**INTRODUÇÃO 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 caracteristicas 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 flexivel), 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.

**O que é um 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 em 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:**

Tela de computador com letras e números em fundo preto

Descrição gerada automaticamente

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:**

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

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.

**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.

Tela preta com letras brancas

Descrição gerada automaticamente

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.

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

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.

Texto

Descrição gerada automaticamente

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 “Voc|ê 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’.

Texto

Descrição gerada automaticamente

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 cerifica 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’.

Tela de celular com aplicativo aberto

Descrição gerada automaticamente

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, reprensatada 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 ambam 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.

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

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 imprimira “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 é 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 quaisquer 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.

**Estruturas 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:

**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.

**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.

**Algoritmos de Busca**

“Um algoritmos 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.

**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.

**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.

**Algoritmos 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:

**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.

**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.

**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.

**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.

**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.

**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.

**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.

**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.

**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.

**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.

**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.

**Referências**

TEIXEIRA, Douglas. “Estrutura de dados”, 2017.

EDISCIPLINAS. “Introdução aos algoritmos de busca em Python”, 2023.

PANDAIME. “Busca sequencial”, 2019.

FREECODECAMP. “Algoritmo de pesquisa binária”, 2022.

KRIGER, Daniel. “Lista em Python”, 2022.

ALURA. “Tupla no Python”, 2023.

HASHTAGTREINAMENTOS, “Dicionário Python”, 2023.

PINHEIRO, Fagner. “Manipulando sets no Python”, 2020.

**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.

**Bibliotecas, 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.

**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.

**Aplicação 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.

**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)*

**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

**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

**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.

**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

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

**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.

**Referências**

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

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

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

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

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

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

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

<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>.

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

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

ALURA; “Herança e Polimorfismo”, 2023

<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>

Pytho Org; “A Biblioteca Padrão do Python”;

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

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

ORG, Python; “10. Um breve passeio pela biblioteca padrão”

[**https://docs.python.org/pt-br/3/tutorial/stdlib.html**](https://docs.python.org/pt-br/3/tutorial/stdlib.html)

SOARES, Gustavo; “Python: Imports e Bibliotecas”; DevFuria, 2023

[**https://dev.to/iugstav/python-imports-e-bibliotecas-42c0**](https://dev.to/iugstav/python-imports-e-bibliotecas-42c0)

PINHEIRO, Fagner; “Gerenciando Pacotes em Projetos Python com o PIP”; TreinaWeb, 2018

<https://www.treinaweb.com.br/blog/gerenciando-pacotes-em-projetos-python-com-o-pip>

ORG, Python; “A Biblioteca Padrão do Python”

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

**Pense Python; “Modulos e Obtendo Ajuda”; Panda, USP**

<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>.

Geo Sem Fronteiras; “5 Bibliotecas Python Para Conhecer”

[**https://geosemfronteiras.org/blog/5-bibliotecas-python-para-conhecer/**](https://geosemfronteiras.org/blog/5-bibliotecas-python-para-conhecer/)

Algoritmos Em Python; “Módulos e Bibliotecas”

<https://algoritmosempython.com.br/cursos/programacao-python/modulos-bibliotecas/#:~:text=Portanto%2C%20a%20biblioteca%20padr%C3%A3o%20Python,funcionalidades%20deles%20quando%20bem%20quiser>

CANDIDO; “Função \_\_init\_\_py”, School Of Net

<https://schoolofnet.com/forum/topico/qual-a-necessidadefuncao-do-__init__py-4308>