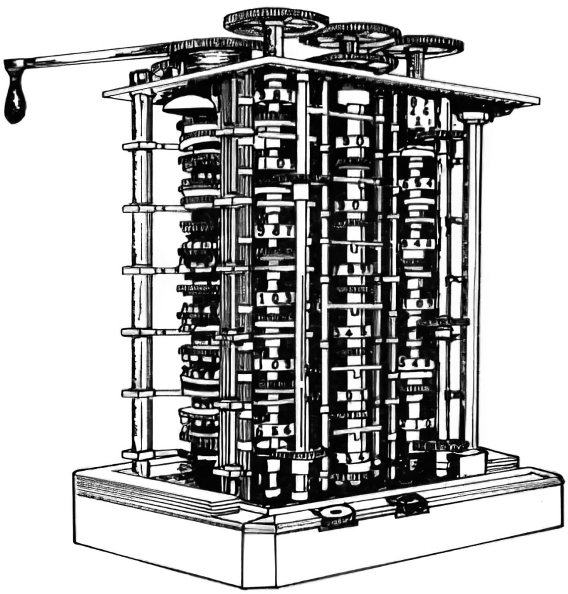
A habilidade de nos expressarmos e nos comunicarmos efetivamente foi crucial para o desenvolvimento de sociedades complexas e o avanço do conhecimento humano. A linguagem é, por si só, uma tecnologia ancestral que desempenhou um papel fundamental na transmissão de informações, no estabelecimento de culturas e na evolução das sociedades ao longo do tempo. Ao reconhecermos a importância da linguagem como uma inovação inicial, podemos compreender como ela serviu de base para a complexa teia de descobertas e avanços científicos que moldaram o nosso mundo contemporâneo.

A linguagem é tão importante que o momento que os seres humanos começaram a conseguir se comunicar e utilizar linguagem com máquinas deu origem a uma das maiores se não a maior onda de revoluções tecnológicas já vista a revolução da computação digital.

A história da computação remonta a milhares de anos, com o ábaco sendo um dos primeiros dispositivos utilizados para auxiliar em cálculos matemáticos. Originário da Mesopotâmia e posteriormente aprimorado por diversas culturas, o ábaco representou um avanço significativo na capacidade humana de lidar com números. No entanto, foi apenas no século XIX que Charles Babbage, junto a primeira programadora da história, Ada Lovelace, projetou a Máquina Analítica, um dispositivo mecânico projetado para realizar cálculos complexos por meio de cartões perfurados. Embora nunca tenha sido construída em sua totalidade, essa máquina é considerada um precursor dos computadores modernos.



Descrição de Imagem Ábaco



Máquina Análitica

No início do século XX, veio o advento de dispositivos eletromecânicos, como as máquinas de Hollerith, usadas para processar dados censitários. Contudo, foi com os esforços durante a Segunda Guerra Mundial que surgiram os primeiros computadores digitais, como o ENIAC nos Estados Unidos. Essas máquinas enormes e complexas marcaram o início da era digital, impulsionando a pesquisa e o desenvolvimento de computadores mais avançados.

Nas décadas seguintes, a computação evoluiu rapidamente, com a miniaturização de componentes eletrônicos, levando ao desenvolvimento de computadores pessoais nas décadas de 1970 e 1980. A ascensão da internet na década de 1990 e o advento da computação em nuvem na virada do século XXI transformaram radicalmente a forma como interagimos com a informação. Hoje, vivemos em uma era de computação ubíqua, com dispositivos interconectados, inteligência artificial em ascensão e a capacidade de processamento computacional aumentando exponencialmente com o passar dos anos.

O estado atual da computação é caracterizado por avanços constantes em inteligência artificial, aprendizado de máquina, análise de big data e computação quântica. A tecnologia está mudando não apenas o mundo digital, mas também influenciando profundamente setores como saúde, transporte, educação e indústria. A história da computação é uma jornada fascinante de inovação, desde os primórdios do ábaco até os horizontes promissores da computação quântica e da inteligência artificial.

A questão que permanece no ar é: como podemos estabelecer comunicação com as máquinas? A resposta aparentemente óbvia para todos seria aprender a linguagem das máquinas. Mas o que isso realmente significa? Desde quando as máquinas têm a capacidade de falar ou se expressar? Bem, quase isso. Quando nos referimos especificamente a computadores, estamos falando de máquinas projetadas para realizar cálculos de maneira mais eficiente do que os humanos. De certa forma, os computadores são capazes de compreender, através de mecanismos e métodos desenvolvidos no contexto da computação, as operações matemáticas e lidar com os resultados dessas operações.

Ao abordarmos a comunicação com as máquinas, entramos no âmbito da **linguagem de programação**, um conjunto de comandos que permite aos humanos interagir e transmitir tarefas aos computadores. Essa linguagem, composta por códigos e algoritmos, atua como uma ponte entre a compreensão humana e a capacidade de processamento computacional. Ao aprendermos a programar, estamos, de fato, adquirindo a habilidade de conversar com as máquinas, instruindo-as sobre as tarefas a serem realizadas.

Portanto, quando nos referimos à linguagem das máquinas, não estamos tratando de um diálogo audível, mas sim de uma comunicação baseada em comandos lógicos e operações matemáticas. Esse entendimento mútuo entre humanos e computadores é crucial para a evolução da tecnologia, permitindo-nos explorar novas fronteiras na automação, inteligência artificial e diversas outras áreas. Assim, a linguagem das máquinas torna-se não apenas um aspecto técnico, mas uma ferramenta fundamental que impulsiona a inovação e o progresso tecnológico.

Os cientistas capazes de conversar com os computadores são chamados de PROGRAMADORES, pois realizam a tarefa de PROGRAMAR COMPUTADORES. Mas afinal, o que é programar um computador? Pense da seguinte maneira: você é um programador e sua mãe é o computador com o qual você consegue se comunicar. Você precisa acordar mais cedo do que o normal para passar na casa do(a) seu(ua) crush e levar flores. Para isso, você pede à sua mãe para acordá-lo no dia seguinte, mas ela não entende o pedido, pois você não especificou a que horas ela deve acordá-lo. Então, ela pergunta a que horas você precisa acordar. Logo em seguida, você diz à sua mãe que precisa acordar às 6:00 horas da manhã. Dessa forma, ela compreende e você consegue agradar o seu amor com lindas flores. O que você acabou de fazer foi programar a sua mãe! Programar é nada mais do que criar uma série lógica de comandos para realizar algo por você! Em algum momento de nossas vidas, já fomos programadores, seja ao programar a máquina de lavar roupas, o alarme ou até mesmo o micro-ondas.

Mas qual é a diferença para ser um programador de computadores? Bem, um programador de computadores é capaz de programar conscientemente para realizar tarefas específicas utilizando princípios lógicos. Ou seja, ele consegue, por meio do raciocínio, criar uma série de comandos que o computador entenderá e executará uma tarefa.

Neste ponto, o caro leitor deve estar se perguntando como exatamente o programador conversa com uma máquina, uma pergunta bastante astuta por sinal ! Para responder essa pergunta temos que entender um pouco sobre como de fato um computador funciona. Como dito anteriormente , os computadores são máquinas que realizam cálculos ,desse modo essencialmente os computadores entendem apenas números , especificamente um sistema numérico baseado em lógica BINÁRIA. Para um computador digital o que ele conhece é apenas se há ou não energia em determinada parte de seu circuito. Não compreendeu como podemos nos comunicar com computadores através de números? Não se preocupe! Imagine duas lâmpadas, uma acesa e outra apagada.

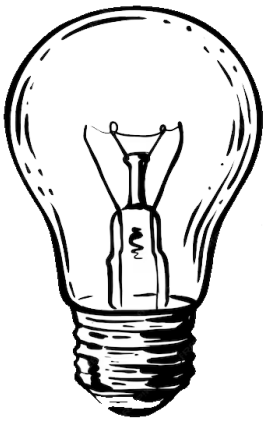


Imagem de lâmpada referenciando o nível lógico baixo

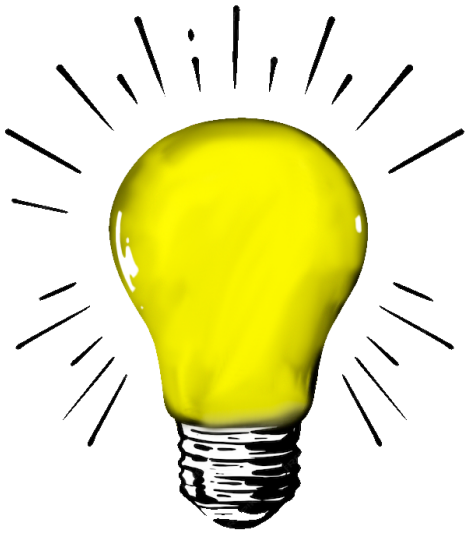


Imagem de lâmpada referenciando o nível lógico alto

Quando a lâmpada está apagada e não recebe energia, convencionou-se dizer que essa lâmpada está em um estado com valor 0. Por outro lado, quando essa lâmpada está recebendo energia, dizemos que ela está em valor 1. Todos os computadores operam essencialmente como essas duas lâmpadas, armazenando valores 0 ou 1. Esse tipo de sistema é denominado sistema

binário. Com números binários podemos representar qualquer quantidade que desejamos e podemos realizar operações de forma simples e eficiente.

No cerne da linguagem dos computadores reside o sistema binário, fundamental para a representação e manipulação de dados. Diferentemente do sistema decimal que utilizamos no dia a dia, o sistema binário opera apenas com dois dígitos, 0 e 1. Esses dígitos, denominados bits (abreviação de binary digits), formam a base do código de máquina utilizado pelos computadores.

Em um número binário, cada posição corresponde a uma potência de 2. Por exemplo, o número binário 1101 representa $1\ 2^3 + 1\ 2^2 + 0\ 2^1 + 1\ 2^0$,que é equivalente a 13 em decimal. A simplicidade do sistema binário é essencial para os computadores, pois os componentes eletrônicos podem distinguir facilmente entre dois estados (ligado/desligado), correspondentes aos dígitos binários 0 e 1.

As operações binárias, como adição, subtração, multiplicação e divisão, são executadas de maneira semelhante às operações em sistemas numéricos mais familiares. No entanto, a simplicidade do sistema binário torna as operações mais diretas. Por exemplo, na adição binária, 0 + 0 é igual a 0, 0 + 1 é igual a 1, 1 + 1 é igual a 10 (em binário). Em outras palavras, a adição binária segue as regras de uma adição normal, mas com a base 2.

Veja a sessão extra : Números e operações binárias

1.1 Programas de computado

Até o momento, discutimos como os computadores funcionam a nível de **hardware,** ou seja, os componentes físicos. No entanto, para que o computador realmente execute uma tarefa, ele precisa compreender estruturas lógicas (sintáticas e semânticas) que façam sentido também para o ser humano. Essas estruturas lógicas constituem o **software** do computador, em outras palavras, os programas.

Para entender de forma simples o que é um programa de computador, vejamos o exemplo de um carrinho de controle remoto. Para controlar o carrinho, utilizamos um controle que possui as opções de fazer o carrinho ir para frente, para trás, para a direita e para a esquerda. Esses botões no controle representam as atividades mais simples que conseguimos fazer com o carrinho e por isso são chamadas de **instruções**. As instruções traduzem operações de hardware para uma forma simbólica. Por exemplo quando pressionamos o botão para frente no carrinho possibilitamos o contato físico que liga os motores do carrinho. De forma análoga o computador

enxerga um conjunto de instruções que estão atreladas a operações específicas , como soma , subtração, multiplicação entre outros.

Essa abordagem mais próxima do hardware e a nível de instruções é uma abordagem na computação chamada de **baixo nível ,** isso porque estamos lidando mais com a parte física do computador, o que acaba sendo mais complicado para o ser humano entender. Ainda analisando o carrinho de controle remoto , imagine que devemos fazer uma curva de 45 º à direita, como podemos fazer isso utilizando apenas os botões apresentados anteriormente ? Nesse caso devemos combinar a instrução de virar à direita com a instrução de ir para frente por um determinado período de tempo , quando combinamos instruções para realizar alguma tarefa estamos na verdade executando **comandos**. Nesse sentido , dessas 4 instruções podemos obter vários comandos , como fazer curva à direita , fazer curva à esquerda , girar no próprio eixo , parar entre outros . Quanto mais simples os comandos para o entendimento dos seres humanos , mais **alto nível** é a abordagem. As linguagens de programação são nada mais nada menos que padronização desses comandos , quanto mais perto do conjunto de instruções a linguagem é de tida como **linguagem de baixo nível** , quanto mais perto os comandos estão da linguagem humana , a linguagem é tida como **linguagem de alto nível.** Exemplos de linguagens de baixo nível são C , Cobol e Fortran, exemplos de linguagem de alto nível são Python, JavaScript , Scratch .

Com esse olhar, nossa tarefa agora é fazer o carrinho percorrer um percurso em formato quadrado. Podemos seguir a seguinte sequência de comandos:

1. Mover para frente por 2 segundos.
2. Virar à direita.
3. Mover para frente por 2 segundos.
4. Virar à direita.
5. Mover para frente por 2 segundos.
6. Virar à direita.
7. Mover para frente por 2 segundos.
8. Virar à direita.

Vamos analisar essa sequência. Quando colocamos uma sequência de comandos para serem executados de forma linear (etapa por etapa) e lógica a fim de executar uma determinada tarefa em um número finito de passos, estamos escrevendo um **algoritmo.**

Algoritmos estão presentes em nosso dia a dia, muitas vezes sem que percebamos. Por exemplo, ao seguir o passo a passo de uma receita, estamos executando um algoritmo.

Exemplo: Receita de Bolo

1. Preaquecer o forno a 180 graus Celsius.
2. Misturar 2 xícaras de farinha, 1 xícara de açúcar e 1 colher de sopa de fermento.
3. Adicionar 1 xícara de leite, 2 ovos e 1/2 xícara de óleo.
4. Mexer até obter uma massa homogênea.
5. Despejar a massa em uma forma untada.
6. Assar por 35 minutos ou até dourar.
7. Cada passo nessa receita é uma instrução que deve ser seguida em ordem para obter o resultado desejado: um bolo assado.

Quando implementamos esses algoritmos em uma linguagem de programação, estamos escrevendo um **programa de computador**. Vamos ver como seria a implementação do percurso do carrinho em **pseudocódigo**, que é uma forma de escrever algoritmos que se assemelha a uma linguagem de programação, mas é mais fácil de entender.

Pseudocódigo para o Percurso Quadrado:

1. iniciarCarrinho()
2. moverParaFrente(2 segundos)
3. virarDireita()
4. moverParaFrente(2 segundos)
5. virarDireita()
6. moverParaFrente(2 segundos)
7. virarDireita()
8. moverParaFrente(2 segundos)
9. virarDireita()
10. pararCarrinho()

Os pseudocódigos são escritos na linguagem do programador e não precisam do rigor estrito presente nas sintaxes próprias de cada linguagem de programação. Porém, eles costumam fazer uso de palavras que possuem termos equivalentes na linguagem de programação, permitindo desta forma uma maior simplicidade no momento de serem traduzidos da linguagem humana para a linguagem de programação escolhida.

Perceba que, para nós, seres humanos, a tarefa de traduzir um algoritmo em um programa e entender a que se referem as instruções é realizada sem muito esforço. No entanto, para o computador, é necessário um programa especial responsável por essa tradução, chamado **compilador**. O compilador tem a função de traduzir uma linguagem de programação de alto nível em uma linguagem de máquina (baixo nível), que o computador pode entender e executar.

Normalmente os nossos computadores não possuem compiladores de todas as linguagens de programação existentes e por isso é necessário instalar um compilador da linguagem quando queremos programar para ela. Em nosso curso utilizaremos a linguagem de programação python , nos próximos capítulos mostraremos como fazer o processo de baixa e instalar o compilador do python.

1.2 A linguagem de programação Python

O Python foi criado no final dos anos 1980 por Guido van Rossum, um programador holandês. Ele começou o desenvolvimento da linguagem em dezembro de 1989, durante seu tempo no Centrum Wiskunde & Informatica (CWI) na Holanda. A inspiração para o nome "Python" veio do grupo de comédia britânico Monty Python, cujo trabalho Van Rossum apreciava. O objetivo principal era criar uma linguagem que fosse fácil de ler e escrever, promovendo a produtividade e a facilidade de uso.

A primeira versão oficial de Python, a 0.9.0, foi lançada em fevereiro de 1991. Esta versão inicial já incluía algumas das características essenciais da linguagem, como exceções, funções, e os tipos de dados mais básicos, como listas e strings. Ao longo dos anos 90, Python continuou a evoluir com a contribuição de uma crescente comunidade de desenvolvedores. Em 1994, foi lançada a versão 1.0, que trouxe a incorporação de módulos, o que permitiu a reutilização de código e a extensão das funcionalidades da linguagem.

Nos anos 2000, Python ganhou ainda mais tração com o lançamento da versão 2.0 em outubro de 2000. Esta versão trouxe novas funcionalidades importantes, como a coleta de lixo por contagem de referências e a introdução da compreensão de listas, que simplificam significativamente a manipulação de dados. Além disso, a criação da Python Software

Foundation (PSF) em 2001 ajudou a organizar o desenvolvimento da linguagem e promover seu uso em diversos setores, desde o desenvolvimento web até a ciência de dados.

A transição para Python 3 marcou um grande marco na história da linguagem. Lançado em dezembro de 2008, Python 3 foi projetado para corrigir as inconsistências e melhorar a usabilidade da linguagem. Embora isso tenha introduzido mudanças incompatíveis com as versões anteriores, a intenção era criar uma base mais sólida para o futuro. Essa transição foi um desafio significativo para a comunidade, mas, ao longo do tempo, a adoção de Python 3 aumentou, especialmente após o fim do suporte oficial para Python 2 em janeiro de 2020.

Hoje, Python é uma das linguagens de programação mais populares do mundo, amplamente utilizada em uma variedade de campos, incluindo desenvolvimento web, automação, ciência de dados, inteligência artificial e mais. Sua simplicidade e versatilidade continuam a atrair novos programadores e a fomentar uma comunidade vibrante. Com bibliotecas robustas e um ecossistema em constante crescimento, Python permanece uma escolha preferida tanto para iniciantes quanto para desenvolvedores experientes.

Agora que você conhece um pouco sobre linguagens de programação e sobre como os computadores são capazes de entender essas linguagens, podemos começar a nos aventurar um pouco mais em como trabalhar com essas tecnologias. O primeiro passo da nossa aventura é configurar o **ambiente de desenvolvimento.** O ambiente de desenvolvimento é o local que utilizaremos para poder executar os comandos na linguagem de programação de uma forma mais simples e rápida. Nos próximos capítulos veremos como configurar esse ambiente no python e como escrever os nossos primeiros programas.

Capítulo 3 - Primeiros Passos

Com o ambiente de desenvolvimento configurado , podemos agora escrever os nossos primeiros programas, mas antes vamos entender um pouco mais sobre como os programas que usamos no dia a dia funcionam.

Todos os programas de computador necessitam de algumas informações que nós, usuários, fornecemos, como endereços de e-mail, senhas e números de telefone. Essa troca de informações é semelhante à comunicação humana, onde perguntas erradas levam a respostas erradas.

Todas as informações que fornecemos ao computador são tratadas como *inputs* (do inglês, entradas). As respostas que o computador nos retorna são chamadas de *outputs* (do inglês,

saídas). Um exemplo de output pode ser um vídeo exibido na tela do computador, um texto impresso por uma impressora ou uma música reproduzida no celular.

O fluxo natural de quase todo programa de computador é receber inputs do usuários , realizar um processamento e retornar algum output com a resposta.

> Vamos fazer o nosso primeiro programa

O nosso primeiro programa é um clássico da computação e será um programa que imprime na tela o texto “olá mundo !” . Para fazer isso devemos apenas lembrar que em inglês a palavra imprimir é ‘print’.

```
print("Olá Mundo")
```

3.1 Memórias e tipos de dados

Para entender melhor o funcionamento de um computador e como podemos programá-lo, é essencial primeiro compreender como ele lê e escreve dados, ou seja, como utiliza a memória. Mas, espera aí! Computadores têm memória?

Sim, se um computador lembra de cálculos, então ele precisa guardar esses cálculos em algum lugar! Na computação, existem dois tipos de memória: as memórias de acesso rápido, como a memória RAM; e as memórias de armazenamento, como o HD e o SSD.

Essas memórias trabalham em conjunto com o processador do computador. É um pouco confuso no início, mas basta lembrar que o processador é o cérebro do computador. Ele é responsável por acessar e gravar dados na memória. A diferença entre as memórias de acesso rápido e as de armazenamento está em como o processador acessa cada uma delas.

Imagine que o processador seja como nosso cérebro quando estamos na fila de um banco e recebemos uma senha de atendimento de um dos atendentes. Esperamos um tempo na fila, somos atendidos e voltamos para casa com o número da nossa nova conta bancária. Passados alguns dias ou até horas, o que você lembraria desse momento no banco? Dificilmente se lembraria de quem lhe deu a senha, do número exato da senha ou das pessoas na fila. Porém, certamente lembraria ou conseguiria acessar o número da sua conta bancária. Em ambos os casos, estamos usando nosso cérebro para processar informações, mas lembramos apenas dos detalhes que consideramos importantes.

Sim, isso acontece porque o cérebro humano utiliza diferentes tipos de memória para processar informações, da mesma maneira que um computador. Quando estamos na fila de um banco, usamos uma memória de curto prazo que só pode processar e compreender informações temporariamente. Quando o cérebro considera uma informação importante, ela é armazenada na memória de longo prazo, capaz de reter informações por mais tempo.

Nos computadores, a memória de curto prazo é representada pela memória RAM (Random Access Memory). A RAM é utilizada pelos programas em execução, e é essencial que os programadores evitem sobrecarregá-la. Quando a memória RAM é sobrecarregada, os programas podem terminar inesperadamente e até comprometer o funcionamento do computador.

Enquanto a memória RAM se assemelha à memória de curto prazo do nosso cérebro, as memórias de armazenamento, como HD e SSD, são comparáveis à memória de longo prazo. Essas memórias armazenam os dados que o processador considera importantes para serem acessados posteriormente pelo computador.

Agora que entendemos o conceito de memória e seus tipos em um computador, vamos explorar uma parte fundamental da programação: os dados. Mas afinal, o que são dados em um contexto computacional?

Dados são informações brutas, a matéria-prima da qual podemos extrair significado. Eles são armazenados em formatos específicos e podem ser temporários (como na memória RAM) ou permanentes (como no HD). Os dados variam amplamente em natureza e podem representar números, textos, imagens, entre outros tipos de informação.

Para ilustrar, imagine uma agenda que pode armazenar até 10 contatos por vez. Cada espaço para salvar um contato na agenda representa um espaço acessado pelo processador na memória RAM. Cada contato na agenda contém dados diferentes, como números de telefone e nomes, cada um representando uma natureza específica de informação.

Quando a agenda atinge sua capacidade máxima, o processador decide quais informações devem ser transferidas para uma agenda maior (como o HD ou SSD) e elimina os dados que não são mais necessários.

Portanto, dados são essenciais porque são a matéria-prima para a criação de informação. Informação, por sua vez, é o dado processado, organizado com significado e contexto definidos. Por exemplo, na agenda, a informação seria o conjunto organizado de contatos com números e nomes claros. Por fim, conhecimento é o uso inteligente da informação, aplicando-a de forma contextualizada na prática, como ao utilizar os contatos da agenda para fazer ligações ou enviar mensagens.

Essa distinção entre dados, informação e conhecimento é fundamental para entender como os computadores processam e utilizam informações em diferentes contextos.

Os dados podem ser agrupados de acordo com caracteristicas semelhantes que possuem. No python agrupamos nos seguintes grupos:

- Tipos numéricos
- Tipos Textuais
- Tipos Binários
- Tipos booleando
- Tipo None
- Tipos de sequências
- Tipos de mapeamento

- Tipo Objeto
- Tipo Class
- Tipos Set

Você vai perceber que cada natureza de dado recebe um nome, por exemplo um texto recebe o nome de String, um número inteiro recebe o nome de Inteiro e que cada tipo recebe uma sigla, por exemplo inteiro é abreviado para int. A seguir iremos estudar cada tipo de dado disponível no python.

A tabela abaixo resume os principais tipos de dados:

Tipo de Dado	Tipos
Texto	`str`
Tipos Numéricos	`int`, `float`, `complex`
Tipos de Sequência	`list`, `tuple`, `range`
Tipo de Mapeamento	`dict`
Tipos de Conjunto	`set`, `frozenset`
Tipo Booleano	`bool`
Tipos Binários	`bytes`, `bytearray`, `memoryview`
Tipo Nenhum	`NoneType`

O Python possui um comando que nos permite descobrir com que tipo de dados estamos trabalhando `type()`

```
print(type(20))

# Deve imprimir <class 'int'> ou seja é do tipo inteiro
```

Vamos aprender um pouco mais sobre os principais tipos de dados no python.

3.1.1 Texto

No Python, o tipo de dado texto é representado pela classe `str` (string). Uma string é uma sequência de caracteres, onde cada caractere pode ser qualquer símbolo, incluindo letras, números, espaços, pontuações, entre outros.

```
print('Python')
```

Neste caso, `"Python"` é uma string que contém os caracteres "P", "y", "t", "h", "o" e "n". Cada um desses caracteres é um `char` (caractere) que compõe a string.

3.1.2 Numerico

Nos programas em Python, existem três tipos principais de dados numéricos: inteiros (`int`), números de ponto flutuante (`float`) e números complexos (`complex`).

int (Número Inteiro)

O tipo `int` em Python representa números inteiros, ou seja, números sem parte decimal. Exemplos de inteiros são `-5`, `0`, `42`, `1000`, etc. Em Python, inteiros podem ter qualquer tamanho, não estão limitados por um tamanho fixo como em algumas outras linguagens de programação.

Exemplos:

```
42
-10
0
```

float (Número de Ponto Flutuante)

O tipo `float` em Python é usado para representar números reais, ou seja, números que têm uma parte decimal. Exemplos de floats são `3.14`, `-0.5`, `2.71828`, etc. Floats são utilizados para representar números precisos que incluem frações decimais.

Exemplos:

```
3.14
-2.5
0.0
```

complex (Número Complexo)

O tipo `complex` em Python representa números complexos, que são números com uma parte real e uma parte imaginária. Em Python, números complexos são escritos na forma `a + bj`, onde `a` é a parte real, `b` é a parte imaginária e `j` denota a unidade imaginária (√-1).

Exemplos:

```
1 + 2j
-3.5 + 0.6j
```

3.1.3 bool (Tipo Booleano)

O tipo `bool` em Python é utilizado para representar valores lógicos, que podem ser `True` (verdadeiro) ou `False` (falso). Este tipo de dado é fundamental para operações de lógica e controle de fluxo em programas.

Exemplos típicos de uso incluem expressões condicionais e avaliações de verdadeiro ou falso:

- `True` representa uma condição verdadeira.
- `False` representa uma condição falsa.

    Em algumas situações o interpretador do python pode enteder o inteiro 1 como verdadeiro e o inteiro 0 como falso.

3.2 Variáveis

Conhecendo o que são dados e como o processador acessa esses dados na memória, você deve estar se perguntando como nós, como programadores, podemos criar, acessar e manipular esses dados na memória. Bem, boa pergunta! Para isso devemos utilizar uma ferramenta chamada variável. Uma variável é um espaço vazio ocupado na memória! Tá, eu sei, não ajudou muito. Uma variável é como se guardássemos um dos espaços para contato em branco, para quê? Bem, para preencher mais tarde exatamente aquele espaço! O processo de guardar o espaço da memória por meio de uma variável é chamado de declaração da variável. Para declarar uma variável, devemos seguir uma série de passos.

1° - Escolher um bom nome para a variável:
Quando declaramos uma variável, essa variável precisa ser identificada e, por isso, é importante darmos um nome a ela. Mas não podemos atribuir qualquer nome a uma

variável! Isso acontece porque na maioria das linguagens de programação não podemos nomear uma variável com o nome de um dos comandos primitivos da linguagem. Por exemplo, no Python, não podemos atribuir uma variável com o nome "print", pois esse nome já é usado pela linguagem. Também não é possível criar variáveis que comecem com números ou caracteres especiais, e por padrão não se usam letras maiúsculas no começo de variáveis. Não é permitido também o uso de espaços. Portanto, quando declaramos variáveis com mais de um nome, utilizamos um dos modos: ou declaramos todo o nome junto sem alterações, por exemplo a variável "notasdosalunos"; ou declaramos no sistema camelo, com as letras que iniciam palavras sendo maiúsculas, exemplo "notasDosAlunos"; podemos também utilizar o sistema de underline para separar as palavras, por exemplo "notas_de_alunos".

2° - Especificar a natureza da variável:
Em algumas linguagens de programação, é necessário que o programador informe ao programa que tipo de variável está declarando, pois cada tipo de variável ocupa um espaço diferente na memória do computador.

3° - Fazer a atribuição de um valor (=):
Em todas as linguagens de programação, quando declaramos uma variável, temos que iniciar ela com uma atribuição, mesmo que o valor atribuído seja nulo. A atribuição de valor na maioria das linguagens de programação costuma utilizar o sinal "=", que na programação chamamos de operador de atribuição. É ele que sinaliza para o computador abrir a "caixinha" da memória e guardar um determinado valor.

```
# Exemplo de declaração de variavel

nota_dos_alunos = 10
```

Devemos ter em mente que variáveis podem ser alteradas no decorrer do código e podem ser utilizadas dependendo do seu escopo (veremos isso mais adiante em nosso curso).

Por exemplo, qual o valor que o código a seguir irá imprimir?

```
nota_dos_alunos = 10
nota_dos_alunos = 0
print(nota_dos_alunos)
```

Se você disse 0, você está correto, pois a variável recebeu outro valor no decorrer do código, nesse caso, recebeu 0.

Regras para Criar uma Variável em Python

Ao criar uma variável em Python, é importante seguir algumas regras e boas práticas para garantir a clareza e a funcionalidade do código. Abaixo estão as principais regras:

Regras Básicas

1. **Nomes Válidos**:
   - Uma variável deve começar com uma letra (a-z, A-Z) ou um underscore (_).
   - Não pode começar com um número.
   - Pode conter letras, números e underscores (a-z, A-Z, 0-9, _).
2. **Palavras Reservadas**:
   - Não utilize palavras reservadas do Python (como `if`, `else`, `while`, `for`, etc.) como nomes de variáveis.
   - Você pode verificar as palavras reservadas com a biblioteca `keyword`:

     ```
     import keyword
     print(keyword.kwlist)
     ```
3. **Case Sensitivity**:
   - Python diferencia maiúsculas de minúsculas. Portanto, `variavel`, `Variavel` e `VARIAVEL` são consideradas variáveis distintas.
4. **Atribuição de Valores**:
   - Utilize o operador de atribuição `=` para definir uma variável.
   - Exemplo:

     ```
     x = 10
     nome = "João"
     ```

Boas Práticas

1. **Nomes Significativos**:
   - Use nomes de variáveis que descrevam claramente o que elas representam.
   - Exemplo:

     ```
     salario_mensal = 5000
     ```

2. **Convenção de Nomes**:
   - Siga a convenção de nomenclatura `snake_case` (letras minúsculas com palavras separadas por underscores) para variáveis.
   - Exemplo:

     ```
     numero_de_clientes = 150
     ```

3. **Constantes**:
   - Embora Python não tenha uma palavra-chave específica para constantes, por convenção, variáveis que não devem ser alteradas são nomeadas com letras maiúsculas.
   - Exemplo:

     ```
     PI = 3.14159
     ```

4. **Comentários**:
   - Adicione comentários explicativos quando necessário para descrever o propósito da variável.
   - Exemplo:

     ```
     # Armazena a idade do usuário
     idade = 25
     ```

Seguindo essas regras e práticas, você garantirá que suas variáveis sejam claras, compreensíveis e livres de erros comuns em Python.

Casting

Em python você também pode especificar o tipo de variável usando o conceito de casting .

```
nota_dos_alunos = int(10)
```

No código acima deixamos explícito que a variável será do tipo inteiro , nos casos que não declaramos o python vai tentar encontrar o melhor tipo que se enquadre com o dado, por padrão ele vai colocar o dado como str, int , float ou bool.

Casting, ou conversão de tipos, é um processo importante em programação Python onde transformamos o valor de uma variável de um tipo para outro. Isso é útil quando precisamos realizar operações entre tipos diferentes ou quando queremos garantir que o valor de uma variável seja interpretado corretamente.

Exemplos de Casting em Python

1. Casting de Inteiro para Float

```
# Exemplo 1: Convertendo um inteiro para float
num_int = 10
num_float = float(num_int)
print(num_float)  # Saída: 10.0
```

Neste exemplo, `num_int` é um número inteiro (10). Usando `float()`, convertemos esse inteiro em um número de ponto flutuante (`float`), armazenando-o em `num_float`.

2. Casting de Float para Inteiro

```
# Exemplo 2: Convertendo um float para inteiro
num_float = 10.75
num_int = int(num_float)
print(num_int)  # Saída: 10
```

Aqui, `num_float` é inicialmente um número de ponto flutuante (10.75). Ao usar `int()`, convertemos esse número para um inteiro, armazenando-o em `num_int`. Observe que a parte decimal é truncada ao realizar essa conversão.

3. Casting de String para Inteiro ou Float

```
# Exemplo 3a: Convertendo uma string para inteiro
num_str = "25"
num_int = int(num_str)
print(num_int)  # Saída: 25

# Exemplo 3b: Convertendo uma string para float
num_str = "3.14"
num_float = float(num_str)
print(num_float)  # Saída: 3.14
```

Nos exemplos acima, `num_str` é inicialmente uma string que representa um número (25 e "3.14", respectivamente). Utilizando `int()` e `float()`, convertemos essas strings para tipos numéricos (`int` e `float`), permitindo operações matemáticas ou armazenamento em variáveis com o tipo correto.

4. Conversão entre Tipos Incompatíveis

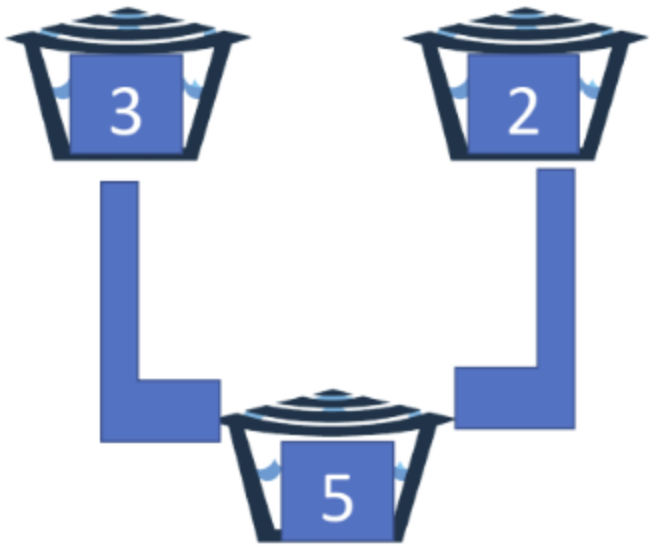
```
# Exemplo 4: Conversão entre tipos incompatíveis
valor = "123"
```

```
numero = int(valor)
print(numero)  # Saída: 123

texto = "Python"
# linha a seguir gera um erro
numero = int(texto)
```

3.3 Operadores

Agora que sabemos como o computador guarda os dados nas suas memórias, iremos aprender como os computadores são capazes de realizar contas. O computador funciona basicamente como um sistema de caixas d'água, tá eu sei não faz sentido, vamos lá!
Pense que queremos somar o número 2 com o número 3, as caixas cheias abaixo simbolizam esses números:



Na operação de soma, os canos transportam água para uma caixa maior que, quando cheia, tem o valor 5. Basicamente, o computador funciona de maneira semelhante. O que o programador faz é simplesmente dizer quais caixas o computador deve acessar e quais torneiras deve ligar para realizar as operações matemáticas.

Na programação, o que o computador entende como torneiras são os OPERADORES. Os operadores são basicamente os mesmos na maioria das linguagens de programação. Existem diferentes tipos de operações que os computadores são capazes de realizar. Vamos conhecer alguns deles na linguagem Python.

> Vejamos abaixo os operadores presentes no python

Operadores aritméticos

São os operadores utilizados para realizar operações matemáticas

Operador	Nome	Exemplo	Exemplo de Uso
+	Adição	x + y	`5 + 3`
-	Subtração	x - y	`7 - 2`
*	Multiplicação	x * y	`4 * 6`
/	Divisão	x / y	`8 / 2`
%	Módulo	x % y	`9 % 4`
**	Exponenciação	x ** y	`2 ** 3`
//	Divisão Inteira	x // y	`10 // 3`

Operadores de comparação

Os operadores de comparação em Python são utilizados para comparar dois valores na memória e retornam um resultado booleano: `True` (verdadeiro) se a comparação for verdadeira, ou `False` (falso) se a comparação for falsa. Aqui estão os operadores de comparação e exemplos de como são usados:

Operador	Nome	Exemplo	Exemplo de Uso
==	Igual	x == y	`5 == 3`
!=	Diferente	x != y	`7 != 2`
>	Maior que	x > y	`4 > 6`
<	Menor que	x < y	`8 < 2`
>=	Maior ou igual a	x >= y	`9 >= 4`

<=	Menor ou igual a	x <= y	`2 <= 3`

Operadores lógicos

São operadores utilizados para realizar operações de lógica booleana.

Operador	Descrição	Exemplo	Exemplo de Uso
and	Retorna True se ambas as condições forem verdadeiras	`x < 5 and x < 10`	`2 < 5 and 2 < 10`
or	Retorna True se pelo menos uma condição for verdadeira	`x < 5 or x < 4`	`2 < 5 or 2 < 4`
not	Inverte o resultado, retorna False se o resultado for verdadeiro	`not(x < 5 and x < 10)`	`not(2 < 5 and 2 < 10)`

Operadores de identidade

Verificam se espaços da memoria são identicamente iguais , não apenas numericamente.

Operador	Descrição	Exemplo	Exemplo de Uso
is	Retorna True se ambas as variáveis forem o mesmo objeto	`x is y`	`lista1 is lista2`
is not	Retorna True se ambas as variáveis não forem o mesmo objeto	`x is not y`	`lista1 is not lista2`

Operadores de associação (membership)

São utilizados para verificar se um valor específico está presente em um objeto, como uma string, lista, tupla, ou outro tipo de coleção.

Operador	Descrição	Exemplo	Exemplo de Uso
in	Retorna True se uma sequência com o valor especificado está presente no objeto	`x in y`	`'a' in 'banana'`
not in	Retorna True se uma sequência com o valor especificado não está presente no objeto	`x not in y`	`'b' not in 'banana'`

Operadores de atribuição

Operador	Exemplo	Equivalente a	Exemplo de Uso
=	x = 5	x = 5	`x = 5`
+=	x += 3	x = x + 3	`x += 3`
-=	x -= 3	x = x - 3	`x -= 3`
*=	x *= 3	x = x * 3	`x *= 3`
/=	x /= 3	x = x / 3	`x /= 3`
%=	x %= 3	x = x % 3	`x %= 3`
//=	x //= 3	x = x // 3	`x //= 3`
**=	x **= 3	x = x ** 3	`x **= 3`
&=	x &= 3	x = x & 3	`x &= 3`
^=	x ^= 3	x = x ^ 3	`x ^= 3`
>>=	x >>= 3	x = x >> 3	`x >>= 3`
<<=	x <<= 3	x = x << 3	`x <<= 3`
:=	print(x := 3)	x = 3	`print(x := 3) print(x)`

Operadores de bits

Operador	Nome	Descrição	Exemplo	Exemplo de Uso
&	AND	Define cada bit como 1 se ambos os bits forem 1	`x & y`	`5 & 3`
|	OR	Define cada bit como 1 se pelo menos um dos dois bits for 1	`x | y`	`5 | 3`
^	XOR	Define cada bit como 1 se apenas um dos dois bits for 1	`x ^ y`	`5 ^ 3`
~	NOT	Inverte todos os bits	`~x`	`~5`
<<	Left Shift	Deslocamento para a esquerda preenchendo com zeros	`x << 2`	`5 << 2`

>>	Right Shift	Deslocamento para a direita com preservação de sinal	`x >> 2`	`5 >> 2`

Precedência de Operadores em Python

1. Operadores Aritméticos

Os operadores aritméticos são usados para realizar operações matemáticas básicas. Aqui está a ordem de precedência para os operadores aritméticos:

1. **Parênteses** têm a maior precedência e são usados para forçar a avaliação de expressões.

Exemplo:

```
resultado = (5 + 3) * 2
print(resultado)  # Saída: 16
```

Neste exemplo, a expressão dentro dos parênteses `(5 + 3)` é avaliada primeiro, resultando em 8. Em seguida, esse resultado é multiplicado por 2.

1. **Exponenciação (`*`)** tem a segunda maior precedência.

Exemplo:

```
resultado = 2 + 3 ** 2
print(resultado)  # Saída: 11
```

Neste caso, primeiro `3 ** 2` é calculado (9), e depois adicionado a 2, resultando em 11.

1. **Multiplicação ( ), Divisão (`/`), e Módulo (`%`)** têm a mesma precedência e são avaliados da esquerda para a direita.

Exemplo:

```
resultado = 10 * 2 / 5
```

```
print(resultado)  # Saída: 4.0
```

Aqui, `10 * 2` é calculado primeiro (20), e depois dividido por 5, resultando em 4.0.

1. **Adição (`+`) e Subtração (`-`)** têm a menor precedência entre os operadores aritméticos e também são avaliados da esquerda para a direita.

Exemplo:

```
resultado = 10 - 2 + 3
print(resultado)  # Saída: 11
```

Primeiro `10 - 2` é calculado (8), e depois adicionado 3, resultando em 11.

2. Operadores de Comparação

Os operadores de comparação são usados para comparar valores. Eles têm precedência menor do que os operadores aritméticos. Aqui está a ordem de precedência para os operadores de comparação:

1. Todos os operadores de comparação (`==`, `!=`, `>`, `<`, `>=`, `<=`) têm a mesma precedência e são avaliados da esquerda para a direita.

Exemplo:

```
resultado = 5 < 3 + 2
print(resultado)  # Saída: False
```

Primeiro `3 + 2` é calculado (5), e depois comparado se 5 é menor que 5, o que retorna `False`.

3. Operadores Lógicos

Os operadores lógicos são usados para combinar condições booleanas. Eles têm precedência menor do que os operadores de comparação. Aqui está a ordem de precedência para os operadores lógicos:

1. `not` tem a maior precedência entre os operadores lógicos.
2. `and` tem precedência mais alta do que `or`.

Exemplo:

```
resultado = 5 > 3 and 2 < 4
print(resultado)  # Saída: True
```

Neste caso, `5 > 3` é avaliado como verdadeiro (True) e `2 < 4` também é avaliado como verdadeiro (True), então `True and True` resulta em `True`.

Utilizando Precedência para Controle

É importante compreender a precedência dos operadores para garantir que as expressões sejam avaliadas na ordem desejada. Utilize parênteses para clarificar a ordem de avaliação e evitar ambiguidades.

Exemplo:

```
resultado = (5 > 3) and (2 + 4 > 5)
print(resultado)  # Saída: True
```

Neste exemplo, os parênteses são usados para garantir que `5 > 3` e `2 + 4 > 5` sejam avaliados separadamente e depois combinados com o operador `and`.