Bultin-in e estruturas definidas pelo usuário

Pense nas estruturas de dados como diferentes caixas de ferramentas. Cada caixa tem ferramentas (ou formas) especiais para guardar e usar informações em computadores. Assim como você escolhe a ferramenta certa para um trabalho, você também precisa escolher a caixa de ferramentas certa – ou seja, a estrutura de dados certa – para cada tipo de problema no mundo da computação.

As estruturas de dados são como mapas que mostram a melhor maneira de organizar e acessar essas informações. Escolher a estrutura de dados certa é como escolher o melhor caminho em um mapa, para que você possa guardar e encontrar as informações de forma rápida e fácil.

Sequências: listas, tuplas, dicionários e conjuntos

Agora, pense na sua lista de compras. Se você pegar os itens na ordem que escreveu, isso é como um tipo de lista chamado FIFO (*First-In, First-Out*), que significa "o primeiro que entra é o primeiro que sai". É como uma fila em um banco: a primeira pessoa na fila é a primeira atendida.

Mas e se você decidir começar pelas últimas coisas que escreveu? Isso é chamado LIFO (*Last-In, First-Out*), ou seja, "o último que entra é o primeiro que sai". É como uma pilha de pratos: você sempre pega o prato de cima.

Armazenamento de dados

Na ciência da computação, os fundamentos operacionais envolvem três processos essenciais: entrada, processamento e saída de dados. Esses processos são realizados por meio da integração sinérgica entre componentes de hardware e software. A metodologia de armazenamento de dados desempenha um papel essencial nesse contexto. O bit, acrônimo de *'binary digit'* (dígito binário), representa a unidade primordial de armazenamento na computação.

Conceitualmente, um bit pode ser considerado como a representação mais elementar de um estado em um dispositivo eletrônico, funcionando análogo a uma chave que se encontra em um dos dois estados mutuamente exclusivos: ligado ou desligado, aberto ou fechado. Esse estado binário é representado pelos valores 0 ou 1, estabelecendo a base do sistema binário, que é fundamental na arquitetura e no funcionamento dos sistemas computacionais contemporâneos.

Dentro da arquitetura de sistemas computacionais, o sistema de numeração binário opera com bytes como uma unidade fundamental. Um byte é composto por uma sequência de oito bits, possibilitando a representação de 256 valores distintos, conforme ilustrado pela expressão:

2^8 (isto é, 2 x 2 x 2 x 2 x 2 x 2 x 2 x 2)

Esse conceito de byte é fundamental para entender como os computadores armazenam uma variedade de dados, transcendendo a mera representação numérica.

Além de números, é necessário que os computadores sejam capazes de armazenar caracteres alfanuméricos e símbolos. Consequentemente, foram estabelecidos padrões para a representação binária de *strings* de caracteres. Cada caractere é

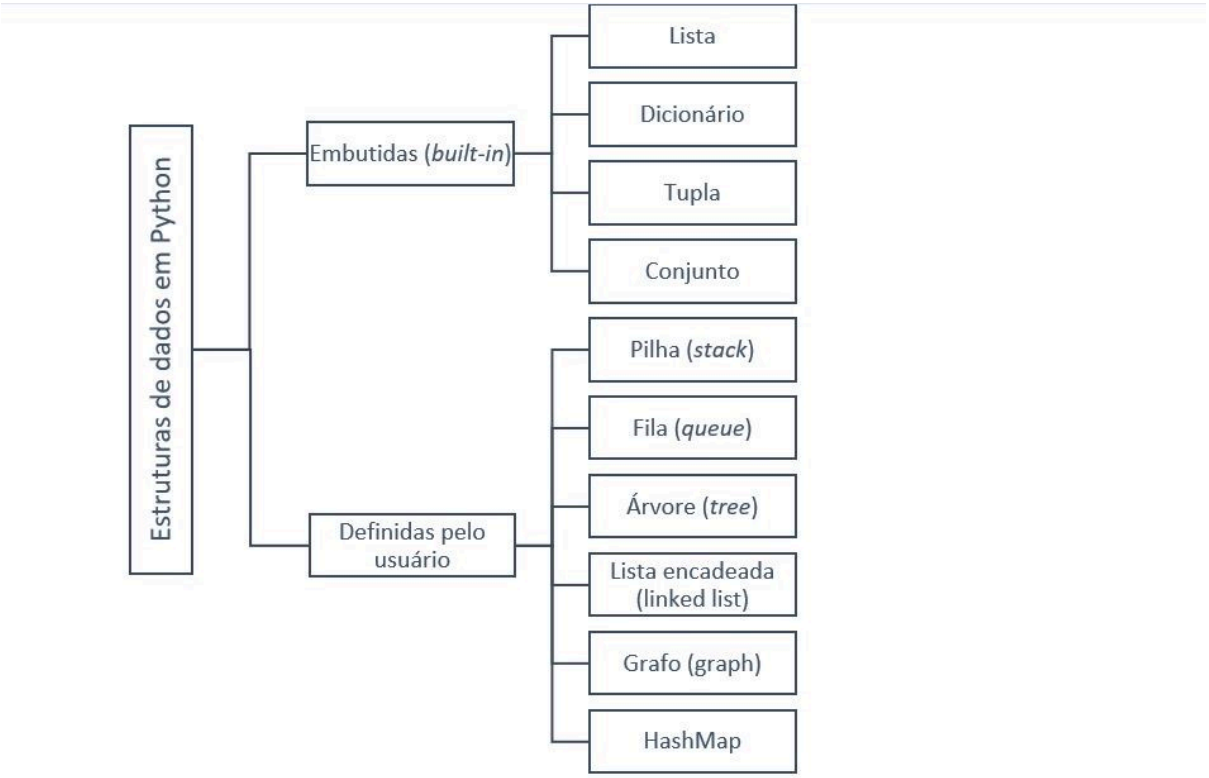
mapeado para uma sequência específica de bits. Por exemplo, nos padrões de codificação adotados por alguns modelos de computadores antigos, o caractere 'A' poderia ser representado pela sequência binária '11000000', enquanto o 'B' seria representado por '11000001'. Portanto, a palavra "AB" seria representada na forma binária como '1100000011000001'. Esses padrões de codificação são essenciais para a representação e o processamento de informações textuais em sistemas computacionais

A problemática central reside na distinção entre a representação de caracteres alfabéticos, como a letra 'A', e a representação numérica em um sistema binário, como o número 192, ambos podendo ser representados pela mesma sequência binária '11000000'. A solução para esse dilema encontra-se na especificação dos tipos de dados no contexto de linguagens de programação. Linguagens de alto nível facilitam a definição do tipo de dado a ser armazenado, seja ele um número inteiro, um decimal, um conjunto de caracteres (como palavras), uma sequência de elementos, ou até mesmo um valor booleano que assume estados binários (verdadeiro ou falso, 1 ou 0, etc.).
Em linguagens de programação de alto nível de gerações anteriores, como a linguagem C, a tipificação das variáveis é uma exigência explícita na sua declaração. Essa abordagem assegura que a interpretação dos dados armazenados seja realizada de forma coerente com o tipo de dado especificado, permitindo assim que a sequência binária seja corretamente interpretada conforme o contexto de sua aplicação, seja como um caractere ou como um valor numérico.

Parte superior da fórmula:

```
int x, y;
float a, b;
```

O exemplo citado aloca espaço na memória do computador para duas variáveis inteiras (x e y) e duas variáveis de ponto flutuante (a e b). Para armazenar uma palavra, usa-se o tipo 'char', especificando-se o número máximo de caracteres permitidos para essa variável.
Para uma gestão eficiente da informação, são necessárias estruturas de dados robustas. Essas estruturas, fundamentais na ciência da computação, permitem a organização, manipulação e utilização eficaz das informações. Em Python, as estruturas de dados se dividem em duas categorias principais: estruturas embutidas (*built-in* ou nativas) e estruturas definidas pelo usuário, que exigem implementação específica pelo programador.



Python possui, de forma nativa, diversas estruturas de dados do tipo embutido (*built-in*), frequentemente chamadas de estruturas de dados sequenciais ou simplesmente sequências.

Sequências

No Python, bem como em outras linguagens de programação, as sequências são utilizadas para armazenar coleções de dados em um único objeto. Esses dados podem ser homogêneos (do mesmo tipo) ou heterogêneos (de tipos variados). Entre os tipos mais comuns de sequências estão listas, dicionários, tuplas e conjuntos

Tipo de variável	Característica	Exemplo de declaração (variável v)
lista	Lista de elementos. Pode receber elementos de tipos distintos, como strings, números inteiros, booleanos ou até mesmo outras listas, tuplas e dicionários.	v = [“abacate”, “banana”, “cenoura”]
tuple	Uma tupla pode ser considerada uma lista imutável de elementos. Uma vez criada, não	v = (1, 2, 3)

	pode ser alterada.	
dictionary	Um dicionário pode ser interpretado como lista composta por conjuntos de pares chave-valor.	v = { “A” : “adenina”, “C” : “citosina”, “T” : “timina”, “G” : “guanina” }
set	Conjuntos de elementos únicos.	v = {1, 2, 3, 4, 5, 5, 5, 5}

No Python, *strings* são consideradas sequências, pois permitem o acesso a caracteres individuais. As sequências podem ser mutáveis, como listas e *arrays*, permitindo alterações, ou imutáveis, como *strings* e tuplas, que não mudam após a criação. Existem também sequências simples, que contêm itens de um tipo, e sequências contêiner, que abrigam itens de vários tipos. Sequências são úteis em programação para várias finalidades, desde listar itens similares até associar termos a definições em dicionários. Em campos específicos, como bioinformática e finanças, são usadas para mapear informações genéticas ou armazenar cotações, respectivamente. Sequências imutáveis são empregadas para representar valores constantes, como conversões de unidades.

Listas

Em Python, as listas representam uma estrutura de dados fundamental caracterizada por sua natureza ordenada e mutável. Essas estruturas são definidas pela capacidade de armazenar uma coleção de elementos, que podem ser de tipos variados, incluindo números, *strings* e até outras listas, em uma única variável. A ordenação das listas implica que cada elemento é associado a um índice, começando do zero, permitindo assim o acesso e a manipulação eficiente dos dados.

Tuplas

Uma tupla em Python, conhecida como tuple, é uma coleção ordenada de elementos que é imutável, ou seja, não pode ser alterada após sua criação. Os elementos de uma tupla são identificados e acessados através de suas posições. As tuplas podem ser criadas colocando os elementos entre parênteses ou simplesmente separando-os com vírgulas.

Dicionários

Dicionários em Python, conhecidos como *dict*, são estruturas de dados que armazenam pares de chave-valor. Eles são coleções não ordenadas, mas são mutáveis, permitindo que os dados sejam modificados após a criação. Cada chave no dicionário é única e é usada para acessar o valor correspondente. Dicionários são ideais para casos em que as relações entre os dados são essenciais.

Conjuntos

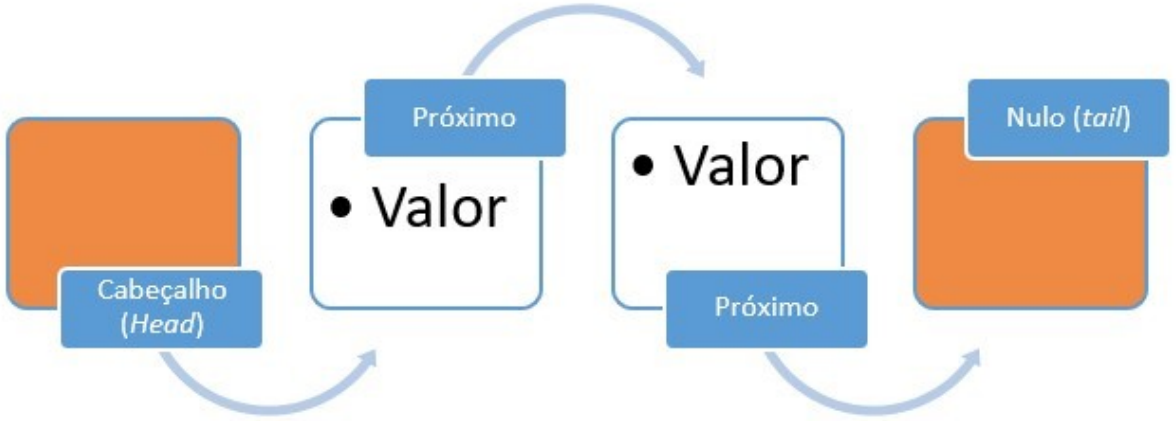
Conjuntos em Python, conhecidos como set, são coleções que armazenam elementos únicos e não ordenados. Diferente das listas e tuplas, os conjuntos não

permitem valores duplicados, e a ordem dos elementos não é garantida. Eles são particularmente úteis para operações como união, interseção e diferença, comuns em lógica de conjuntos matemáticos.

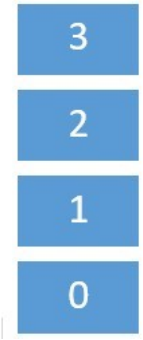
Tipos de dados em python

Python distingue-se de outras linguagens de programação por sua sintaxe intuitiva e uma ampla gama de funcionalidades pré-construídas, incluindo várias estruturas de dados. No entanto, a linguagem também permite que os usuários criem suas próprias estruturas de dados personalizadas.
Conforme apontado por Alves (2021), existem seis tipos principais de estruturas de dados definidas pelo usuário que podem ser implementadas em Python. Essas estruturas incluem listas encadeadas, pilhas, filas, árvores, grafos e *hashmap*. Nas próximas aulas, esses conceitos serão detalhadamente introduzidos e discutidos, fornecendo uma base sólida para o entendimento e a aplicação dessas estruturas ao longo do livro.
Listas encadeadas, pilhas, filas, árvores e grafos são estruturas de dados fundamentais na computação, cada uma com suas características e usos específicos.

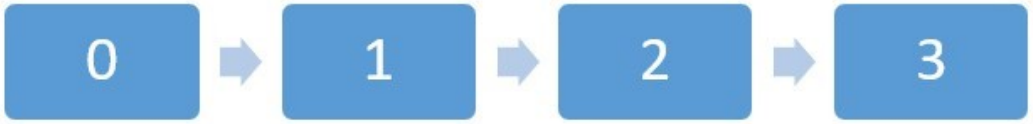
1. **Listas encadeadas:** são coleções de elementos chamados nós, onde cada nó está conectado sequencialmente ao próximo por meio de "ponteiros". Eles oferecem flexibilidade na inserção e remoção de elementos, já que não exigem realocação de outros elementos, ao contrário dos *arrays*.



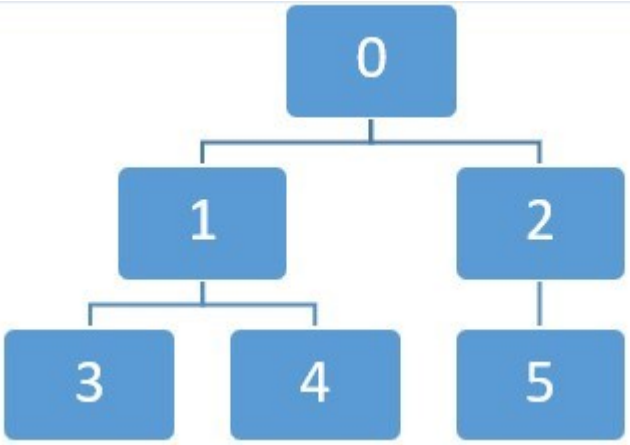
2. **Pilhas:** funcionam com o princípio de "último a entrar, primeiro a sair" (LIFO). São usadas em diversas aplicações, como na execução de algoritmos de busca em profundidade e na gestão de tarefas em sistemas operacionais.



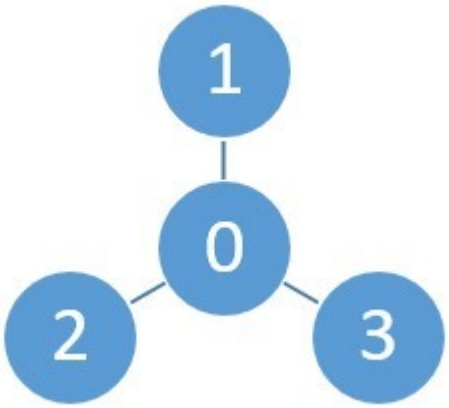
3. **Filas:** operam no princípio de "primeiro a entrar, primeiro a sair" (FIFO). São úteis em situações que exigem processamento sequencial, como na gestão de filas de impressão ou em algoritmos de busca em largura.



4. **Árvores:** estruturas hierárquicas que começam com um nó raiz, de onde se ramificam nós filhos. São cruciais em aplicações como a organização de sistemas de arquivos, implementação de bases de dados e algoritmos de busca.

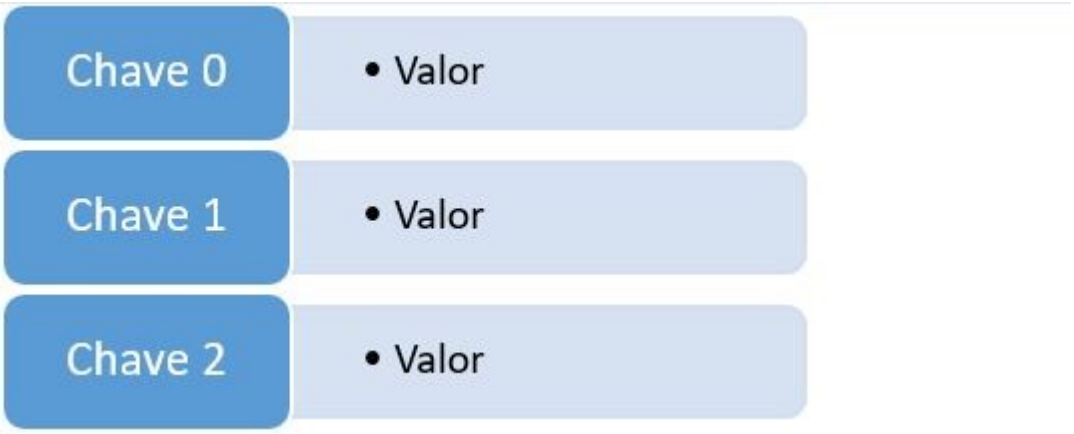


5. **Grafos:** consistem em nós (ou vértices) que podem ser conectados por arestas. Essenciais para representar relações complexas, os grafos são amplamente utilizados em redes sociais, sistemas de mapas e algoritmos de roteamento.



6. ***Hashmap*:** também conhecido como tabela de dispersão, o *hashmap* é uma estrutura de dados que armazena pares de chave-valor. Ele oferece acesso extremamente rápido aos elementos, pois utiliza uma função de *hash* para converter a chave em um índice, onde o valor correspondente é armazenado. *Hashmaps* são ideais para situações onde é necessário um acesso rápido aos dados, como em caches, bases de dados e na implementação de conjuntos. Eles são eficientes tanto

para inserção quanto para busca e remoção de elementos, tornando-os uma ferramenta poderosa para manipulação de dados em larga escala.

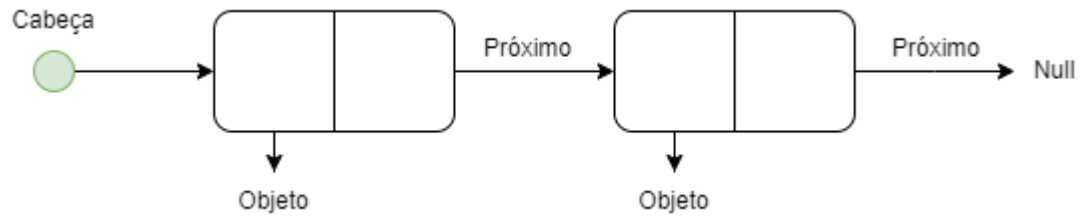


Listas encadeadas ou ligadas

No âmbito das estruturas de dados, uma lista é uma entidade que permite o armazenamento e a organização de dados de maneira sequencial. Em Python, por exemplo, a implementação intrínseca de uma lista pode ser realizada através da função ***list()*** ou mediante a inclusão de elementos entre colchetes. Essa linguagem de programação oferece uma série de métodos nativos para a manipulação de listas que facilitam operações como a inserção, atualização e remoção de elementos (Lambert, 2022).

Apesar da simplicidade proporcionada pelas estruturas de dados nativas de Python, existem variantes mais complexas de listas que não são intrínsecas à linguagem, como as listas encadeadas, também conhecidas como listas ligadas: imagine uma fila de pessoas onde cada pessoa segura um papel com seu próprio nome e uma seta apontando para a próxima pessoa na fila. Em computação, temos algo parecido chamado lista encadeada. Cada pessoa é como um "nó" que sabe quem vem depois dela na fila. Em uma lista encadeada, cada nó tem uma informação e um ponteiro que indica quem é o próximo nó. Se precisarmos tirar alguém do meio da fila, como a pessoa número 56 de uma fila de 100, não precisamos reorganizar toda a fila. Em vez disso, apenas dizemos para a pessoa número 55 que agora a pessoa número 57 está depois dela. É uma mudança simples que economiza muito tempo, pois não precisamos atualizar a posição de cada pessoa na fila.

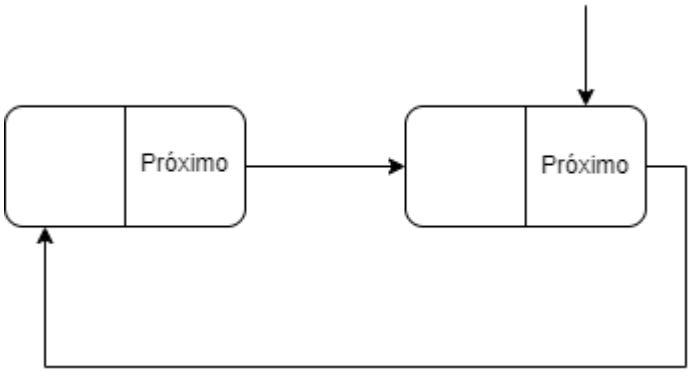
Estruturas definidas dessa forma exigem uma implementação manual para sua utilização, dada a necessidade de funcionalidades específicas que transcendem as capacidades das listas padrão (Szwarcfiter; Markenzon, 2020). Parte superior do formulário.



Nas listas encadeadas, também conhecidas como listas ligadas, os dados são armazenados em nós. Cada nó contém um valor e um ponteiro para o próximo nó na lista. Dessa forma, a posição de um item não depende de toda a lista, mas sim

do item que o segue. Para navegar pela lista encadeada, começa-se pelo primeiro elemento, chamado de cabeça (*head*), seguindo os ponteiros de um nó para o outro até alcançar um valor nulo, que indica o fim da lista, conhecido como cauda (*tail*).

Adicionalmente, as listas encadeadas podem ser configuradas de modo que não tenham um "último item" definido. Nessa variação, o nó que seria o último na lista aponta de volta para o primeiro nó, criando um ciclo contínuo. Esse tipo de estrutura é conhecido como **lista circular** (Alves, 2021).

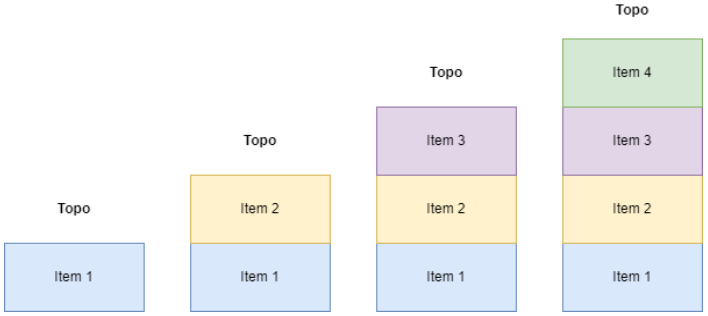


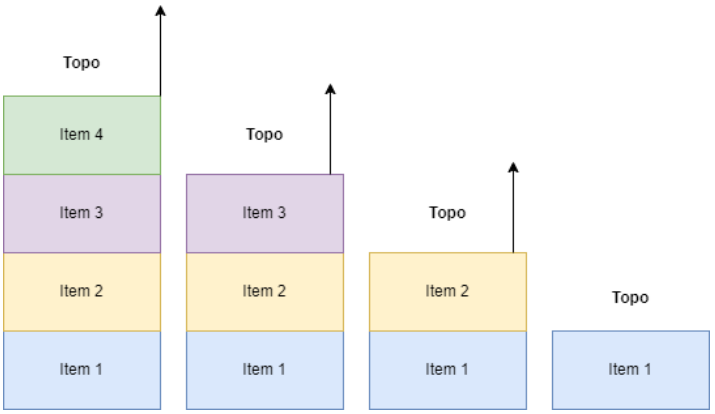
Para complementar seus estudos, acesse os artigos abaixo:

1. Introdução a listas encadeadas;
2. Implementação e busca em listas encadeadas em Python;
3. Inserção e deleção de itens.

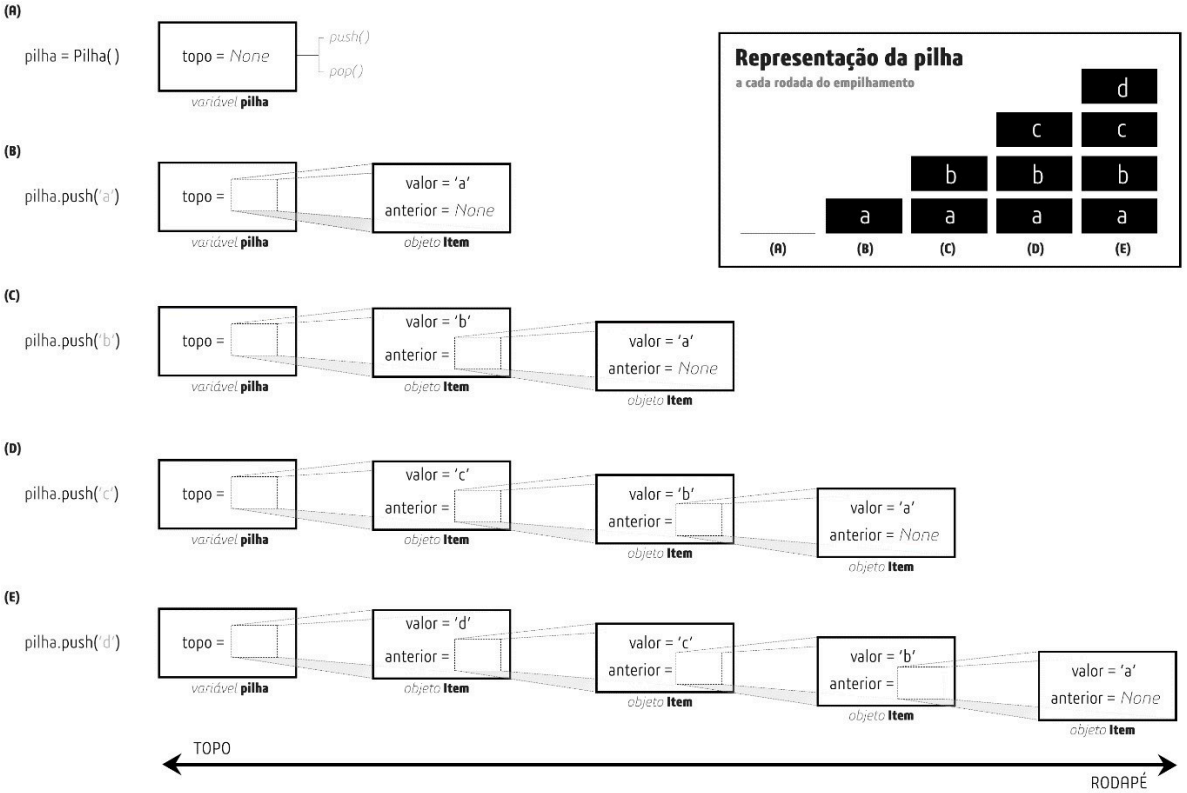
Pilhas

Na ciência da computação, a estrutura conhecida como pilha (ou *stack* em inglês) é uma forma de armazenar dados que segue a política de "último a entrar, primeiro a sair" (LIFO - *Last In, First Out*). Essa estrutura permite que os itens sejam adicionados em sequência, mas a remoção ocorre a partir do item mais recentemente adicionado, isto é, do topo da pilha. Conforme descrito por Lambert (2022), a pilha é uma implementação de lista onde o último elemento inserido é o primeiro a ser removido. Um exemplo prático e ilustrativo dessa estrutura pode ser observado em uma pilha física de pratos ou livros, onde o item no topo é sempre o primeiro a ser removido, quando se deseja transferir itens de uma pilha para outra. Dentro do contexto acadêmico da ciência da computação, as pilhas são estruturas de dados que suportam duas operações fundamentais: o **empilhamento**, conhecido também como operação *'push'*, que consiste na adição de um item à pilha, e o **desempilhamento**, referido como operação 'pop', que envolve a remoção de um item da pilha. Essa descrição é corroborada pelos trabalhos de Lambert (2022). A natureza dessas operações implica que, ao adicionar elementos a uma pilha, estes são dispostos uns sobre os outros, seguindo a sequência de inserção.

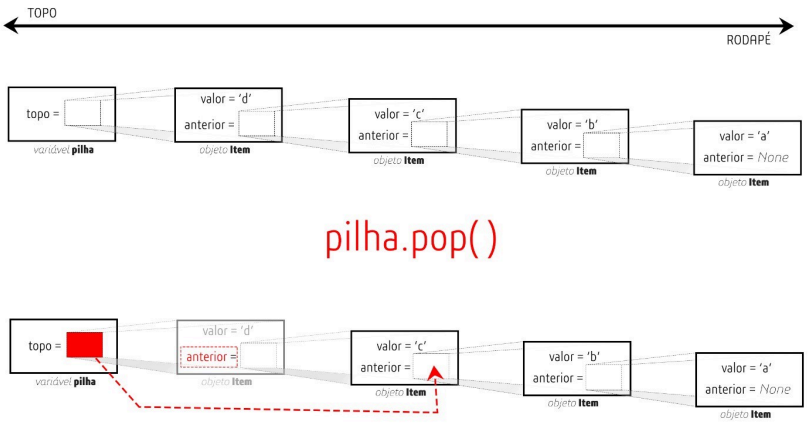




Conceito de pilha com lista encadeada:

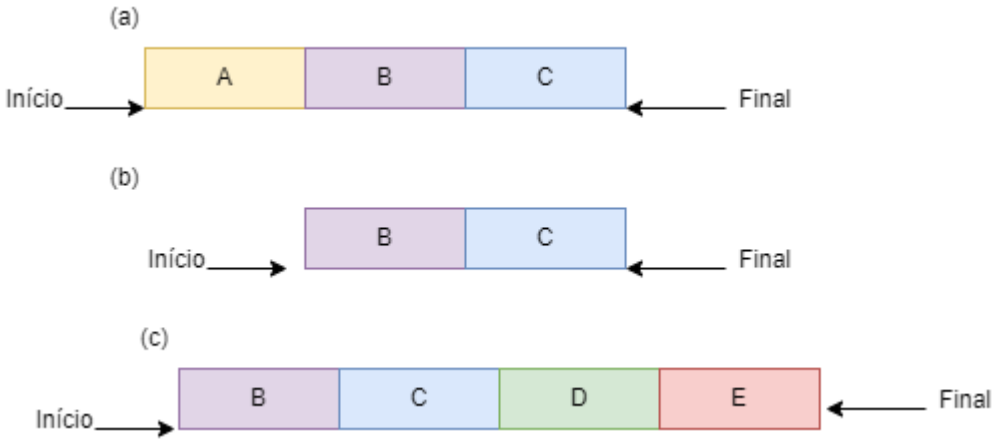


Exemplo da função pop():



Fila

Nos estudos de estruturas de dados, uma fila é frequentemente descrita como uma estrutura do tipo FIFO (*First-In, First-Out*), que se traduz como "primeiro a entrar, primeiro a sair" (Alves, 2021). Nessa estrutura, o elemento que é adicionado primeiro na fila será também o primeiro a ser removido e, consequentemente, o elemento adicionado por último será o último a sair. A figura ilustrativa a seguir exibe uma fila composta por três elementos: A, B e C. Nesse caso, o elemento A encontra-se na frente da fila, enquanto o C está na posição final. Durante uma operação de remoção, o elemento na frente da fila (A) é o primeiro a ser removido. Inversamente, quando novos elementos (como D e E) são adicionados à fila, eles são posicionados sequencialmente após o último elemento existente na fila.



Introdução a filas e suas operações

- **Algoritmos em Python**

Introdução à estrutura de dados pilhas

- **Algoritmos em Python**

Pilhas baseadas em listas encadeadas

- Estruturas de dados: listas, filas, pilhas, conjuntos, árvores e hash tables

A escolha apropriada de uma estrutura de dados pode influenciar significativamente o desempenho de um algoritmo ou aplicativo, afetando diretamente a eficiência, a rapidez e até a viabilidade de operações computacionais específicas (Lambert, 2022).
Por exemplo, as listas encadeadas são essenciais quando se necessita de inserções e exclusões eficientes em qualquer posição da lista, sem a necessidade de realocar a memória, como é o caso em *arrays* (Backes, 2023). As árvores, por outro lado, são extremamente úteis para operações de busca e ordenação de dados, como demonstrado nas árvores binárias de busca, que permitem inserções, exclusões e buscas em tempo proporcional à altura da árvore.
Além disso, as pilhas e filas são amplamente utilizadas para controlar o fluxo de execução em programas, com as pilhas apoiando a implementação de chamadas de função e as filas auxiliando na gestão de tarefas em sistemas operacionais e aplicativos de servidor (Backes, 2023). De forma similar, as tabelas de *hash* são empregadas para buscas rápidas e eficientes, sendo um componente crítico em muitos algoritmos de indexação e banco de dados (Lambert, 2022).

A escolha de uma estrutura de dados apropriada não é apenas uma questão de desempenho, mas também de conveniência e clareza. Estruturas de dados adequadas podem simplificar significativamente a implementação de um algoritmo e tornar o código mais compreensível e manutenível.
Por exemplo, ao lidar com grandes volumes de dados, a utilização de árvores de busca balanceadas, como AVL ou Red-Black Trees, oferece uma vantagem significativa sobre listas não ordenadas, principalmente em operações de busca, inserção e deleção, mantendo o tempo de execução proporcional ao logaritmo do número de elementos (Szwarcfiter; Markenzon, 2020). Em contextos onde as buscas frequentes e a inserção rápida são essenciais, as tabelas de *hash* demonstram ser extremamente eficientes, fornecendo, na maioria dos casos, uma busca em tempo constante (O(1)) (Backes, 2023).
Outro exemplo notável é a aplicação de pilhas e filas. Pilhas são fundamentais na implementação de algoritmos de *backtracking* e na realização de chamadas recursivas, enquanto filas são essenciais em algoritmos de busca em largura e na gestão de *buffers* em sistemas operacionais (Alves, 2021). A seleção entre uma pilha ou uma fila pode impactar diretamente a maneira como os dados são acessados e processados, influenciando a eficiência do algoritmo. Filas, em particular, desempenham um papel significativo na otimização de processos em ambientes de computação em nuvem, facilitando o gerenciamento eficiente de carga de trabalho e processamento de dados.
Um exemplo bastante interessante é o uso de Filas na AWS (Amazon Web Services), especialmente implementadas através do Amazon Simple Queue Service (SQS). O SQS oferece uma solução de fila de mensagens escalável, que permite aos desenvolvedores desacoplar e escalar microsserviços, sistemas distribuídos e aplicações *serverless* (AWS Amazon, 2023).

Para ilustrar esse conceito, considere o exemplo de um sistema de reserva de bilhetes on-line. Suponha que você esteja desenvolvendo um sistema de reserva de bilhetes para um cinema e que esse sistema precisa lidar com um grande número de consultas de clientes e processar reservas de assentos em tempo real.
Qual tipo de estrutura de dados você poderia utilizar na construção de uma possível solução?

1. **Uso de *arrays* ou matrizes:** para gerenciar os assentos do cinema, você pode usar uma matriz bidimensional, onde cada elemento representa um assento específico no cinema (por exemplo, assentos[fila][coluna]). Essa estrutura de dados permite acessar rapidamente o estado de qualquer assento, verificando se ele está disponível ou reservado.

2. **Filas para gerenciar reservas:** uma fila pode ser usada para gerenciar as solicitações de reserva. Como as filas seguem a política FIFO (*First-In, First-Out*), elas garantem que as reservas sejam processadas na ordem em que são recebidas, mantendo a justiça e a eficiência no processamento.
3. **Árvores para pesquisas rápidas:** caso o sistema permita aos usuários escolher assentos com base em preferências específicas (por exemplo, proximidade da tela, acesso para cadeirantes), uma árvore de busca, como uma árvore binária de busca balanceada, pode ser usada para organizar os assentos. Isso permite buscas rápidas e eficientes para encontrar assentos que atendam aos critérios dos usuários.

Por exemplo, árvores B e árvores B+ são amplamente utilizadas em sistemas de gerenciamento de banco de dados para indexação. Essas estruturas permitem inserções, buscas e exclusões eficientes, mesmo em grandes conjuntos de dados, o que é importante para o desempenho de operações de consulta e atualização em bancos de dados relacionais. As árvores B+ são particularmente eficazes para buscas intervalares e varreduras sequenciais, pois todos os valores estão nos nós folha e há um encadeamento entre eles, facilitando operações sequenciais (Pereira, 2016).

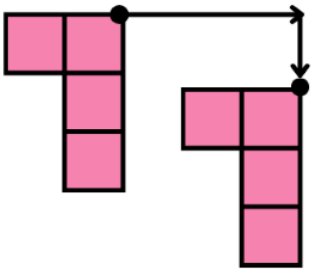
Além disso, as estruturas de grafos são amplamente utilizadas na modelagem e gestão de dados complexos e relacionais, particularmente em aplicações como redes sociais, sistemas de recomendação e análise de redes. Grafos permitem representar relações complexas entre entidades de maneira intuitiva e eficaz, e algoritmos baseados em grafos, como busca em profundidade e largura, são fundamentais para navegar e analisar essas relações (Lambert, 2022).

Tabelas *hash*, por outro lado, são usadas para garantir acessos rápidos a dados em operações de busca, sendo essenciais em cenários onde o tempo de resposta é um fator crítico. A estrutura de tabelas *hash*, com sua capacidade de fornecer acesso de tempo quase constante, é ideal para armazenamento e recuperação rápida de dados, especialmente em aplicações de alta frequência, como caches de sistemas web (Pereira, 2016).

FUNDAMENTOS DE ESTRUTURAS DE DADOS

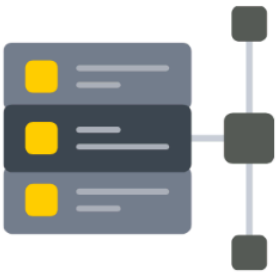
INTRODUÇÃO A ESTRUTURA DE DADOS EM PYTHON

Visão geral das estruturas de dados fundamentais disponíveis em Python



LISTAS ENCADEADAS

Uma forma de estrutura de dados onde cada elemento aponta para o próximo, facilitando a inserção e remoção de elementos sem reorganização da estrutura de dados inteira.



PILHAS E FILAS

Pilhas (LIFO - Last In, First Out) e **Filas** (FIFO - First In, First Out), estruturas de dados especializadas que gerenciam elementos em ordem específica, adequadas para certos algoritmos e aplicações.