**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.*

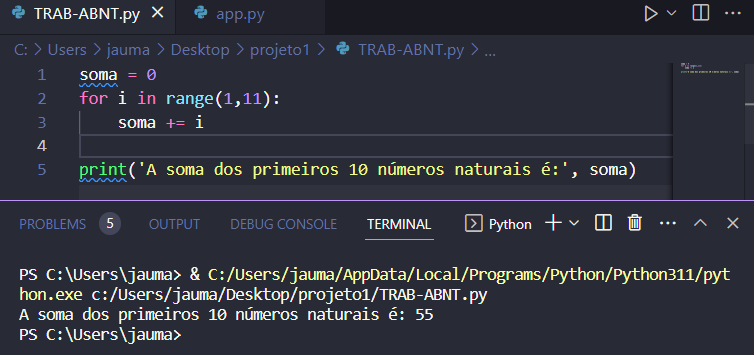
*Para exemplo, vamos usar os loops For e While, que são os principais.*

*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**:

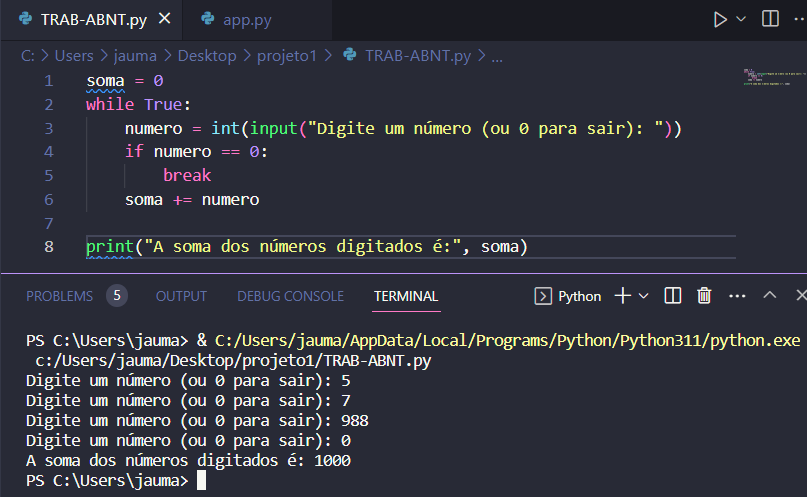
* **FOR:**



Fonte: Grupo.

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

* **WHILE:**



Fonte: Grupo.

Pode se notar que, neste exemplo, usamos um loop ‘while True’ para criar um loop infinito. Dentro do loop, pedimos ao usuário para digitar um número. Se o número digitado for igual a zero, o loop é interrompido com a instrução ‘break’. Caso contrário, o número é somado à variável ‘soma’. O programa continuará solicitando números até que o usuário insira 0, o momento em que a soma dos números digitados será exibida.

**Estruturas condicionais e lógicas**

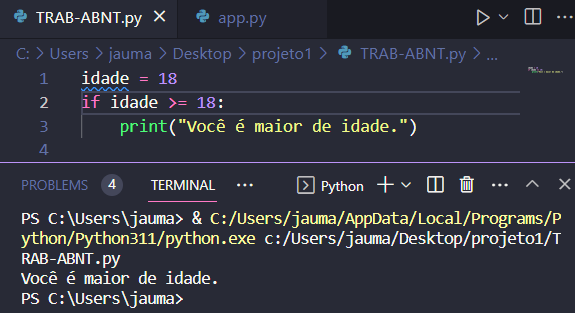
*“Uma estrutura condicional na linguagem Python, como a Python If Else, corresponde a um bloco de código que é iniciado com uma expressão para avaliar se uma determinada condição é verdadeira ou falsa. Com ele, podemos testar se uma variável é igual a zero, por exemplo. O resultado dessa verificação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

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

As estruturas lógicas em Python são fundamentais para controlar o fluxo de um programa com base em condições. Por sua vez, estruturas condicionais são compornentes-chaves da programação, permitindo decisões com base em condições. Usam ‘if, ‘else’ e ‘elif’(que explicaremos a seguir com exemplos) para executar diferentes ações com base em valores booleanos, direcionando o fluxo do programa e tornando-o adaptável a difetentes situações. Essas estruturas são essenciais para se criar programas que respondam dinamicamente às necessidades do usuário ou às condições variáveis durante a execução.

As principais portas lógicas incluem:

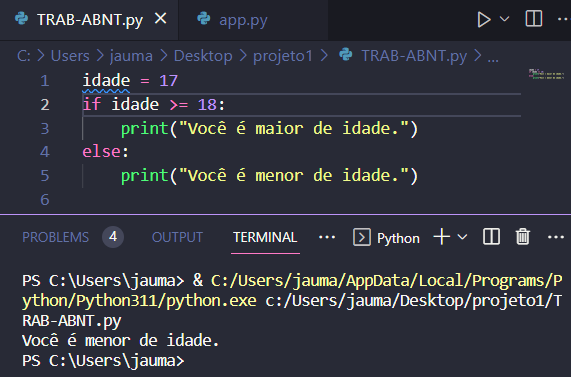
* **IF:** usado para executar um bloco de código se uma condição for verdadeira.



Fonte: Grupo.

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

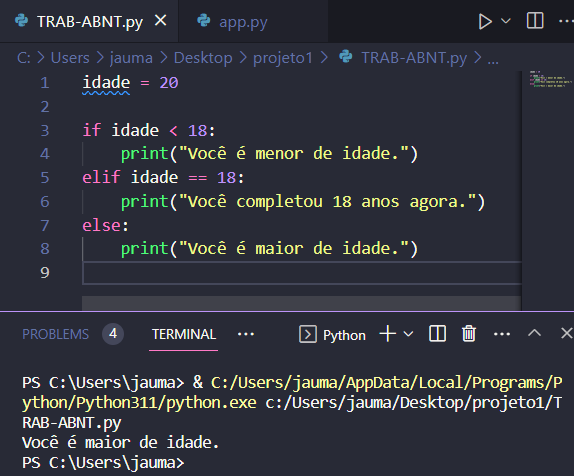
* **ELSE:** complementando o ‘if’, é usado para executar um bloco de código quando a condição não é verdadeira.



Fonte: Grupo.

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

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



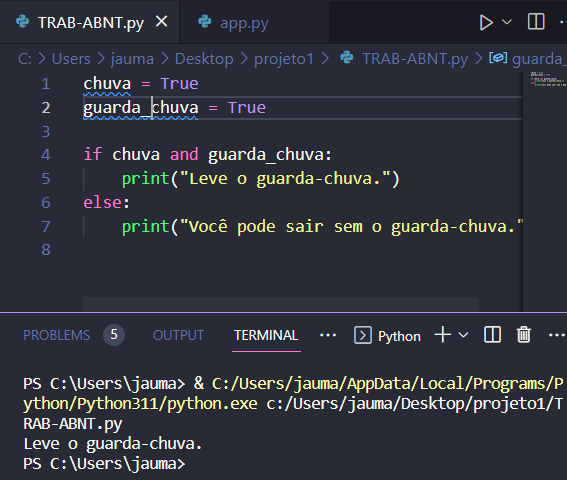
Fonte: Grupo

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

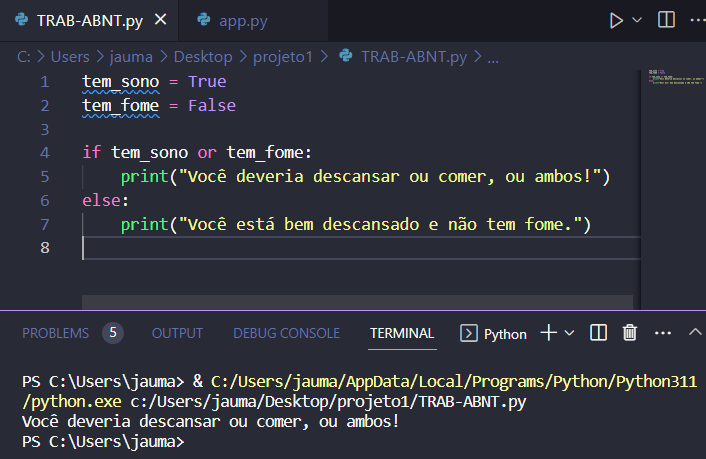
* **AND:** é um operador lógico que retorna ‘True’ apenas se todas as condições forem verdadeiras, caso contrário, retorna ‘False’.



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’.

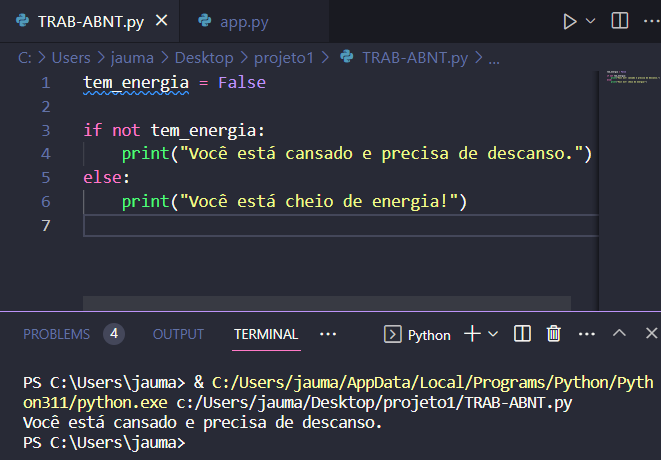


Fonte: Grupo.

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.



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.