Python estruturado

Prof. Humberto Henriques de Arruda

Prof. Kleber de Aguiar

Apresentação

Neste conteúdo, exploraremos as estruturas de decisão e repetição na linguagem Python, analisaremos os conceitos relacionados à implementação e utilização de subprogramas e bibliotecas, e examinaremos as técnicas de tratamento de exceções e eventos.

Propósito

Preparação

Antes de iniciar a leitura deste conteúdo, é necessário possuir uma versão de um interpretador Python e o ambiente de desenvolvimento PyCharm (ou outro ambiente que suporte o desenvolvimento na linguagem Python). Também é preciso conhecer tipos de variáveis em Python, assim como realizar a entrada e saída de dados em Python.

Para começar nossa jornada, baixe os **códigos-fonte originais** propostos para o aprendizado de Python estruturado. Descompacte-o em seu dispositivo. Com isso, você poderá utilizar os códigos como material de apoio ao longo de sua leitura!

Objetivos

Módulo 1

Decisão e repetição em Python

Descrever as estruturas de decisão e repetição em Python.

Módulo 2

Uso de subprogramas em Python

Definir os principais conceitos de subprogramas e a sua utilização em Python.

Módulo 3

Bibliotecas em Python

Módulo 4

Eventos em Python

Identificar o uso correto de recursos de bibliotecas em Python.

Reconhecer as formas de tratamento de exceções e eventos em Python.

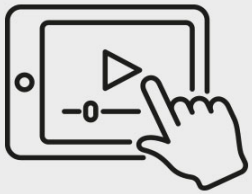
Introdução

Programar significa, como em qualquer disciplina, aprender ferramentas que permitam desenvolver melhor a sua atividade. Ao conhecer os conceitos básicos de programação, o estudante desenvolve habilidades iniciais para escrever seus primeiros programas. No entanto, é difícil imaginar que aplicações profissionais sejam feitas totalmente baseadas apenas nesses conceitos básicos.

Em aplicações mais complexas, é essencial considerar a necessidade de ganhar tempo, com o computador executando as tarefas repetitivas, assim como as demandas de manutenção e tratamento de erros. Para avançar no aprendizado da programação, você conhecerá novas ferramentas (entre elas, as estruturas de controle, como decisão e repetição), além de subprogramas e bibliotecas, bem como formas de tratar exceções e eventos.

No vídeo a seguir, faremos a apresentação deste conteúdo, abordando as estruturas de decisão e de repetição, a utilização de subprogramas e de bibliotecas e as formas de tratamento de exceção e eventos. Acompanhe!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Material para download

Clique no botão abaixo para fazer o download do conteúdo completo em formato PDF.

Download material

1 - Decisão e repetição em Python

Ao final deste módulo, você será capaz de descrever as estruturas de decisão e repetição em Python.

Estruturas de decisão

Você irá descobrir o poder do condicional **if** em Python, compreendendo a estrutura de controle de fluxo mais fundamental da linguagem. Aprenderá a criar condições, tomar

decisões e controlar o fluxo do seu código de maneira eficiente e clara.

Assista ao vídeo e entenda como as estruturas de decisão permitem selecionar partes do código que serão executadas, repetindo blocos de instruções com base em algum critério. Confira!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.

Tratamento das condições

As estruturas de controle permitem selecionar quais partes do código (chamadas de estruturas de decisão) serão executadas e repetir blocos de instruções com base em algum critério, como uma variável de controle ou a validade de alguma condição (denominadas estruturas de repetição).

> **As estruturas de decisão e de repetição possuem sintaxes bastante semelhantes em C e em Python. Mesmo com essa grande semelhança, existe uma diferença crítica no tratamento das condições.**

Diferentemente da linguagem C, Python oferece o tipo bool. Por isso, cabe ressaltar a diferença de comportamento das duas linguagens nesse tratamento. Veja!

Python	C
Existe o tipo bool	Não existe o tipo bool
True	Qualquer valor diferente de 0 (zero)
False	0 (zero) ou vazio

Tabela: Tratamento das condições.
Humberto Henriques de Arruda.

Observe que o fato de haver o tipo bool em Python permite que as condições sejam tratadas como verdadeiras ou falsas, o que não é exatamente igual em C.

Estruturas de decisão

if

Em Python, usamos o **if** para executar um bloco de código se uma condição for verdadeira. A sintaxe básica é a que mostraremos a seguir.

Python

Se a condição for verdadeira, o bloco_de_codigo será executado.

Em seguida observe um exemplo do uso da condição **if**.

Exemplo com **if**

Imagine criar um sistema simples para orientar sobre a vacinação de crianças com base na idade informada pelo usuário. Veja como isso pode ser implementado.

Python

Se a condição **idade < 5** for verdadeira, as instruções dentro do bloco **if** serão executadas.

Agora, vamos adicionar uma instrução fora do bloco **if**, veja!

Python

No exemplo, se a condição **idade < 5** for verdadeira, as instruções dentro do bloco **if** serão executadas.

O que acontece com a instrução print('Cuide da saúde sempre. Até a próxima.') se a condição **idade < 5** for falsa?

Clique em um dos botões para responder.

Será executada.

Não será executada.

if-else

Podemos adicionar uma condição alternativa usando **else**. Isso nos permite definir o que fazer se a condição do **if** for falsa. A sintaxe é a que mostraremos a seguir.

Python

Em seguida apresentaremos um exemplo com **if-else**.

Dica

Lembre-se de que em Python a indentação é fundamental. Todas as instruções dentro de um bloco de decisão devem ser indentadas com quatro espaços. Isso torna o código mais legível e organizado.

Exemplo com **if-else**

Vamos expandir o exemplo com **if** para incluir uma condição alternativa. Veja!

Python

No exemplo, se **idade** for menor que 5, o primeiro bloco de código será executado. Caso contrário, o bloco dentro do **else** será executado.

Agora, vamos adicionar uma instrução fora do bloco **if-else**. Acompanhe!

Python

No exemplo, se **idade** for menor que 5, o primeiro bloco de código será executado. Caso contrário, o bloco dentro do **else** será executado. Independentemente da condição, a instrução **print**('Cuide da saúde sempre. Até a próxima.') será sempre executada porque está fora do bloco **if-else**.

elif

Além do **if** e **else**, Python nos oferece o **elif**, que é uma combinação de **else if**. O **elif** nos permite testar várias condições de forma sequencial. Confira a sintaxe no código.

Python

Agora, veja um exemplo com **elif**.

Exemplo com **elif**

Vamos adicionar mais uma condição ao nosso exemplo. Acompanhe!

Python

No exemplo, o Python verifica a primeira condição **idade** < 5. E a partir de então segue os seguintes passos.

1. Se a condição **idade** < 5 for verdadeira, executa o bloco de código correspondente.
2. Se a condição **idade** < 5 não for verdadeira, verifica a próxima condição **idade==** 5.
3. Se essa condição **idade==** 5 for verdadeira, executa o bloco de código do **elif**.
4. Se nenhuma das condições anteriores for verdadeira, executa o bloco de código do **else**.

Atividade 1

Entender o que um trecho de código está executando é sempre muito importante para o programador. Analise o código a seguir.

Python

Agora indique qual é a saída do programa se o valor digitado for 70.

A Você é uma criança.

B Você é um adolescente.

C Você é menor de idade.

D Você é adulto.

E

Você é um idoso.

Parabéns! A alternativa E está correta.

Se a idade inserida for menor que 10, o primeiro bloco if será executado e a mensagem "Você é uma criança" será exibida.

Se a idade estiver entre 10 e 15 (inclusive), o segundo bloco elif será executado e a mensagem "Você é um adolescente" será exibida.

Se a idade estiver entre 16 e 18 (exclusive), o terceiro bloco elif será executado e a mensagem "Você é menor de idade" será exibida.

Se a idade estiver entre 18 (inclusive) e 65 (exclusive), o quarto bloco elif será executado e a mensagem "Você é adulto" será exibida.

Se a idade for 65 ou mais, o bloco else será executado e a mensagem "Você é um idoso" será exibida.

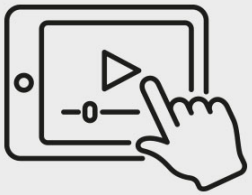
Como a idade digitada foi maior que 65, o else será executado.

Estrutura de repetição **for**

Aqui você vai compreender o universo do loop **for** e da função range em Python. Veja de forma clara e prática como usar essas ferramentas para iterar sobre sequências e executar tarefas repetitivas.

Assista ao vídeo e observe a estrutura de repetição For na linguagem de programação Python. Confira as listas do tipo range(), a sintaxe da estrutura **for**, o laço **for** com uma **string** e o uso do laço **for** com qualquer sequência.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



As estruturas de repetição **for** permitem repetir um bloco de código para cada item de uma sequência.

Antes de detalharmos o **for**, vamos conhecer uma função de Python que gera uma lista de valores numéricos. Essa lista nos ajudará a entender a repetição e deixará mais claro o funcionamento do laço.

As listas do tipo range()

Ao chamar o **método range()**, Python cria uma sequência de números inteiros, desde uma maneira simples até a mais complexa. Observe!

1

Simples

Envolve apenas um argumento. Nesse caso, a sequência começará em 0 e será incrementada de uma unidade até o limite do parâmetro passado (exclusive).

Exemplo: range(3) # Cria a sequencia (0, 1, 2)

2

Não iniciadas em 0

Para que a sequência não comece em 0, pode-se informar o início e o fim como parâmetros. Lembre-se de que o parâmetro fim não entra na lista (exclusive o fim). O padrão é incrementar cada termo em uma unidade.

Exemplo: range(2, 7) # Cria a sequência (2, 3, 4, 5, 6)

3

Indicando início, fim e passo

É possível criar sequências mais complexas indicando, na ordem, os parâmetros de início, fim e passo. O passo é o valor que será incrementado de um termo para o próximo.

Exemplo: range(2, 9, 3) # Cria a sequência (2, 5, 8)

A sintaxe da estrutura **for**

A estrutura **for** tem a seguinte sintaxe em Python:

Python

Exemplos com **for**

Vamos analisar um exemplo simples em Python: imprimir todos os elementos de uma sequência criada com a chamada **range()**. Observe a seguir uma possível implementação desse exemplo (código 2 no arquivo disponibilizado na introdução deste conteúdo). Clique em Executar no emulador.

Exercício

TUTORIAL COPIAR

Python3

Input

Digite cada input em uma linha.

Console

Executar

Observe a análise a seguir.

1. A linha 1 mostra a criação do laço com a variável item percorrendo a sequência (2, 5, 8), que é criada pela chamada range(2, 9, 3).
2. A linha 2 indica a instrução que será executada para cada repetição desse laço. O laço **for** executa a instrução da linha 2 três vezes, uma para cada elemento da sequência (2, 5, 8).

Agora vamos criar um exemplo em que contamos de 0 a 10, imprimindo cada um dos números. Acompanhe!

Python

Analisaremos o código a seguir.

1. A linha 1 mostra a criação do laço com a variável numero percorrendo a sequência de 0 a 10, que é criada pela chamada range(11).
2. A linha 2 indica a instrução que será executada para cada repetição desse laço. O laço **for** executa a instrução da linha 2 onze vezes, uma para cada elemento da sequência (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10).

Vamos explorar mais um exemplo do uso da estrutura for em Python. Neste caso, vamos calcular e imprimir a tabuada do número 5. O código a seguir realiza essa tarefa de forma simples e eficiente. Confira!

Python

No exemplo, a variável numero é definida com o valor 5, que é o número para o qual queremos calcular a tabuada.

Utilizamos um laço **for** para iterar sobre uma sequência de números de 1 a 10, gerada pela função range(1, 11). Para cada valor de i nessa sequência, calculamos o resultado da multiplicação de numero por i e armazenamos esse valor na variável

resultado. Em seguida, imprimimos o resultado no formato numero **x** i = resultado, utilizando uma **f-string** para formatar a saída.

O laço **for** com uma **string**

Python também permite que a repetição aconteça ao longo de uma **string**.

Relembrando

A **string** é uma sequência de caracteres individuais.

Exemplo com **string**

Suponha que você queira soletrar o nome informado pelo usuário. Uma possível implementação está demonstrada a seguir (no arquivo disponibilizado na introdução, trata-se do código 3).

Informe um nome (**string**) no campo Input do emulador e clique em Executar.

Exercício

TUTORIAL COPIAR

Python3

Input

Digite cada input em uma linha.

Console

Executar

Observe a análise a seguir.

1. A linha 1 faz com que a palavra inserida pelo usuário seja armazenada na variável nome.
2. A linha 2 mostra a criação do laço, com a variável letra percorrendo a sequência de caracteres armazenada na variável nome.
3. A linha 3 indica a instrução que será executada para cada repetição desse laço. O laço **for** executará a instrução da linha 3 tantas vezes quantos forem os elementos da sequência que está na variável nome.

Vamos explorar mais um exemplo do uso da estrutura **for** em Python para trabalhar com strings. Neste exemplo, iremos contar quantas vezes uma letra específica aparece em determinada palavra. Veja o código a seguir.

Python

Agora, vamos analisar cada uma das linhas.

1. A variável texto contém a **string** na qual queremos contar as ocorrências de uma letra específica.
2. A variável letra_para_contar armazena a letra que queremos contar.
3. A variável contador é inicializada com o valor 0 e será usada para contar as ocorrências da letra.
4. O laço **for** itera sobre cada letra da **string** texto.
5. Dentro do laço, uma estrutura **if** verifica se a letra atual é igual à letra_para_contar. Se for, o contador é incrementado em 1.
6. Após o laço, o resultado é impresso, mostrando quantas vezes a letra especificada aparece na **string**.

Uso do laço for com qualquer sequência

Até agora estudamos o uso do laço for com iterações sobre strings e sequências numéricas, mas Python permite ainda mais do que isso!

Podemos utilizar o laço for com iterações sobre qualquer sequência, e não somente as numéricas e as strings.

O for pode ser empregado para percorrer uma lista. Em Python, uma lista é uma coleção de elementos que pode conter itens de diferentes tipos, embora geralmente seja usada para armazenar itens do mesmo tipo.

> **Listas são mutáveis, o que significa que seus elementos podem ser alterados após a criação. Elas são definidas usando colchetes [ ] e os elementos são separados por vírgulas.**

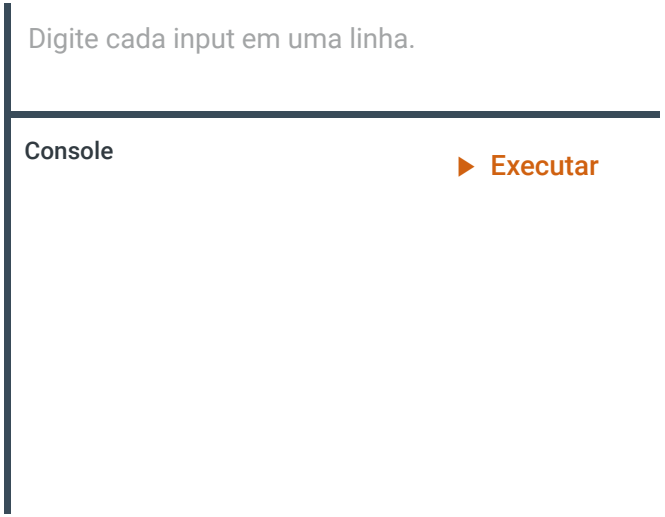
Observe este exemplo (código 4 do arquivo disponibilizado) que utiliza listas.

Exercício

TUTORIAL COPIAR

Python3

Input



Clique em Executar no emulador anterior e veja o resultado da execução do código.

Agora analisaremos o código.

1. A variável nomes é uma lista que contém cinco elementos, cada um representando um nome: 'Laura', 'Lis', 'Guilherme', 'Enzo', e 'Arthur'.
2. Utilizamos um laço **for** para iterar sobre cada elemento da lista nomes.
3. Para cada iteração, a variável nome assume o valor do próximo elemento da lista, e o **print(nome)** imprime esse valor.

Como resultado, o código imprime cada nome da lista em uma nova linha. Veja!

Laura
Lis
Guilherme
Enzo
Arthur

Vamos ver outro exemplo em que somamos todos os números em uma lista. Acompanhe!

Python

No exemplo podemos observar que:

1. A linha 1 define uma lista de números chamada numeros.
2. A linha 2 inicializa a variável soma com 0.
3. A linha 3 mostra a criação do laço, com a variável numero percorrendo cada elemento na lista numeros.
4. A linha 4 adiciona o número atual à variável soma.
5. A linha 5 imprime a soma de todos os números.

Vamos agora ver outro exemplo em que calculamos o quadrado de cada número em uma lista de números. Acompanhe!

Python

Agora, analise o exemplo e preencha as lacunas com as palavras apresentadas. No final, verifique o resultado.

1. A linha 1 define uma lista de números chamada [ ] .

2. A linha 2 mostra a criação do laço, com a variável [ ] percorrendo cada elemento na lista [ ] .

3. A linha 3 [ ] o quadrado do número atual.

4. A linha 4 [ ] o número e seu quadrado.

numeros imprime numero calcula numeros

Tentar novamente | Verificar

Atividade 2

Considere o seguinte trecho de um programa escrito em Python.

Python

Marque a opção que apresenta corretamente o que será impresso na tela.

A 0 3 9 18 30

B 0 3 6 9 12

C 30

D 45

E 3 6 9 12 15

Parabéns! A alternativa C está correta.

A variável s é iniciada com o valor zero.

Na primeira iteração do laço for, i vale 0 e assim, a variável s continua com o valor 0 (3x0 + 0). Na iteração seguinte, i vale 1 e a variável s é atualizada para 3 (3x1+ 0). Quando o for está na terceira iteração, i vale 2 e com isso a variável s recebe o valor 9 (3x2 + 3). Na quarta iteração, i vale 3 e s passa a valer 18 (3x3 + 9). Por fim, na quarta e última iteração do laço for, i vale 4 e a variável s recebe 30 (3x4 + 18)

Por isso, será impresso na tela o valor 30.

Estrutura de repetição **while** e instruções auxiliares

Agora você vai descobrir os segredos do loop **while** e das declarações **break**, **continue** e **pass** em Python. Confira a lógica por trás dessas estruturas de controle de fluxo e aprenda a criar loops eficientes, interromper iterações indesejadas, continuar para a próxima iteração ou simplesmente passar adiante.

Para entender melhor, assista ao vídeo!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.

Estrutura de repetição **while**

Permite executar repetidamente um bloco de código enquanto uma condição for verdadeira. Vamos aprender como utilizá-la e ver alguns exemplos para entender melhor.

A sintaxe do **while** é a apresentada a seguir.

O bloco_de_codigo será repetido **enquanto** a condição for verdadeira. Assim que a condição se tornar falsa, a execução sai do laço **while**.

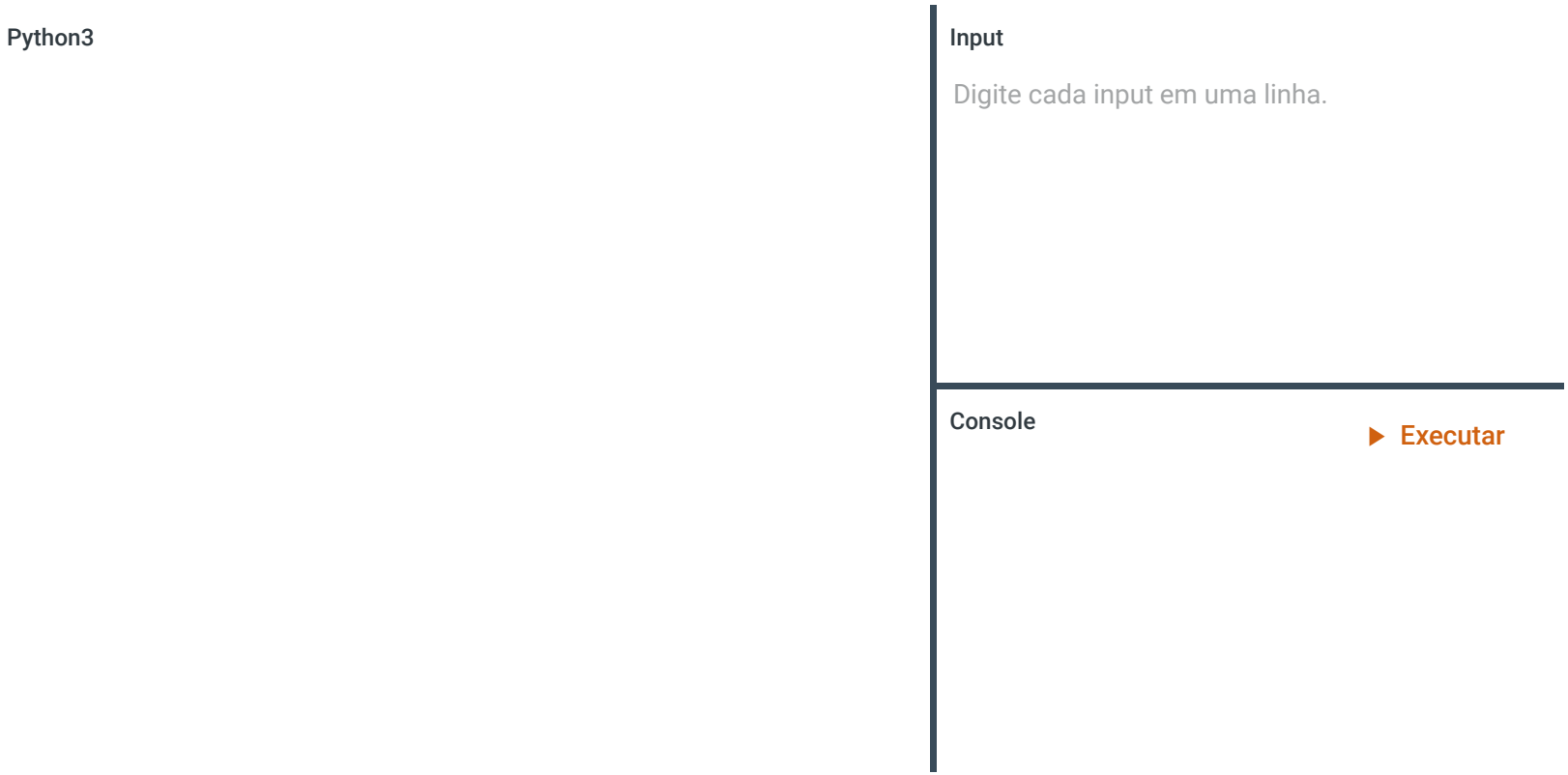
A seguir veremos um exemplo.

Exemplo com **while**

Vamos analisar um programa em que o usuário precisa digitar a palavra "sair" para que o laço **while** seja encerrado.

> **Para que o exemplo proposto seja executado corretamente, deve-se inserir, no campo Input do emulador, uma sequência de palavras, uma por linha, sendo que a última tem de ser a palavra “sair”.**

Observe o código!



No exemplo, podemos observar o seguinte:

1. A linha 1 solicita ao usuário que insira uma palavra, que é armazenada na variável palavra.
2. A linha 2 cria o laço **while**, que depende da condição .
3. A linha 3 será repetida enquanto a condição for verdadeira, ou seja, enquanto o valor da variável palavra for diferente de 'sair'. Quando os valores forem iguais, a condição do laço **while** será falsa e o laço será encerrado.
4. A linha 4 imprime a mensagem fora do laço **while**.

Agora, analise o código e responda.

Se você digitar “sair” logo na primeira solicitação, o que acontecerá com a linha 3 do nosso programa?

Clique em um dos botões para responder.

Será executada uma vez.

Não será executada nenhuma vez.

O laço **while** infinito

Laços infinitos são úteis quando queremos executar um bloco de instruções indefinidamente. O laço **while** infinito tem o formato que mostraremos a seguir.

Python

Veremos agora um exemplo!

Exemplo de laço infinito

Vamos criar uma aplicação que permanece por meses ou anos sendo executada, registrando a temperatura ou a umidade de um ambiente. Logicamente, você possui essa informação disponível a partir da leitura de algum sensor. Você, portanto, precisa tomar cuidado e ter a certeza de que seu uso é realmente necessário para evitar problemas de consumo excessivo de memória.

Observe o código!

Python

No exemplo, o laço **while** é infinito e continuará lendo e registrando a temperatura a cada 60 segundos.

As instruções **break**, **continue** e **pass**

A instrução **break**

É utilizada para interromper as repetições dos laços **for** e **while**. Quando o programa encontra uma instrução **break**, a repetição é encerrada e o fluxo do programa continua a partir da primeira instrução após o laço. Vamos entender como a instrução **break** funciona!

Exemplo com **break** no laço **while**

Vamos voltar ao exemplo do laço **while** infinito. O laço será encerrado quando o usuário inserir a palavra "sair".

> **Para que o exemplo proposto seja executado corretamente, deve-se inserir, no campo Input do emulador, uma sequência de palavras, uma por linha, sendo que a última tem de ser a palavra “sair”.**

No emulador, clique em Executar (trata-se do código 6 do arquivo disponibilizado):
while true:

Exercício

TUTORIAL COPIAR

Python3

Input

Digite cada input em uma linha.

Console

Executar

Vamos entender linha a linha o que o código faz, confira!

1. A linha 1 cria um laço **while** infinito.
2. A linha 2 solicita ao usuário que insira uma palavra.
3. A linha 3 verifica se a palavra inserida é "sair". Se for, a instrução **break** é executada, encerrando o laço **while**.
4. A linha 4 imprime uma mensagem fora do laço **while**.

Exemplo com **break** em laços aninhados

Vejamos um exemplo com laços **while** aninhados.

Python

Vamos analisar, novamente, linha a linha o que esse código realiza. Confira!

1. A linha 1 cria o primeiro laço **while** infinito.
2. A linha 2 imprime uma mensagem indicando que o usuário está no primeiro laço.

3. A linha 3 solicita ao usuário que insira uma opção.
4. A linha 4 verifica se a opção inserida é SIM. Se for, a instrução **break** é executada, encerrando o primeiro laço **while**.
5. Caso contrário, a linha 6 cria um segundo laço **while** infinito.
6. A linha 7 imprime uma mensagem indicando que o usuário está no segundo laço.
7. A linha 8 solicita ao usuário que insira uma opção.
8. A linha 9 verifica se a opção inserida é SIM. Se for, a instrução **break** é executada, encerrando o segundo laço **while**.
9. A linha 11 imprime uma mensagem indicando que o usuário saiu do segundo laço.
10. A linha 12 imprime uma mensagem indicando que o usuário saiu do primeiro laço.

A instrução **break** é muito útil quando precisamos sair de um laço antes que ele termine normalmente. Ela pode ser usada tanto em laços **while** quanto em laços **for**.

A instrução **continue**

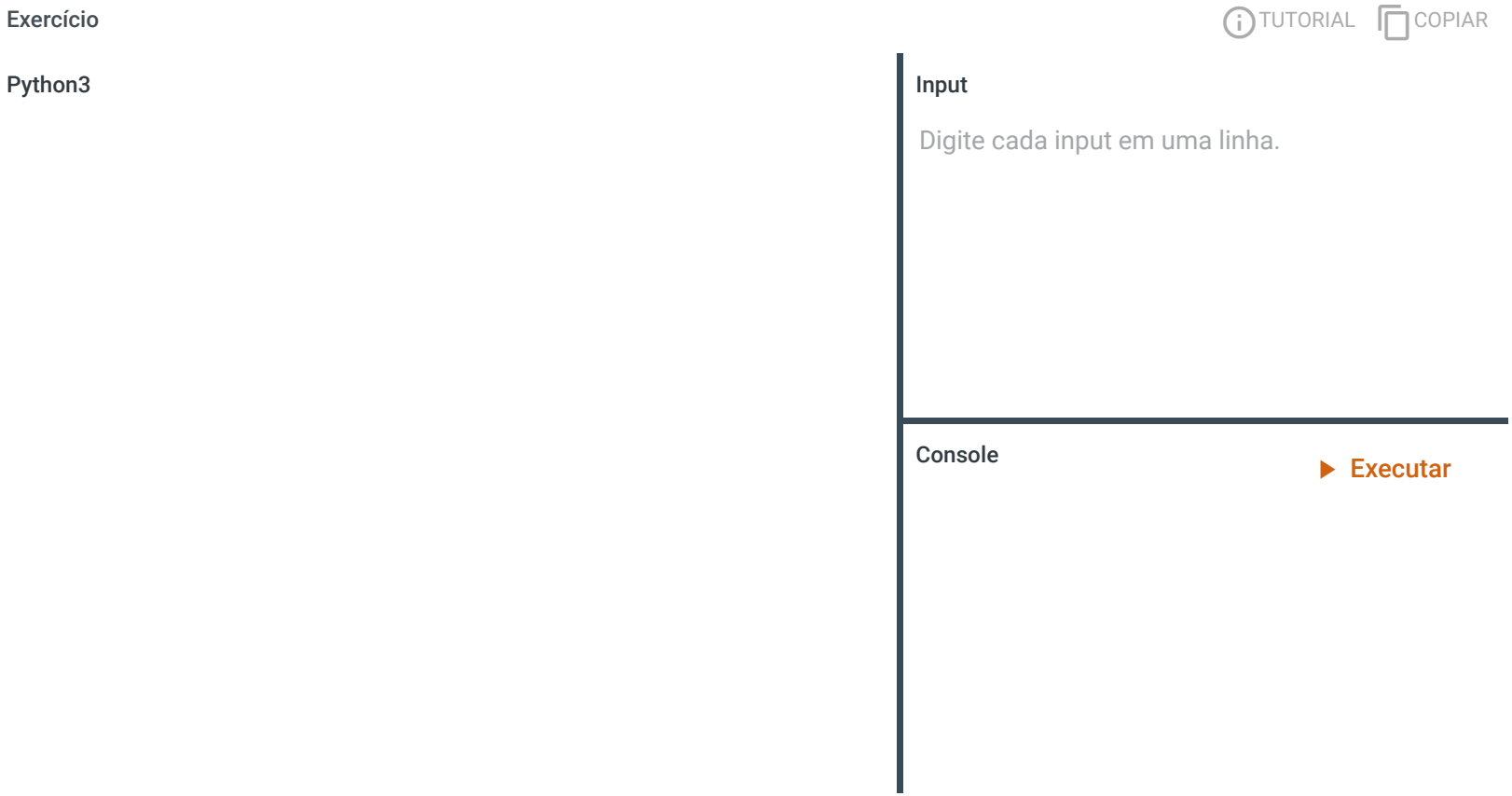
É usada para pular a iteração atual de um laço e passar para a próxima iteração. Diferente da instrução **break**, que encerra completamente o laço, a instrução **continue** apenas interrompe a iteração corrente e continua o laço desde o início.

Comentário

Quando o Python encontra a instrução **continue**, ele ignora todas as instruções restantes no laço para aquela iteração e vai direto para a próxima iteração. Isso é útil quando você deseja pular certos valores ou condições dentro de um laço.

Exemplo com **continue**

Vejamos um exemplo no qual imprimimos todos os números inteiros de 1 a 10, pulando apenas o número 5. Veja sua implementação (código 8 do arquivo disponibilizado) no emulador.



Analisando linha a linha do código, observamos que:

1. A linha 1 cria um laço **for** que itera sobre a sequência de números de 1 a 10.
2. A linha 2 verifica se o número atual é 5.
3. Se o número atual for 5, a instrução **continue** é executada, pulando o restante do bloco do laço e iniciando a próxima iteração.
4. Caso contrário, a linha 5 imprime o número atual.
5. Após o laço terminar, a linha 7 imprime a mensagem "Laço encerrado".

Para entender a diferença entre **break** e **continue**, vamos modificar o exemplo anterior substituindo **continue** por **break** e ver o que acontece. Essa alteração está no arquivo disponibilizado (código 9).

Python

Agora, analise o exemplo e preencha as lacunas com as palavras apresentadas. No final, verifique o resultado.

1. Quando o número atual é 5, a instrução [ ] é executada, encerrando o laço [ ] imediatamente.

2. Isso significa que os números após 5 [ ] impressos.

for não serão **break** serão numeros

Tentar novamente | Verificar

Com os exemplos, podemos ver claramente como **continue** permite pular uma iteração específica sem interromper todo o laço, enquanto **break** encerra o laço por completo.

A instrução **pass**

É usada como um marcador de posição. Ela é útil quando você precisa de uma sintaxe que exige um bloco de código, mas você ainda não decidiu o que escrever nesse bloco. Em outras palavras, **pass** permite que você escreva uma estrutura que não faz nada, mas mantém a sintaxe correta.

Comentário

Quando o Python encontra a instrução **pass**, ele simplesmente continua a execução sem fazer nada. Isso pode ser útil em várias situações, como durante o desenvolvimento de código, em que você deseja planejar a estrutura do seu programa antes de preencher os detalhes.

Exemplo com **pass**

Vejamos um exemplo no qual imprimimos somente os números ímpares entre 1 e 10. Usaremos **pass** para os números pares.

Uma implementação possível está no emulador a seguir (código 10 do arquivo disponibilizado).

Exercício

TUTORIAL COPIAR

Python3

Input

Digite cada input em uma linha.

Console

Executar

Vamos verificar o que esse código faz!

1. A linha 1 cria um laço **for** que itera sobre a sequência de números de 1 a 10.
2. A linha 2 verifica se o número atual é par (ou seja, divisível por 2 sem resto).
3. Se o número **for** par, a instrução **pass** é executada, o que significa que nada acontece e o laço continua para a próxima iteração.
4. Se o número **for** ímpar, a linha 6 imprime o número atual.
5. Após o laço terminar, a linha 8 imprime a mensagem "Laço encerrado".

Nesse exemplo, a instrução **pass** é usada para indicar que nada deve ser feito quando o número é par. Isso ajuda a manter a estrutura do código clara e correta.

Claramente, seria possível reescrever a condição do **if-else** para que pudéssemos transformá-lo em um **if** simples, sem **else**. Entretanto, o objetivo aqui é mostrar o uso da instrução **pass**.

Atividade 3

Considere o seguinte trecho de um programa escrito em Python.

Python

Marque a opção que apresenta corretamente o que será impresso na tela.

A 9

B 3 6

C 3 3

D 3 6 9 12

E 0 3 6 9

Parabéns! A alternativa B está correta.

Ao ser testada pela primeira vez, a condição do while é verdadeira, já que s vale zero. Assim, a variável s recebe o valor 3 (3x1), enquanto a variável a é acrescida de uma unidade, ficando com o valor 2. Em seguida, é impresso o valor de s (3). A condição do while é então testada novamente, sendo mais uma vez verdadeira, porque s tem o valor 3 (menor que 5). Nessa iteração, a variável s recebe o valor 6 (3x2), e a variável a é acrescida de uma unidade, ficando com o valor 3. Em seguida, é impresso o valor de s (6). A condição do while é testada novamente e se revela falsa, pois s tem o valor 6 (maior que 5). Com isso, o laço while é encerrado e nada mais é impresso. Por isso, foram impressos os valores 3 e 6.

Tipos de dados sequenciais e dicionário

Em Python, os tipos sequenciais, como listas, tuplas e conjuntos, têm um papel essencial na organização e manipulação de dados, permitindo armazenar coleções ordenadas e únicas de elementos. Operadores comuns, como indexação, fatiamento e concatenação, são utilizados para acessar e manipular essas sequências de forma eficiente.

Além disso, os dicionários de estruturas de dados chave-valor oferecem uma maneira eficaz de associar informações, sendo úteis para armazenamento e recuperação de dados de forma rápida e eficiente em aplicações Python.

Assista ao vídeo e acompanhe os tipos de dados sequenciais e os dicionários em Python.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.

Tipos sequenciais

Existem três tipos sequenciais básicos em Python, veja!

1

Listas

2

Tuplas

3

Objetos range

Além desses, existe um tipo especial criado para tratamento de dados textuais: o tipo str (**string**).

Assim como em C ou Java, a indexação dos itens é iniciada com 0 e cada item tem o seu índice incrementado a uma unidade em relação ao item anterior. No entanto, Python também permite a indexação com valores negativos. O valor -1 é o índice do último item, e cada item anterior é decrementado de uma unidade em relação ao sucessor. Observe a tabela!

índice	0	1	
s	**t**	**e**	
índice negativo	-5	-4	

Índices em tipos sequenciais.
Humberto Henriques de Arruda.

Strings

Em uma variável do tipo **str**, é possível armazenar letras, números, espaços, pontuação e diversos símbolos. Diferentemente da linguagem C, não existe o tipo **char**. Cada caractere em Python é uma **string**. Veja o que podemos utilizar para delimitar uma **string**.

Aspas simples	'uma **string**'
Aspas duplas	"uma string"
Aspas simples triplas	'''uma string'''
Aspas duplas triplas	"""uma string"""

Tabela: Tratamento das condições.
Humberto Henriques de Arruda.

O código a seguir ilustra um exemplo de delimitadores de strings, veja!

Python

Existem alguns métodos interessantes para tratar strings em Python. Entre eles, ressaltamos os seguintes:

Upper

Transforma todas as letras em maiúsculas.

Lower

Transforma todas as letras em minúsculas.

Split

Quebra a **string** em substrings.

Veja o exemplo!

Python

A lista gerada com o método **split()** tem três elementos, porque a **string** original tinha três palavras.

Listas

São sequências mutáveis, normalmente usadas para armazenar coleções de itens homogêneos. Uma lista pode ser criada de algumas maneiras, confira!

[]

Usando um par de colchetes para denotar uma lista vazia.

[a], [a, b, c]

Usando colchetes, separando os itens por vírgulas.

[x for x in iterable]

Usando a compreensão de lista.

list() ou list(iterable)

Usando o construtor do tipo **list**.

Iterable pode ser uma sequência, um container que suporte iteração ou um objeto iterador.

Por exemplo, **list**('abc') retorna ['a', 'b', 'c'] e **list**((1, 2, 3)) retorna [1, 2, 3]. Se nenhum argumento for passado, o construtor cria uma lista vazia: [ ].

Suporte iteração ou um objeto iterador

Um iterador é um objeto que contém um número contável de valores. Ele pode ser iterado, o que significa que podemos percorrer todos os valores.

Tuplas

São sequências imutáveis, tipicamente usadas para armazenar coleções de itens heterogêneos. Elas são aplicadas também quando é necessário utilizar uma sequência imutável de dados homogêneos. Uma tupla pode ser criada de algumas maneiras:

()

Usando um par de parênteses para denotar uma tupla vazia.

a, b, c ou (a, b, c)

Separando os itens por vírgulas.

tuple() ou tuple(iterable)

Usando o construtor do tipo tuple.

Novamente, **iterable** pode ser uma sequência, um container que suporte iteração ou um objeto iterador.

Por exemplo, **tuple('abc')** retorna ('a', 'b', 'c') e **tuple([1, 2, 3])** retorna (1, 2, 3). Se nenhum argumento for passado, o construtor cria uma tupla vazia: ().

Comentário

O uso das vírgulas é o que gera a tupla, e não o uso de parênteses. Os parênteses são opcionais, exceto no caso em que queremos gerar uma tupla vazia.

Range

Representa uma sequência imutável de números e frequentemente é usado em loops de um número específico de vezes, como o **for**.

> **O range pode ser chamado de maneira simples, apenas com um argumento. Nesse caso, a sequência começará em 0 e será incrementada de uma unidade até o limite do parâmetro passado (exclusive). Por exemplo, range(3) cria a sequência (0, 1, 2).**

Para que a sequência não comece em 0, podemos informar o início e o fim como parâmetros, lembrando que o parâmetro fim não entra na lista (exclusive o fim). O padrão é incrementar cada termo em uma unidade. Ou seja, a chamada **range(2, 7)** cria a sequência (2, 3, 4, 5, 6).

Comentário

Em Python, é possível criar sequências mais complexas, indicando os parâmetros de início, fim e passo, nessa ordem. O passo é o valor que será incrementado de um termo para o próximo. Por exemplo, **range(2,**

9, 3) cria a sequência (2, 5, 8).

Operadores sequenciais comuns

Permitem a manipulação dos tipos sequenciais, inclusive as strings. Vale ressaltar a sobrecarga dos operadores **+** e ***, que realizam operações diferentes quando os operandos são numéricos ou sequenciais.

Exemplo

O operador **==** verifica se as strings dos dois lados são iguais. Já os operadores < e> comparam as strings usando a ordem do dicionário.

A tabela a seguir traz um pequeno conjunto dos operadores disponíveis em Python para manipulação de sequências. Lembre-se de que você pode utilizar o utilitário **help** no Python Console para verificar a lista completa. Para isso, basta digitar **help(str)** e pressionar [ENTER] no teclado.

Uso	Resultado
x in s	**True** se **x** for um subconjunto de **s**
x not in s	**False** se **x** for um subconjunto de **s**
s + t	Concatenação de **s** e **t**
n*s	Concatenação de **n** cópias de **s**

Uso	Resultado
s[i]	Caractere de índice **i** em **s**
len(s)	Comprimento de **s**
min(s)	Menor item de **s**
max(s)	Maior item de **s**

Operadores sequenciais.
Humberto Henriques de Arruda.

Dicionários

Permitem que itens de uma sequência recebam índices definidos pelo usuário. Um dicionário contém pares de (chave, valor). A seguir, observe o formato geral de um objeto dicionário.

Python

Poderíamos criar um dicionário em que cada pessoa fosse representada pelo seu CPF, com nome e sobrenome. Para isso, teríamos o seguinte código:

Python

No código, o dicionário tem 3 entradas. Observe como foi possível recuperar nome e sobrenome de uma entrada, baseado na chave informada ‘111222333-44’.

Atividade 4

Em Python, as strings podem ser delimitadas por aspas simples, aspas duplas, aspas simples triplas ou aspas duplas triplas. Essa flexibilidade permite a inclusão de letras, números, espaços, pontuações e diversos símbolos em uma única variável do tipo str.

Além disso, Python oferece métodos úteis para manipulação de strings. Agora, observe a frase a seguir.

'CONSIDERANDO O TEXTO, ANALISE AS ALTERNATIVAS E ASSINALE A OPÇÃO CORRETA:'

Nesse sentido, ao usar o método 'split' na frase acima, quantos elementos a lista resultante terá?

A Dois

B Cinco

C Dez

D Onze

E Quatorze

Parabéns! A alternativa D está correta.

Quando a função 'split' é aplicada a uma string, ela quebra a string em substrings com base em um delimitador especificado, como um espaço em branco. No exemplo

fornecido, a string original tinha onze palavras, então a lista resultante terá onze elementos. Isso demonstra como o método 'split' pode ser utilizado para dividir strings em partes menores, facilitando a manipulação e análise de dados textuais em Python.

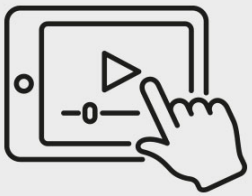
Estruturas de decisão e repetição na prática

Vamos agora colocar em prática os conceitos que vimos até agora.

Faça um programa em Python que mostre os números entre 1000 e 9999 cuja raiz quadrada seja igual à soma dos números formados pelos dois algarismos menos significativos e pelos dois algarismos mais significativos. Dica: existem três números que atendem a condição.

Neste vídeo, será executado um exercício prático mostrando a criação de estruturas de decisão e de repetição em Python. Acompanhe!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Roteiro de prática

Realize o seguinte passo a passo:

1. Crie um projeto no PYCHARM.
2. Utilize uma estrutura de **for** para gerar os números a serem testados.
3. Gere o número formado pelos algarismos menos significativos.
4. Gere o número formado pelos algarismos mais significativos.
5. Obtenha a raiz somando os dois números obtidos.
6. Eleve a raiz ao quadrado e valide se é igual ao número que está sendo testado.
7. Se for igual, exiba o número que está sendo testado, os números dos algarismos mais e menos significativos e a raiz.
8. Ao término do loop, informe que terminou e mostre o valor final da variável do **for**.

Confira o código fonte gerado no vídeo.

Código fonte

Python

Atividade 5

A solução apresentada inicialmente no vídeo prático é eficaz, mas não é a mais eficiente, já que ela testa todos os números entre 1000 e 9999.

Foi observado no vídeo uma característica especial, que é o fato de a raiz ser obrigatoriamente um número inteiro, já que ela irá resultar da soma de dois números inteiros.

A otimização demonstrada no vídeo foi diminuir o loop para testar apenas as raízes inteiras de 32 a 99, que geram números entre 1000 e 9999.

O valor 32 representa o menor número que tem uma raiz inteira e cujo quadrado está no intervalo de 1000 a 9999. Já o valor 99 representa o maior número que tem uma raiz inteira e cujo quadrado está no intervalo citado.

Altere então o programa apresentado inicialmente no vídeo prático, de modo que os valores 32 e 99 da otimização possam ser obtidos de maneira automática, sem nenhum cálculo anterior à execução do programa.

Digite sua resposta aqui

Chave de resposta

Python

2 - Conceitos de subprogramas e a sua utilização em Python

Ao final deste módulo, você será capaz de definir os principais conceitos de subprogramas e a sua utilização em Python.

Subprogramas

Vamos explorar agora a arte da modularidade com subprogramas em Python. Destacaremos a importância de dividir seu código em funções reutilizáveis, facilitando a manutenção e a compreensão.

Assista ao vídeo e acompanhe as características gerais dos subprogramas e suas definições básicas na linguagem de programação Python.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.

O que são subprogramas?

São blocos de código que realizam tarefas específicas dentro de um programa maior. Eles são fundamentais para a organização e reutilização de código em linguagens de programação.

> Em Python, chamamos subprogramas de funções.

Características gerais dos subprogramas

Vamos explorar algumas características importantes dos subprogramas. Confira!

Ponto único de entrada

Cada subprograma tem um único ponto de entrada, que é onde sua execução começa.

Suspensão do programa chamador

Quando um subprograma é chamado, o programa chamador é suspenso até que o subprograma termine sua execução.

Retorno ao chamador

Após a execução do subprograma, o controle sempre retorna ao ponto de chamada no programa chamador.

Definições básicas de subprogramas

Agora, vamos entender alguns conceitos básicos sobre subprogramas em Python.

Definição de subprograma

Um subprograma, ou função, é definido quando o desenvolvedor especifica nome, parâmetros (se houver) e o conjunto de ações que ele executará. Isso cria uma "interface" que descreve como a função deve ser chamada e o que ela faz.

Chamada de subprograma

Uma função é chamada quando o programa executa uma instrução que solicita explicitamente a execução dessa função. Essa chamada ativa a função, fazendo com que o Python execute o bloco de código associado a ela.

Ativação do subprograma

Uma vez chamada, a função se torna ativa. Ela continua em execução até que todas as instruções dentro dela sejam executadas. Quando a função termina, o controle retorna ao ponto de onde a função foi chamada.

Cabeçalho do subprograma

O cabeçalho de uma função é a primeira parte da definição da função. Ele inclui o nome da função e, opcionalmente, uma lista de parâmetros.

Vejamos agora um exemplo. Em Python, você define uma função com a palavra-chave **def**, seguida pelo nome da função e parênteses contendo os parâmetros, se houver. Observe!

No exemplo, minha_funcao é o nome da função e parametro1 e parametro2 são os parâmetros que a função aceita.

Definindo e chamando subprogramas

Em Python, as funções definidas pelo desenvolvedor devem ser precedidas pela palavra reservada **def**. Não são especificados o tipo de retorno e os tipos dos parâmetros.

Vamos criar uma primeira função para imprimir o clássico “Olá Mundo!” em Python. Como ficaria?

Aqui, definimos uma função chamada diz_ola que imprime "Olá, Mundo!". Para chamar essa função e executar o código dentro dela, basta usar o nome da função seguido de parênteses: diz_ola()

Comentário

Em Python, as sentenças de função **def** são executáveis. Isso implica que a função só pode ser chamada após a execução da sentença **def**.

Exemplo de função

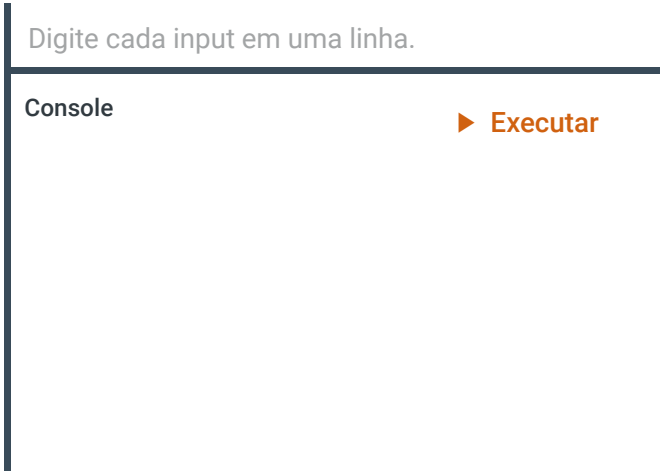
Vejamos um exemplo no qual o usuário pode escolher qual função chamar (código 11 do arquivo disponibilizado). Digite 1 ou 2 no campo Input do emulador e, depois, em Executar. Acompanhe!

Exercício

TUTORIAL COPIAR

Python3

Input



Vamos analisar linha a linha, acompanhe!

1. A linha 1 solicita ao usuário que escolha entre duas opções de função.
2. Se o usuário escolher "1", a função func1 é definida e chamada com o argumento 10, retornando 11.
3. Se o usuário escolher outra opção, a função func2 é definida e chamada com o argumento 10, retornando 12.
4. A linha final imprime o resultado da função escolhida.

Atividade 1

Qual das seguintes afirmações descreve corretamente as características de um subprograma em Python?

A Um subprograma em Python é uma função que pode ser chamada várias vezes durante a execução do programa.

B Um subprograma em Python é uma classe que define métodos específicos.

C Um subprograma em Python é um bloco de código que pode ser executado apenas uma vez.

D Um subprograma em Python é um conjunto de instruções que deve ser sempre executado em sequência.

E Um subprograma em Python é um comentário que documenta o código em um programa.

Parabéns! A alternativa A está correta.

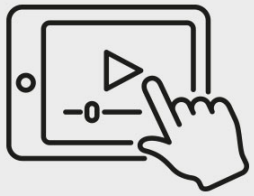
Um subprograma em Python é geralmente implementado como uma função. Uma função é um bloco de código que pode ser chamado várias vezes durante a execução do programa. Quando uma função é chamada, ela executa seu bloco de código e pode ser reutilizada sempre que necessário, promovendo a reutilização e organização do código.

Parâmetros, procedimentos e funções

Você aprenderá a definir, chamar e organizar subprogramas, bem como definir e utilizar parâmetros em funções, criar procedimentos para executar tarefas específicas e modularizar seu código de forma eficiente.

Assista ao vídeo e acompanhe a utilização de parâmetros, procedimentos e funções na linguagem de programação Python.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Parâmetros em Python

Vamos falar sobre parâmetros! Quando você cria uma função em Python, ela geralmente precisa de alguns dados para trabalhar. Existem duas maneiras principais para uma função obter esses dados, observe!

1

Acessar variáveis globais

Variáveis que estão fora da função, mas que a função consegue ver e usar.

2

Passagem de parâmetros

Passar variáveis diretamente para a função quando ela é chamada.

Usar parâmetros é uma maneira mais segura e flexível de passar dados para a função, pois evita que você altere acidentalmente variáveis globais importantes.

O que são parâmetros?

São os valores que você passa para a função quando a chama. Vamos ver um exemplo!

Aqui, peso e altura são parâmetros da função calculaIMC. Você chama essa função passando valores específicos para esses parâmetros. Veja!

Python

Quando chamamos a função com calculaIMC(70, 1.75), estamos passando "70 e 1.75" como argumentos para os parâmetros peso e altura, respectivamente.

Parâmetros formais e reais

Observe agora a diferença entre os parâmetros!

Formais		Reais (argumentos)
São aqueles que você define no cabeçalho da função. No exemplo, peso e altura são parâmetros formais.	×	São os valores que você passa para a função quando a chama. No exemplo, "70, 1.75" são argumentos.

Valores padrão para parâmetros

Em Python, você pode definir valores padrão para os parâmetros. Isso é útil quando você quer que um parâmetro tenha um valor padrão se não for fornecido.

Veja o exemplo de definição e chamada da função taxímetro no próximo código (no arquivo disponibilizado, trata-se do código 12).

Exercício

TUTORIAL COPIAR

Python3

Input

Digite cada input em uma linha.

Console

Executar

No exemplo, o parâmetro **multiplicador** tem um valor padrão de 1. Isso significa que, se não passarmos um valor para **multiplicador** ao chamar a função, ele usará 1 como padrão. Quando chamamos taximetro(3.5), a função usa multiplicador=1 por padrão.

A palavra reservada **return** indica que a função retorna algum valor. Isso implica que o valor retornado seja armazenado em uma variável do programa chamador (como ocorre na linha 8) ou utilizado como parâmetro para outra função.

Comentário

Retornar um valor é diferente de imprimir na tela. Ao utilizar a função **print()**, ocorre apenas a impressão de algo nela, o que não significa o retorno de qualquer função definida pelo usuário.

Procedimentos e funções

Os subprogramas podem ser procedimentos e funções. Veja a diferença entre eles de acordo com Sebesta (2018).

Procedimentos	Funções

São aqueles que não retornam valores.

São aquelas que retornam valores.

Na maioria das linguagens que não explicita a diferença entre ambos, as funções podem ser definidas sem retornar qualquer valor, tendo um comportamento de procedimento. Esse é o caso de Python.

Veja o código a seguir (código 13 no arquivo disponibilizado na introdução deste texto).

Exercício

TUTORIAL COPIAR

Python3

Input

Digite cada input em uma linha.

Console

Executar

As funções func1(x) e func2(x) não possuem qualquer retorno. Ou seja, elas são funções com comportamento de procedimentos.

Ambientes de referenciamento local

Variáveis locais

Quando um subprograma define as próprias variáveis, ele estabelece ambientes de referenciamento local. Essas variáveis são chamadas de **variáveis locais**, nas quais seu escopo usualmente é o corpo do subprograma.

As variáveis locais podem ser as que veremos a seguir.

Dinâmicas da pilha

São vinculadas ao armazenamento no início da execução do subprograma e desvinculadas quando essa execução termina. As variáveis locais dinâmicas da pilha têm diversas vantagens, e a principal delas é a flexibilidade. Suas principais desvantagens são o custo do tempo – para alocar, inicializar (quando necessário) e liberar tais variáveis para cada chamada ao

subprograma – e o fato de que os acessos a essas variáveis locais devem ser indiretos, enquanto os acessos às variáveis estáticas podem ser diretos.

Estáticas

São vinculadas a células de memória antes do início da execução de um programa e permanecem vinculadas a essas mesmas células até que a execução do programa termine. Elas são um pouco mais eficientes que as variáveis locais dinâmicas da pilha, já que não é necessário tempo para alocar ou liberar essas variáveis. Sua maior desvantagem é a incapacidade de suportar recursão.

Todas as variáveis locais em Python são dinâmicas da pilha. As variáveis globais são declaradas em definições de método; além disso, qualquer variável declarada **global** em um método precisa ser definida fora dele. Caso haja uma atribuição à variável local com o mesmo nome de uma variável **global**, a **global** é implicitamente declarada como local.

Voltemos agora ao exemplo do tópico Procedimentos e funções. Vamos detalhar as funções func1(x) e func2(x). Acompanhe!

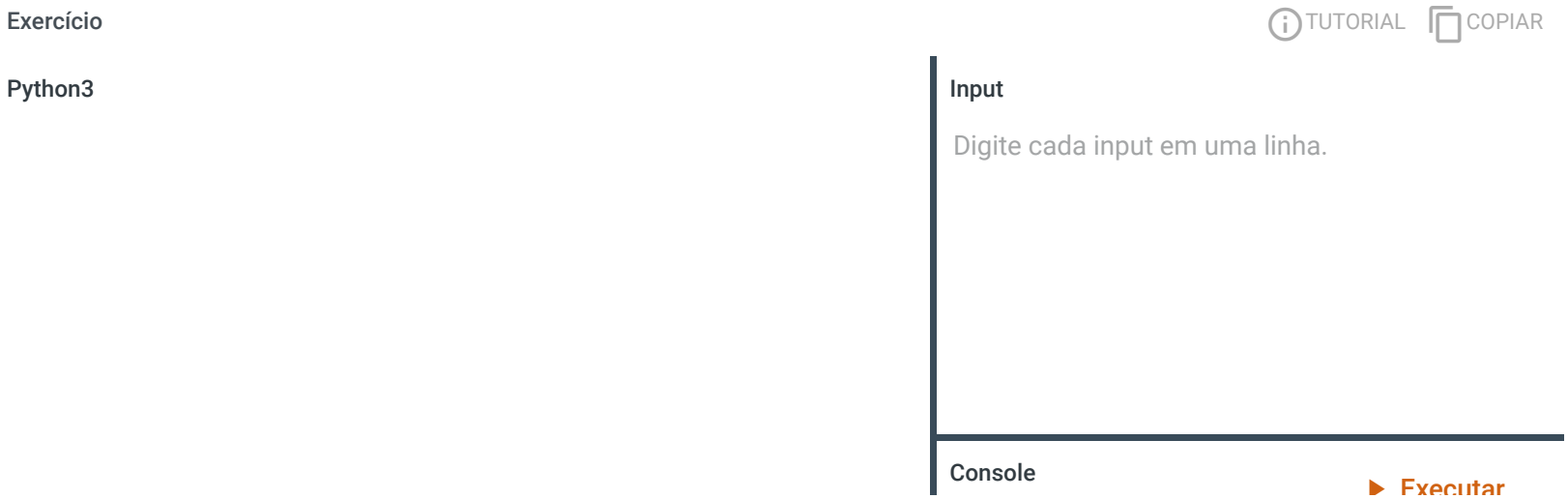
1. As linhas 1, 2 e 3 definem a função func1(x), que recebe o parâmetro x, mas tem uma variável local de nome x, cujo valor atribuído é 10.
2. Analogamente, a função func2(x) – definida nas linhas 6, 7 e 8 – recebe o parâmetro x e tem uma variável de mesmo nome, mas com valor atribuído

20.

3. O programa principal tem uma variável **global** de mesmo nome x, cujo valor atribuído é 0, na linha 11.
4. Veja que as chamadas às funções func1(x) e func2(x) ocorrem nas linhas 12 e 13, quando a variável x **global** já recebeu o valor 0. Porém, ao ser executada, cada uma dessas funções tem a própria variável local a quem todas as referências internas são feitas.
5. Mesmo com a variável **global** tendo valor nulo, cada variável local das funções func1(x) e func2(x) tem o próprio valor. Não há alterações na variável **global** mesmo com as atribuições das linhas 2 e 7.

Para alterar a variável **global** x, seria necessário explicitar, dentro de cada função, que o nome x é referente a ela. Isso pode ser feito com a palavra reservada **global**. Além de explicitar a referência à variável **global**, as funções func1(x) e func2(x) não recebem mais os parâmetros de mesmo nome, já que fazem referência à variável **global**.

Veja como ficaria o nosso exemplo com essa pequena alteração no código anterior (trata-se do código 14 no arquivo disponibilizado).



Observando a execução do código anterior, percebe-se que o **print()** do programa principal está na linha 16, depois da chamada à função func2(x). Dessa forma, a variável **global** x, alterada na execução da func2(x), fica com o valor 20 quando a execução volta ao programa principal.

Subprogramas aninhados

Em Python (e na maioria das linguagens funcionais), é permitido aninhar subprogramas.

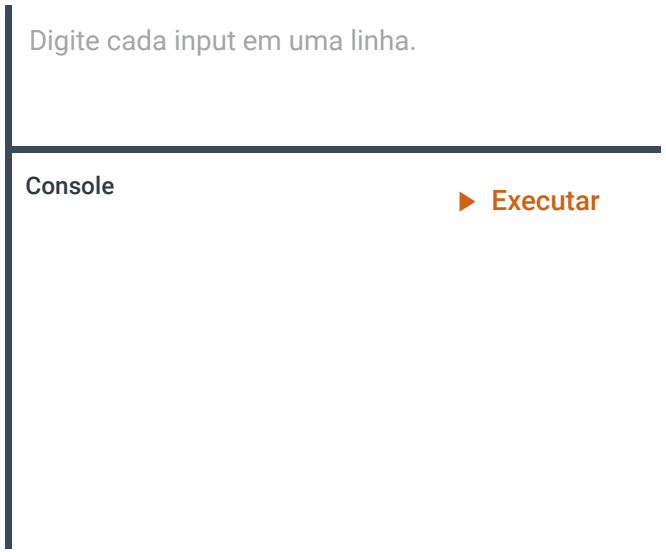
Veja o exemplo a seguir (código 15 no arquivo disponibilizado neste conteúdo). No campo Input do emulador, digite o valor que será atribuído à variável **dist** e clique em Executar.

Exercício

TUTORIAL COPIAR

Python3

Input



A função taximetro() tem, dentro de sua definição, a definição de outra função denominada **calculaMult**(). Na linha 7, a função **calculaMult**() é chamada, e o seu retorno é armazenado na variável **multiplicador**.

Métodos de passagens de parâmetros

São as maneiras existentes para transmiti-los ou recebê-los dos subprogramas chamados. Os parâmetros podem ser passados principalmente pelos seguintes métodos:

Valor

O parâmetro formal funciona como uma variável local do subprograma, sendo inicializado com o valor do parâmetro real. Dessa maneira, não ocorre uma alteração na variável externa ao subprograma caso ela seja passada como parâmetro.

Referência

Em vez de passar o valor do parâmetro real, é transmitido um caminho de acesso (normalmente, um endereço) para o subprograma chamado. Isso fornece o caminho de acesso para a célula que armazena o parâmetro real. Assim, o subprograma chamado pode acessar esse parâmetro na unidade de programa chamadora.

O método de passagem de parâmetros de Python é chamado de passagem por atribuição. Como todos os valores de dados são objetos, toda variável é uma referência para um objeto.

Resumindo

Ao se estudar a orientação a objetos, fica mais clara a diferença entre a passagem por atribuição e a passagem por referência. Por enquanto, podemos entender que a passagem por atribuição é uma passagem por referência, pois os valores de todos os parâmetros reais são referências.

Atividade 2

Considere o seguinte trecho de um programa escrito em Python:

Python

O que acontecerá quando o usuário tentar executar esse programa?

A Ocorrerá um erro, e o programa não será executado.

B Ocorrerá um erro durante a execução.

C Será impresso na tela: 0 10 0.

D Será impresso na tela: 0 10 10.

E 10 10 10.

Parabéns! A alternativa C está correta.

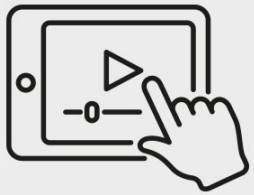
A variável x da linha 6 é global. Mas, como existe outra variável com o mesmo nome dentro da função func1() – na linha 2, apenas dentro da função func1(), x vale 10 –, chamamos essa variável de local. Assim, o print da linha 7 recebe o valor da variável global (0). A execução da linha 8 chama a função func1(), que imprime o valor de x válido dentro dela (10). Em seguida, a execução do programa sai da função func1() e o print da linha 9 volta a enxergar a variável global x, cujo valor é 0.

Parâmetros, procedimentos e funções na prática

As regras de validação de um CPF (Cadastro de Pessoa Física) no Brasil são definidas pela Receita Federal e consistem em um conjunto de procedimentos para garantir a integridade e autenticidade do número de identificação de cada cidadão.

No vídeo a seguir, praticaremos passagem de parâmetros, procedimentos e funções, utilizando a linguagem de programação Python. Será proposto um exercício para criar uma função que valida um CPF. Acompanhe!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Roteiro de prática

Aqui estão as principais regras de validação de um CPF. Confira!

Número de dígitos

Um CPF é composto por 11 dígitos numéricos.

Dígitos verificadores

Os dois últimos dígitos do CPF são chamados de dígitos verificadores, e são usados para garantir a integridade do número. Eles são calculados com base nos 9 primeiros dígitos do CPF.

Cálculo dos dígitos verificadores

Os dígitos verificadores são calculados usando o algoritmo de módulo 11. O primeiro dígito verificador é calculado a partir dos 9 primeiros dígitos do CPF. O segundo dígito verificador é calculado a partir dos 9 primeiros dígitos do CPF, incluindo o primeiro dígito verificador.

Validação dos dígitos verificadores

Após calcular os dígitos verificadores, eles são comparados com os dígitos originais do CPF. Se os dígitos verificadores calculados coincidirem com os dígitos verificadores originais, o CPF é considerado válido. Caso contrário, é considerado inválido.

Exceções

CPFs com todos os dígitos iguais (ex.: 111.111.111-11) ou CPFs com padrões específicos que foram anulados pela Receita Federal.

Para realizar o exercício proposto, siga o passo a passo.

1. Crie um arquivo fonte chamado validacpf.
2. Crie uma função que:
    2.1. Remova os caracteres não numéricos.
    2.2. Valide a quantidade de dígitos do CPF.
    2.3. Valide se todos os dígitos não são iguais.
    2.4. Calcule o primeiro dígito verificador e o valide.
    2.5. Calcule o segundo dígito verificador e o valide.
3. Teste a função passando um CPF como parâmetro.

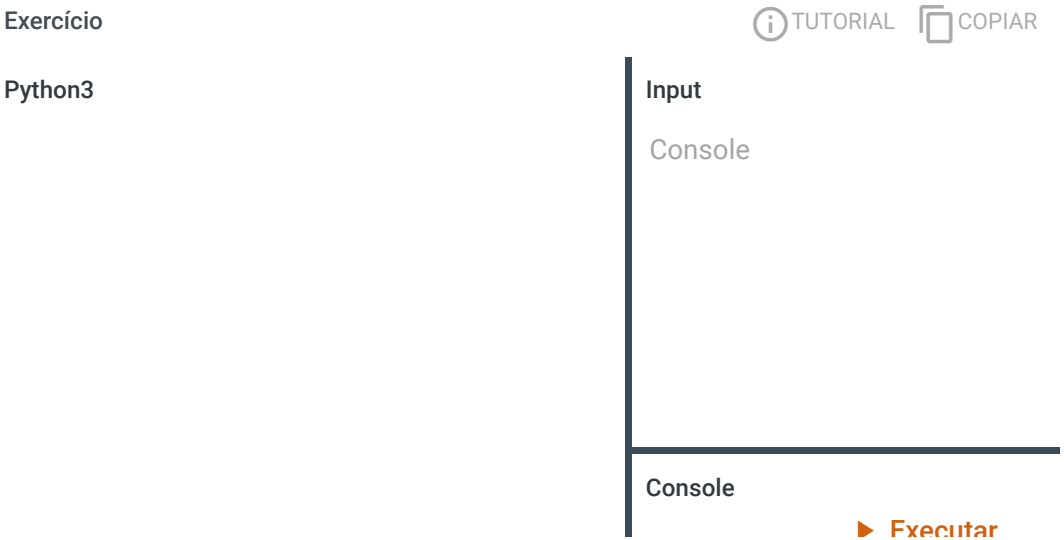
Confira o código fonte gerado no vídeo.

Código fonte

Python

Atividade 3

Escreva uma função que busca de forma iterativa o maior elemento em uma lista de números inteiros.



Chave de resposta

Veja a seguir o código da solução da atividade.

Python

Funções recursivas e docstrings

Vamos explorar as funções recursivas e aprender a criar funções que chamam a si mesmas para resolver problemas de maneira eficiente. Descubra como aplicar esse conceito em algoritmos complexos e desbloqueie novas possibilidades na programação.

Assista ao vídeo e acompanhe a utilização de funções recursivas na linguagem de programação Python, abordando temas como função recursiva fatorial, sequência de Fibonacci e documentação de funções (docstrings).

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.

Recursividade

Uma função recursiva é aquela que chama a si mesma. A seguir, veja um exemplo de função regressiva() (código 16 do arquivo disponibilizado).

Python

Na implementação da função regressiva(), tendo x como parâmetro, ela mesma é chamada com o parâmetro x – 1.

Vamos analisar a chamada regressiva(2)!

Ao chamar a regressiva(2)

O valor 2 é exibido na tela pela linha 2 e ocorre uma nova chamada da função regressiva() na linha 3 com o parâmetro 1. Vamos continuar com esse caminho de execução da regressiva(1).

Ao chamar a regressiva(1)

O valor 1 é exibido na tela pela linha 2 e ocorre uma nova chamada da função regressiva() na linha 3 com o parâmetro 0.

Ao chamar a regressiva(0)

O valor 0 é exibido na tela e ocorre uma nova chamada da função regressiva com o parâmetro -1 e assim sucessivamente.

Conceitualmente, a execução será repetida indefinidamente até que haja algum erro por falta de memória. Perceba que não definimos adequadamente uma condição de parada para a função regressiva(), o que leva a esse comportamento ruim.

Em Python, o interpretador pode interromper a execução indefinida, mas essa não é uma boa prática. Uma forma de contornar esse problema é definir adequadamente uma condição de parada como a do exemplo a seguir (código 17 do arquivo disponibilizado neste texto). Observe!

Python

Uma função recursiva que termina tem:

1. Um ou mais casos básicos que funcionam como condição de parada da recursão.
2. Uma ou mais chamadas recursivas que têm como parâmetros valores mais próximos do(s) caso(s) básico(s) que o ponto de entrada da função.

Alguns exemplos clássicos de funções que podem ser implementadas de forma recursiva são o cálculo do fatorial de um inteiro não negativo e a sequência de Fibonacci, que serão explorados a seguir.

A função recursiva fatorial

A função matemática fatorial de um inteiro não negativo n é calculada pela seguinte fórmula:

$$n! = \begin{cases} 1, \text{ se } n = 0 \text{ ou } n = 1 \\ n \cdot (n-1) \cdot (n-2) \ldots 3.2.1, \text{ se } n \geq 2 \end{cases}$$

Além disso, ela pode ser definida recursivamente pela fórmula a seguir.

$$n! = \begin{cases} 1, \text{ se } n = 0 \text{ ou } n = 1 \\ n \cdot [(n-1)!], \text{ se } n \geq 2 \end{cases}$$

Observe agora uma implementação recursiva da função fatorial em Python (trata-se do código 18 no arquivo disponibilizado).

Python

Vale ressaltar que a função fatorial também poderia ter sido implementada de forma não recursiva, como é mostrado a seguir (código 19 no arquivo disponibilizado).

Python

Porém, neste tópico, o intuito principal é explorar a recursividade.

A sequência de Fibonacci

Observe a sequência de Fibonacci: 1, 1, 2, 3, 5, 8, 13, 21... Os dois primeiros termos são 1; a partir do 3º termo, cada termo é a soma dos dois anteriores.

Veja uma possível implementação recursiva de função que determina o n-ésimo termo da sequência de Fibonacci (no arquivo disponibilizado, trata-se do código 20).

Python

Analisaremos o código a seguir.

1. A linha 2 traz as condições de parada.
2. A linha 5 traz as chamadas recursivas para calcular os dois termos anteriores da sequência.

Documentação de funções – Docstrings

Em Python, é possível definir uma **string** que serve como documentação de funções definidas pelo desenvolvedor. Ao chamar o utilitário **help**() passando como parâmetro a função desejada, essa **string** é exibida.

Verifique o código (no arquivo disponibilizado, trata-se do código 21) e a sua saída.

Python

Veja agora a exibição da docstring da função fibo().

Python

Analisaremos o código a seguir.

1. No código “Docstring da função fibo()”, a linha 2 mostra a declaração da docstring.
2. A linha 8 exibe a impressão na tela da chamada **help**(fibo). No código “Exibição da docstring da função fibo()”, está o resultado da execução desse programa.

Atividade 4

Considere o seguinte trecho de um programa com uma implementação de função recursiva escrito em Python.

Python

Quando o usuário tentou executar esse programa, houve um erro. Qual é a causa?

A Na linha 2, o if está escrito de maneira errada.

B A função não tem condição de parada.

C A função está sem retorno.

D A função não poderia ter sido definida com uma chamada a ela própria.

E Na linha 3, o return deve ter a mesma identação do if.

Parabéns! A alternativa B está correta.

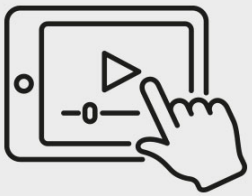
A função é recursiva, mas não apresenta parada. Ao ser chamada com o parâmetro 1, o if da linha 2 tem condição verdadeira. Então ocorre a chamada a rec(0). Mas rec(0) não é definido; assim, ocorrerá a chamada a rec(-1) - e assim sucessivamente.

Funções recursivas na prática

O problema das Torres de Hanói é um quebra-cabeça matemático e de lógica que envolve a movimentação de discos entre três pinos ou torres. Escreva uma função em Python para resolver as Torres de Hanói de forma recursiva.

Neste vídeo, realizaremos um exercício prático para solução recursiva das Torres de Hanói. Confira!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Roteiro de prática

Torres de Hanói

O problema envolve uma base de pinos verticais, em que os discos estão inicialmente empilhados em ordem crescente de tamanho em um dos pinos, com o disco menor no topo e o disco maior na base.

Torres de Hanói.

O objetivo do problema é mover todos os discos da torre de origem para a torre de destino, usando a torre auxiliar, de forma que os discos permaneçam empilhados em ordem crescente de tamanho em todos os momentos e que apenas um disco seja movido por vez. Além disso, em nenhum momento um disco maior pode ser colocado sobre um disco menor.

As regras básicas do problema são:

1

Apenas um disco pode ser movido por vez.

2

Cada movimento consiste em retirar o disco do topo de uma das torres e colocá-lo no topo de outra torre.

3

Um disco maior nunca pode ser colocado sobre um disco menor.

Agora, para realizar o exercício proposto, execute os passos a seguir.

1. Crie um arquivo de código fonte de Python Hanói recursivo.
2. Crie uma função para processar as torres chamando outra função para movimentar os discos.
3. Crie uma função para exibir o conteúdo das torres a cada passo.

4. Chame a função de forma recursiva.
5. Para iniciar o processamento, defina a quantidade de discos, crie as torres e chame para execução a função principal.

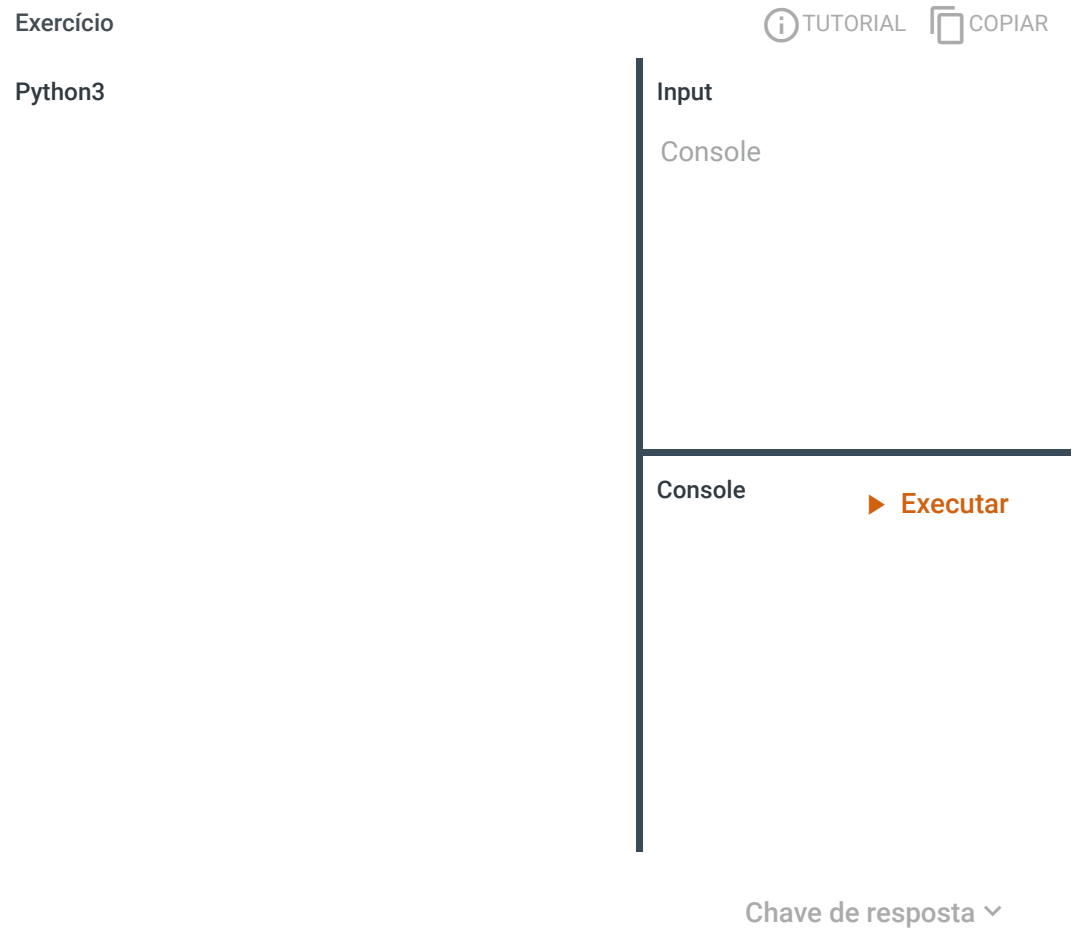
Confira o código fonte gerado no vídeo.

Código fonte

Python

Atividade 5

Escreva uma função que busca recursivamente o maior elemento em uma lista de números inteiros.



Veja a seguir o código da solução da atividade.

Python

3 - Bibliotecas em Python

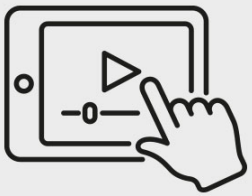
Ao final deste módulo, você será capaz de identificar o uso correto de recursos de bibliotecas em Python.

Importação de funções e módulos

Vamos explorar a importação de funções e módulos em Python! Você aprenderá a importar bibliotecas nativas da linguagem e a utilizar funções importadas em seu programa. Isso ampliará sua capacidade de desenvolvimento e facilitará a reutilização de código.

Neste vídeo, falaremos sobre a importação de funções e módulos na linguagem de programação Python, biblioteca padrão Python e de como usar uma função de módulo importado. Confira!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Biblioteca padrão Python

Python oferece, em seu núcleo, algumas funções que já utilizamos, como **print()** e **input**(), além de classes, como int, float e **str**. Logicamente, o núcleo da linguagem Python disponibiliza muitas outras funções (ou métodos) e classes além das citadas. Mas, ainda assim, ele é pequeno, tendo o objetivo de simplificar o uso e ganhar eficiência.

Para aumentar a disponibilidade de funções, métodos e classes, o desenvolvedor pode usar a biblioteca padrão Python. Neste material, apresentaremos alguns dos principais recursos dessa biblioteca e a forma de utilizá-los.

Logomarca Python em tela de computador.

A biblioteca padrão Python (Python Standard Library) consiste em milhares de funções, métodos e classes relacionados a determinada finalidade e organizados em componentes chamados de módulos. Há mais de 200 módulos que dão suporte, entre outras coisas, a:

1. Operações matemáticas.
2. Interface gráfica com o usuário (GUI).
3. Funções matemáticas e geração de números pseudoaleatórios.

Lembre-se dos conceitos de classes e objetos, pois eles são os principais conceitos do paradigma de programação orientada a objeto.

> As classes são fábricas, que podem gerar instâncias chamadas objetos.

Uma classe Pessoa, por exemplo, pode ter como atributos nome e CPF. Ao gerar uma instância de Pessoa com nome João da Silva e CPF 000.000.000-00, há um objeto.

Dica

Para compreender mais os conceitos de classe e objeto, pesquise sobre o paradigma orientado a objeto.

Como usar uma função de módulo importado

Para usar as funções e os métodos de um módulo, são necessários dois passos. Acompanhe!

1

Fazer a importação do módulo desejado com a instrução:

import nome_modulo

2

Chamar a função desejada, precedida do nome do módulo, com a instrução:

nome_modulo.nome_funcao(paramêtros)

Como exemplo, vamos importar o módulo **math** (dedicado a operações matemáticas) e calcular a raiz quadrada de 5 por meio da função **sqrt**(). Clique em Executar e observe o uso do **math** no próximo código (trata-se do código 22 no arquivo disponibilizado).

Exercício

TUTORIAL COPIAR

Python3

Input

Digite cada input em uma linha.

Console

Executar

A partir desse ponto, serão apresentados os principais aspectos dos módulos a seguir.

math

Usado para operaçoes matematicas.

random

Usado para gerar números pseudoaleatórios.

smtplib

Usado para permitir envio de e-mails.

time

Usado para implementar contadores temporais.

tkinter

Usado para desenvolver interfaces gráficas.

Atividade 1

Sabemos que é possível importar módulos e chamar funções desses módulos em Python. Considere o módulo math, que oferece diversas funções matemáticas. Uma dessas funções é a ceil(x), que retorna o menor inteiro maior ou igual a x. Suponha que um estudante queira usar uma variável n, que recebe o valor 5.9, e, em seguida, imprimir na tela o menor inteiro maior ou igual a ela. O código correto é:

A

```
import math
n = 5.9
print(ceil(n))
```

B

```
import math
n = 5.9
```

```
math.ceil(n)
print(n)
```

C

```
import math
n = 5.9
print(ceil.math(n))
```

D

```
import math
n = 5.9
print(math.ceil(n))
```

E

```
import math
n = 5.9
ceil.math(n)
print(n)
```

Parabéns! A alternativa D está correta.

Para utilizar uma função de um módulo em um projeto do Python, temos primeiro que importar a biblioteca ou pelo menos a função desejada para o projeto.

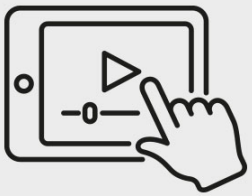
A seguir, para podermos utilizá-la, devemos chamar a função utilizando a sintaxe: nomemodulo.nomefunção(parâmetros)

Módulos básicos nativos do Python

Vamos explorar mais detalhadamente os módulos básicos nativos do Python! Veremos uma introdução abrangente aos módulos incorporados, como os de matemática, tempo e funções para números inteiros e sequências.

Neste vídeo, abordaremos a utilização de módulos básicos nativos da linguagem de programação Python. Falaremos sobre módulo **math** e **random**, distribuições de valores reais, funções para números inteiros e para sequências. Acompanhe!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Módulo **math**

Este módulo provê acesso a funções matemáticas de argumentos reais. As funções não podem ser usadas com números complexos.

Listaremos agora algumas das funções do módulo **math**. Observe!

Função	Retorno
sqrt(x)	Raiz quadrada de x
ceil(x)	Menor inteiro maior ou igual a x
floor(x)	Maior inteiro menor ou igual a x
cos(x)	Cosseno de x
sin(x)	Seno de x
log(x, b)	Logaritmo de x na base b
pi	Valor de Pi (3.141592...)
e	Valor de e (2.718281...)

Principais funções do módulo math
Humberto Henriques de Arruda

Para mais informações sobre o módulo math, visite a biblioteca Python.

Módulo random

Este módulo implementa geradores de números pseudoaleatórios para várias distribuições. Conheça!

Números inteiros

- Existe uma seleção uniforme a partir de um intervalo.

Sequências

- Existe uma seleção uniforme de um elemento aleatório.
- Existe uma função para gerar uma permutação aleatória das posições na lista.
- Existe uma função para escolher aleatoriamente sem substituição.

Distribuições de valores reais

A tabela a seguir mostra algumas das principais funções disponíveis para distribuições de valores reais no módulo random. Confira!

Função	Retorno
random()	Número de ponto flutuante no intervalo (00,0, 1,0)
uniform(a, b)	Número de ponto flutuante N tal que a ≤ N ≤ b
gauss(mu, sigma)	Distribuição gaussiana. **mu** é a média e **sigma** é o desvio padrão.
normalvariate(mu, sigma)	Distribuição gaussiana. **mu** é a média e **sigma** é o desvio padrão.

Principais distribuições de valores reais.
Humberto Henriques de Arruda.

Funções para números inteiros

Veja algumas das principais funções disponíveis para inteiros no módulo random.

Função	Retorno
randrange(stop)	Um elemento selecionado aleatório de **range**(0, stop), mas sem construir um objeto **range**.

Função	Retorno
randrange(start, stop, [step])	Um elemento selecionado aleatório de **range**(start, stop, step), mas sem construir um objeto **range**.
randint(a, b)	Número inteiro N tal que $a \leq N \leq b$

Principais funções do módulo random para inteiros.
Humberto Henriques de Arruda.

Funções para sequências

A tabela a seguir mostra algumas das principais funções disponíveis para sequências no módulo random. Conheça!

Função	Retorno
choice(seq)	Elemento aleatório de uma sequência não vazia **seq**.
shuffle(x[, random])	Embaralha a sequência **x** no lugar.

Função	Retorno
sample(pop, k)	Uma sequência de tamanho **k** de elementos escolhidos da população **pop**, sem repetição. Usada para amostragem sem substituição.

Principais funções do módulo random para sequências.
Humberto Henriques de Arruda.

Para mais informações sobre o módulo random, visite a biblioteca Python.

Atividade 2

O Python disponibiliza uma série de bibliotecas, algumas padrão da instalação e outras que podem ser instaladas pelo programador. Cada biblioteca fornece diversas funções que permitem aumentar muito as capacidades da linguagem.

Em relação às bibliotecas padrão do Python, é correto afirmar que no Python

A só é permitida a utilização dos módulos contidos na biblioteca padrão Python.

B há o módulo de interface gráfica **tkinter**, que não permite a criação de janelas com botões.

C há o módulo de interface de e-mails **smtplib**, que não permite envio de e-mails por servidores gratuitos.

D há o módulo de operações matemáticas **math**, que não permite operações com números complexos.

E o módulo math é usado para implementar geradores de números aleatórios.

Parabéns! A alternativa D está correta.

O módulo math do Python é uma biblioteca padrão que fornece funções matemáticas básicas para operações de ponto flutuante. Ele inclui funções para cálculo de raiz quadrada, exponenciação, logaritmos, trigonometria, manipulação de números inteiros e constantes matemáticas, como “Pi” e “e”.

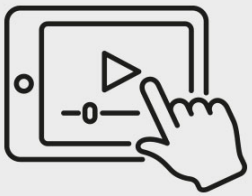
O math é amplamente utilizado devido à sua eficiência e facilidade de uso para operações matemáticas comuns, entretanto, ele não suporta operações com números complexos. As funções no módulo math são projetadas para trabalhar exclusivamente com números reais (float e int). Se você tentar passar um número complexo para uma função do módulo math, ele retornará um erro.

Mais módulos nativos do Python

Aqui, veremos o módulo SMTPLIB para enviar e-mails de forma eficiente, o módulo time para manipular datas e horários com precisão, e o módulo tkinter para criar interfaces gráficas interativas.

Para compreender a utilização dos módulos básicos complementares nativos da linguagem de programação Python, assista ao vídeo!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Módulo SMTPLIB

Define um objeto de sessão do cliente SMTP que pode ser usado a fim de enviar e-mail para qualquer máquina da internet com um serviço de processamento SMTP ou ESMTP. O exemplo a seguir vai permitir que você envie um e-mail a partir do servidor SMTP do Gmail.

Como a Google não permite, por padrão, realizar o login com a utilização do smtplib por considerar esse tipo de conexão mais arriscada, será necessário alterar uma

configuração de segurança. Para resolver isso, siga as instruções a seguir.

1. Acesse sua conta no Google.
2. Depois acesse Segurança.
3. Acesso a app menos seguro. (Atenção! Em algumas contas, os nomes estarão escritos no idioma inglês - *allow less secure apps*).
4. Mude para "ativada" a opção de "Permitir aplicativos menos seguros".

Para fazer seu primeiro envio, crie um programa no seu projeto. O código a seguir mostra as importações necessárias para o envio. Confira!

Python

O código seguinte mostra a criação da mensagem com o corpo e seus parâmetros.

Python

Agora, vamos analisar o código.

1. A linha 2 mostra a criação de um objeto de mensagem.
2. A linha 3 exibe o corpo da mensagem em uma **string**.
3. As linhas de 6 a 9 devem ser preenchidas com os valores adequados para que seu programa seja executado com sucesso.
4. A linha 12 anexa o corpo da mensagem (que estava em uma **string**) ao objeto msg.

O próximo código mostra os passos necessários para o próprio envio. Conheça!

Python

A seguir, faremos análise do código.

1. As linhas 2 e 3 mostram a criação do servidor e a sua conexão no modo TLS.
2. A linha 6 mostra o login na conta de origem do e-mail.
3. A linha 9 representa o envio propriamente dito.
4. A linha 12 exibe o encerramento do servidor.

O código referente ao programa para o envio de e-mail a partir do servidor SMTP do Gmail encontra-se no arquivo disponibilizado neste conteúdo (código 28).

Dica

Para mais informações sobre o módulo **smtplib**, visite a biblioteca Python.

Módulo time

Provê diversas funções relacionadas a tempo. Também pode ser útil conhecer os módulos datetime e calendar.

Esta tabela aponta algumas das principais funções disponíveis no módulo **time**, confira!

Função	Retorno
time()	Número de segundos passados desde o início da contagem (epoch). Por padrão, o início é 00:00:00 do dia 1 de janeiro de 1970.
ctime(segundos)	Uma **string** representando o horário local, calculado a partir do número de segundos passado como parâmetro.
gmtime(segundos)	Converte o número de segundos em um objeto **struct_time** descrito a seguir.
localtime(segundos)	Semelhante à **gmtime()**, mas converte para o horário local.

Função	Retorno
sleep(segundos)	A função suspende a execução por determinado número de segundos.

Principais funções do módulo time.
Humberto Henriques de Arruda.

O código a seguir (código 23 no arquivo disponibilizado) mostra um exemplo de chamada das funções **time()** e **ctime()**.

Exercício

TUTORIAL COPIAR

Python3

Input

Digite cada input em uma linha.

Console

Executar

Veremos a seguir o resultado.

1. A variável x recebe o número de segundos desde 00:00:00 de 01/01/1970 pela função time(). Ao executar ctime(x), o número de segundos armazenado em x é convertido em uma **string** com o horário local.
2. A classe time.struct_time gera objetos sequenciais com valor de tempo retornado pelas funções gmtime() e localtime(). São objetos com interface de tupla nomeada: os valores podem ser acessados pelo índice e pelo nome do atributo.

Aparecem os seguintes valores:

Índice	Atributo	Valores
0	tm_year	Por exemplo, 2020
1	tm_mon	range [1, 12]
2	tm_mday	range [1, 31]
3	tm_hour	range [0, 23]
4	tm_min	range [0, 59]
5	tm_sec	range [0, 61]

Índice	Atributo	Valores
6	tm_wday	range [0, 6] Segunda-feira é 0
7	tm_yday	range [1, 366]
8	tm_isdst	0,1 ou -1
N/A	tm_zone	Abreviação do nome da timezone

Principais funções do módulo time.
Humberto Henriques de Arruda.

Para mais informações sobre o módulo time, visite a biblioteca Python.

Módulo tkinter

O pacote tkinter é a interface Python padrão para o Tk GUI (interface gráfica com o usuário) toolkit. Na maioria dos casos, basta importar o próprio tkinter, mas diversos outros módulos estão disponíveis no pacote.

> **A biblioteca tkinter permite a criação de janelas com elementos gráficos, como a entrada de dados e botões, por exemplo.**

O exemplo a seguir vai permitir que você crie a primeira janela com alguns elementos. Para isso, crie um programa novo no seu projeto. O código adiante mostra a criação da sua primeira janela, ainda sem qualquer elemento gráfico.

Python

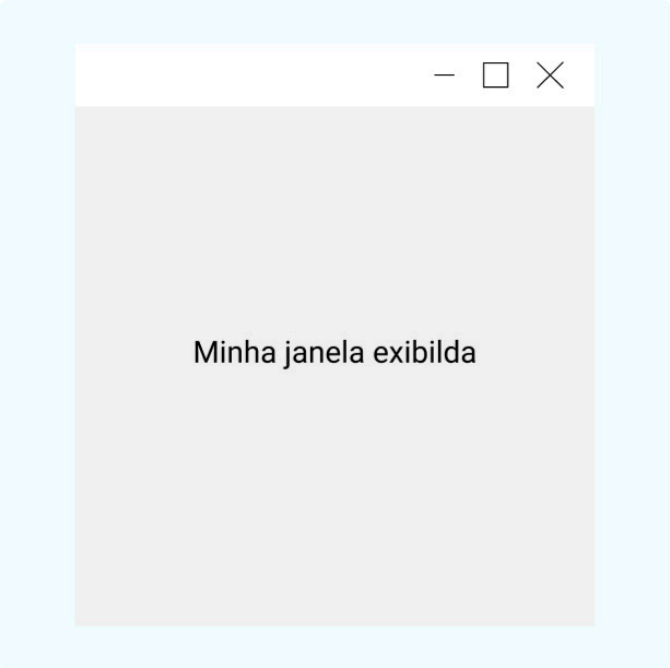
Agora, veremos a análise do código.

1. A linha 1 mostra a importação de todos os elementos disponíveis em tkinter. O objeto janelaPrincipal é do tipo Tk. Um objeto Tk é um elemento que representa a janela GUI. Para que essa janela apareça, é necessário chamar o método **mainloop**();
2. Para exibir textos, vamos usar o elemento Label. O próximo código mostra as linhas 4 e 5, com a criação do elemento e o seu posicionamento. O tamanho padrão da janela é 200 X 200 pixels, com o canto superior esquerdo de coordenadas (0,0) e o inferior direito de coordenadas (200,200).

Observe agora a primeira janela com tkinter 2.

Python

Veja o resultado de sua primeira janela, apenas com o texto, na imagem a seguir.



Exibição da primeira janela com tkinter.

Vamos agora incrementar um pouco essa janela. Para isso, acrescentaremos uma imagem e um botão. A imagem precisa estar na **mesma pasta** do seu arquivo .py.

Confira o código da segunda janela com tkinter.

Python

A seguir, analisaremos o código.

1. No código, há a inserção do elemento de imagem e o botão. Nas linhas 10, 11 e 12, é feita a criação do objeto Label para conter a imagem e seu posicionamento. Observe que passamos a utilizar o método pack(), que

coloca o elemento centralizado e posicionado o mais perto possível do topo, depois dos elementos posicionados anteriormente.

2. O elemento botão é criado na linha 14 com os atributos text e command, os quais são respectivamente o texto exibido no corpo do botão e a função a ser executada quando o botão é clicado.
3. Para o funcionamento correto do botão, é preciso definir a função funcClicar(), nas linhas 3 e 4.

Agora, responda:

Para que serve a função funcClicar()?

Clique em um dos botões para responder.

Garantir que o botão possua link.

Imprimir na tela a **string** "Botão pressionado".

Verifique o resultado na imagem.

Segunda janela com tkinter exibida.

Tendo implementado sua janela, clique no botão para ver o resultado no console, como mostra a imagem a seguir.

Resultado do clique no botão da segunda janela.

O código referente ao programa gráfico com o uso do módulo Tkinter encontra-se no arquivo disponibilizado neste conteúdo (código 29).

Dica

Para mais informações sobre o tkinter, visite a biblioteca Python.

Atividade 3

O Python oferece uma gama de módulos nativos que simplificam tarefas comuns de programação, sendo um desses módulos o tkinter. Qual das seguintes opções descreve corretamente o tkinter em Python?

A Tkinter é uma linguagem de programação para desenvolvimento web.

B Tkinter é uma biblioteca gráfica para criar interfaces gráficas de usuário (GUI) em Python.

C Tkinter é uma ferramenta para análise de dados em Python.

D Tkinter é uma biblioteca para lidar com arquivos de texto em Python.

E Tkinter é uma estrutura para programação de jogos em Python.

Parabéns! A alternativa B está correta.

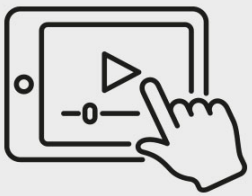
O tkinter é a biblioteca padrão do Python para criar interfaces gráficas de usuário (GUI). Ele fornece ferramentas para criar janelas, botões, caixas de entrada, rótulos e outros elementos de interface gráfica.

Módulos nativos do Python na prática

Vamos aprender agora a utilizar o módulo nativo tkinter. Para isso, crie um programa com interface gráfica em Python que receba dois números e realize a soma de ambos.

Neste vídeo, praticaremos a implementação de módulos nativos utilizando a linguagem de programação Python. Iremos desenvolver um aplicativo que utiliza o tkinter para criar uma interface gráfica. Confira!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Roteiro de prática

Execute os passos a seguir.

1. Crie um arquivo de código-fonte de Python somanumero.
2. Importe o tkinter.
3. Crie uma função para realizar a soma dos números.
4. Crie a janela.
5. Crie os widgets da janela.
6. Rode o loop principal.

Confira, agora, o código fonte gerado no vídeo.

Código fonte

Python

Atividade 4

Utilizando o PyCharm, crie um programa com interface gráfica em Python que receba dois números, compare-os e informe se o primeiro é maior, menor ou igual ao segundo.

Chave de resposta

Veja a seguir o código da solução da atividade.

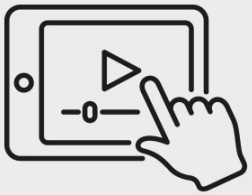
Python

Pacotes externos

Agora, vamos explorar os pacotes externos do Python! Utilizaremos o pip, a ferramenta padrão de gerenciamento de pacotes do Python, para instalar e gerenciar pacotes externos, desde bibliotecas científicas até ferramentas de desenvolvimento web. Você aprenderá a acessar e utilizar diversos recursos para aprimorar seus projetos e ampliar suas habilidades de programação, além de criar seus próprios módulos.

Neste vídeo, falaremos sobre a utilização de pacotes externos na linguagem de programação Python. Confira!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Usando pacotes externos

Além dos módulos fornecidos de forma integrada pela biblioteca padrão do Python, a linguagem possui uma grande vantagem: sua característica **open-source** permite que qualquer desenvolvedor, com o conhecimento adequado, desenvolva a própria biblioteca e os próprios módulos, os quais chamaremos, a partir de agora, de pacotes. Veremos como criar módulos mais adiante neste conteúdo.

Pessoa programadora em computador.

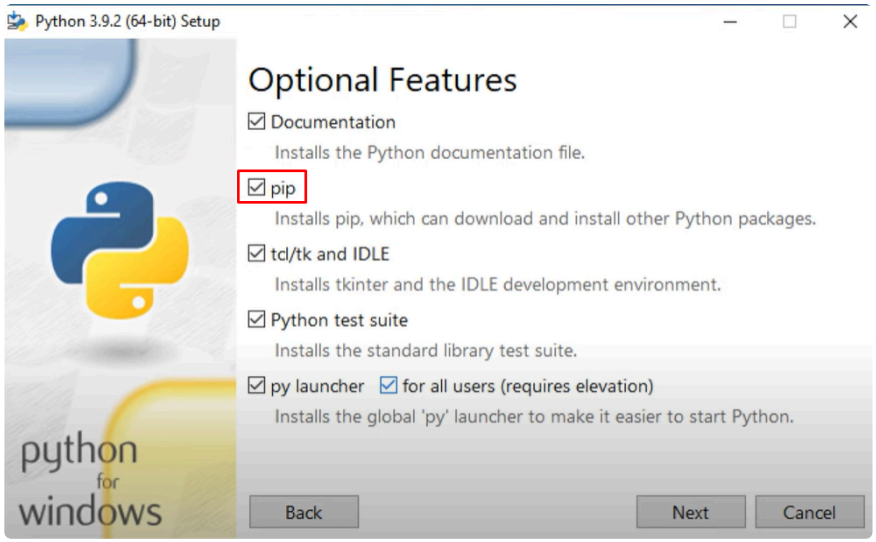
Com um pacote pronto, o desenvolvedor pode disponibilizar esse material na internet de forma que qualquer pessoa possa aproveitar o código implementado. Isso foi um dos fatores que fez com que o Python se tornasse uma das linguagens mais utilizadas no mundo atualmente.

Comentário

Como forma de facilitar a distribuição dos pacotes entre os usuários, existe um grupo dentro da comunidade Python que mantém o chamado Python package index, ou PyPI, um grande servidor no qual os desenvolvedores podem hospedar os seus pacotes, contendo bibliotecas e/ou módulos, para que sejam baixados e instalados por outras pessoas.

É possível acessar o PyPI por meio do **pip**, um programa que pode ser instalado com a distribuição do Python. Para isso, certifique-se de que a caixa "pip" está marcada durante a instalação, conforme ilustrado a seguir.

Veja a seguir a captura de tela da instalação do pip.



Instalação do pip.

Para verificar se a sua instalação Python incluiu o pip, procure pela pasta Scripts dentro do diretório de instalação que você escolheu para o Python. Dentro dessa pasta, localize o arquivo pip.exe.

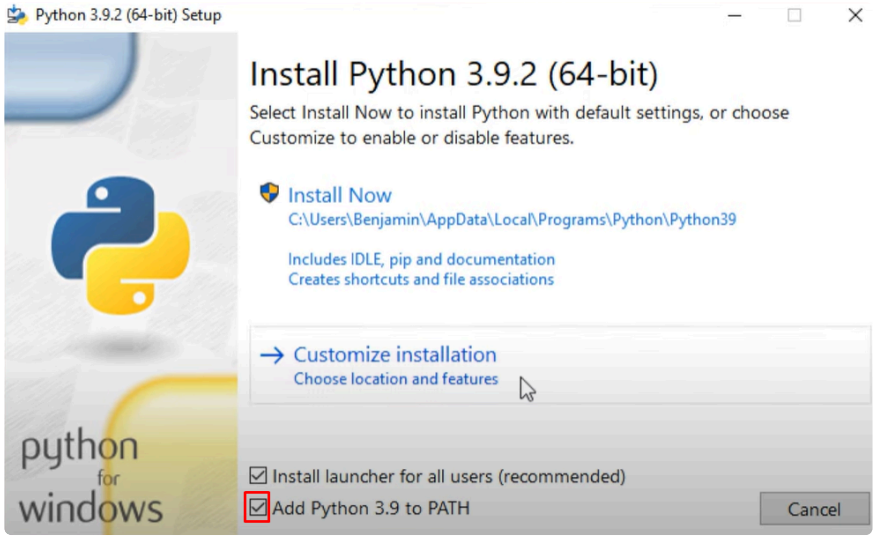
Recomendação

Se o arquivo pip não foi incluído na instalação do Python, pesquise sobre como instalá-lo. Ou tente reinstalar o Python sem esquecer de marcar a opção de incluir o pip durante o processo de instalação.

Além do pip, é necessário ter em mãos o endereço para acessar o pip dentro da variável de ambiente PATH. O processo de instalação do Python também permite incluir automaticamente o Python no seu PATH, porém, caso não tenha feito isso, siga os passos a seguir.

1. Clique na **tecla do Windows** e escreva “Editar as variáveis de ambiente para a sua conta”.
2. Na janela que abrir, procure pela variável Path, selecione-a e clique no botão **“Editar”**.
3. Clique no botão **“Novo”** e digite o endereço da sua instalação do Python (por exemplo, D:\Python).
4. Clique no botão **“Novo”** uma segunda vez e digite o endereço da pasta Scripts dentro da sua instalação do Python (por exemplo, D:\Python\Scripts).
5. Aperte **Ok** até fechar todas as janelas.

É importante que você se certifique de que a opção Add Python to PATH está marcada durante a instalação, como apontado na imagem a seguir.



Adicionando o Python ao PATH.

Com essas etapas concluídas, já conseguimos instalar nossos pacotes externos!

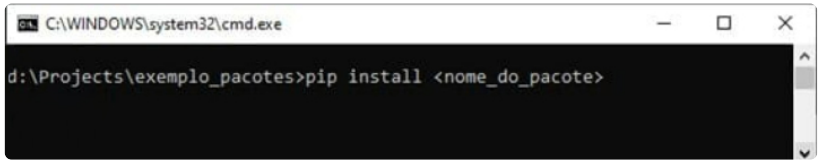
> **Quando estiver trabalhando com pacotes externos, é extremamente recomendado o uso de ambientes virtuais (em inglês, *virtual environments* ou simplesmente *virtualenvs*). Esses ambientes isolam o projeto em que você está trabalhando.**

Uma das vantagens do uso de ambientes virtuais é que você consegue saber exatamente quais pacotes externos estão sendo usados no projeto. É possível usar esses pacotes sem o uso de ambientes virtuais, porém isso pode causar uma confusão caso você tenha vários projetos Python no seu computador.

Dica

Pesquise mais sobre ambientes virtuais e configure um em cada projeto. Não é muito difícil e vai ajudá-lo a deixar o seu código mais profissional!

Para instalar um pacote externo disponível no PyPI, basta abrir o seu terminal (clique no botão do Windows e digite **“cmd”** ou **“prompt de comando”**), ativar o seu ambiente virtual (se você estiver usando um) e digitar o seguinte comando:



Instalando um pacote usando pip.

Substitua <nome_do_pacote> pelo pacote que você deseja usar. Temos inúmeros pacotes prontos à nossa disposição. Cada pacote normalmente possui um site que apresenta a sua documentação de forma similar à da documentação oficial do Python.

A seguir você encontra uma lista com alguns dos pacotes externos mais comuns e utilizados no mercado. Conheça!

Nome do módulo	Pra que serve?
numpy	Cálculos, operações matemáticas e simulações
pandas	Manipulaçao de dados
scikit-learn	Modelos de aprendizado de máquina

Nome do módulo	Pra que serve?
matplotlib	Visualização de dados
requests	Biblioteca de comandos de comunicação pelo protocolo HTTP
flask	Construção de aplicações web

Pacotes externos mais comuns e utilizados no mercado.
Humberto Henriques de Arruda.

Antes de começar o próprio módulo, é sempre recomendado pesquisar se o que você quer fazer já existe em algum pacote popular. Se existir, procure pela documentação e instale esse pacote.

O uso de módulos oriundos de pacotes externos é idêntico à utilização daqueles da biblioteca padrão. Basta, para isso, utilizar o import nome_do_modulo no seu código.

Criação do próprio módulo

Os desenvolvedores podem criar os próprios módulos de forma a reutilizar as funções que já escreveram e organizar melhor seu trabalho. Para isso, basta criar um arquivo .py e escrever nele suas funções. O arquivo do módulo precisa estar na **mesma pasta** do arquivo para onde ele será importado.

Atividade 5

Python é uma linguagem de programação de alto nível, conhecida por sua sintaxe simples e legibilidade. É uma escolha popular para uma variedade de aplicações, desde desenvolvimento web e científico até automação de tarefas e inteligência artificial.

Uma das vantagens do Python é sua vasta coleção de bibliotecas e ferramentas, que simplificam muitas tarefas comuns de programação, sendo uma dessas ferramentas essenciais o pip. O que é o pip em Python?

A O pip é um editor de texto utilizado para escrever código Python.

B O pip é um interpretador de Python usado para executar programas.

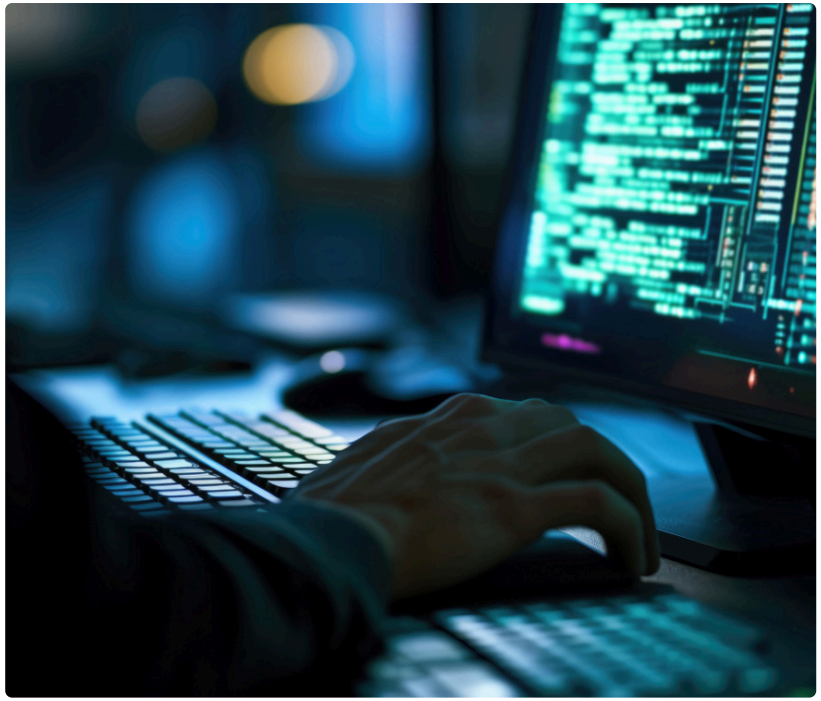
C O pip é um gerenciador de pacotes que facilita a instalação, atualização e remoção de pacotes em Python.

D O pip é uma ferramenta para criar interfaces gráficas de usuário (GUI) em Python.

E O pip é uma estrutura para desenvolvimento de jogos em Python.

Parabéns! A alternativa C está correta.

Pip é o gerenciador de pacotes padrão do Python, usado para instalar, atualizar e desinstalar pacotes de software. Ele facilita a gestão de bibliotecas e suas dependências, permitindo a adição de funcionalidades extras ao ambiente Python de forma simples e eficiente. Com pip, você pode facilmente instalar pacotes de repositórios on-line, como o Python Package Index (PyPI), e gerenciar versões específicas de pacotes. Além disso, pip permite a criação de ambientes virtuais isolados, ajudando a evitar conflitos de dependências entre projetos diferentes.



4 - Eventos em Python

Ao final deste módulo, você será capaz de reconhecer as formas de tratamento de exceções e eventos em Python.

Erros em um programa Python

Vamos agora identificar, compreender e lidar com diferentes tipos de erros: erros de sintaxe, exceções e falhas de lógica.

Neste vídeo falaremos sobre como identificar, compreender e lidar com diferentes tipos de erros na linguagem de programação Python. Acompanhe!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.

Erros e exceções

Até agora consideramos que nossos programas tiveram seu fluxo de execução normal. Neste conteúdo, vamos analisar o que acontece quando o fluxo de execução é interrompido por uma exceção, além de controlar esse fluxo excepcional.

Dois tipos básicos de erros podem acontecer em um programa em Python. Os **erros de sintaxe** são aqueles que ocorrem devido ao formato incorreto de uma instrução. Esses erros são descobertos pelo componente do interpretador Python, que é chamado analisador ou parser.

Veja exemplos nas duas imagens a seguir.

```
>>> print 'hello'
      File "<input>", line 1
            print 'hello'
                    ^
SyntaxError: Missing parenthesesin call to 'print'. Did you mean print('hello')?
```

Erro de sintaxe 1.

```
>>> lista = [1 ; 2 ; 4]
      File "<input>", line 1"
            lista = [1 ; 2 ; 4]
                        ^
SyntaxError: invalid sytax
```

Erro de sintaxe 2.

Além deles, existem os erros que ocorrem em **tempo de execução do programa**, que não se devem a uma instrução escrita errada, e sim ao fato de que o programa entrou em um estado indevido.

Elencamos os seguintes exemplos, observe!

1. A divisão por 0.
2. A tentativa de acessar um índice indevido em uma lista.
3. Um nome de variável não atribuído.
4. Um erro causado por tipos incorretos de operando.

Em cada caso, quando o programa atinge um estado inválido, é dito que o interpretador Python **levanta uma exceção**. Isso significa que é criado um objeto que contém as informações relevantes sobre o erro.

A tabela a seguir traz alguns tipos comuns de exceção. Conheça!

Exceção	Explicação
KeyboardInterrupt	Levantado quando o usuário pressiona CTRL+C, a combinação de interrupção.
OverflowError	Levantado quando uma expressão de ponto flutuante é avaliada como um valor muito grande.
ZeroDivisionError	Levantado quando se tenta dividir por 0.
IOError	Levantado quando uma operação de entrada/saída falha por um motivo relacionado a isso.
IndexError	Levantado quando um índice sequencial está fora do intervalo de índices válidos.
NameError	Levantado quando se tenta avaliar um identificador (nome) não atribuído.
TypeError	Levantado quando uma operação da função é aplicada a um objeto do tipo errado.

Exceção	Explicação
ValueError	Levantado quando a operação ou função tem um argumento com o tipo correto, mas valor incorreto.

Tipos comuns de exceção.
Humberto Henriques de Arruda.

Em Python, **as exceções são objetos**. A classe **Exception** é derivada de **BaseException**, classe base de todas as classes de exceção. BaseException fornece alguns serviços úteis para todas as classes de exceção, mas normalmente não se torna uma subclasse diretamente.

Atividade 1

Considere o seguinte trecho de um programa escrito em Python.

Python

Suponha que, durante a execução, o usuário entre com a palavra “numero” quando solicitado. Assinale a opção que mostra o resultado imediato dessa ação.

A O programa deixará de ser executado.

B Será impresso na tela Mensagem 1.

C Será impresso na tela Mensagem 2.

D Será impresso na tela Mensagem 3.

E O programa vai imprimir imediatamente o numero digitado.

Parabéns! A alternativa D está correta.

Como o usuário inseriu uma palavra, e não um número, a exceção não será do tipo ValueError nem do tipo IndexError. Assim, a cláusula except a ser executada é a da linha 8, imprimindo Mensagem 3.

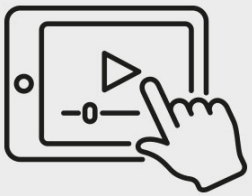
Erros e exceções na prática

A interpretação e execução de programas podem gerar erros em tempo de execução ou de sintaxe em tempo de interpretação. Vejamos agora alguns exemplos de erros e como podemos fazer a correção.

Assista ao vídeo e pratique a implementação do controle de erros e exceções utilizando a linguagem de programação Python.

Você pode obter aqui os arquivos-fonte do Python com os erros a serem vistos no vídeo.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Roteiro de prática

Para realizar o exercício, siga o passo a passo.

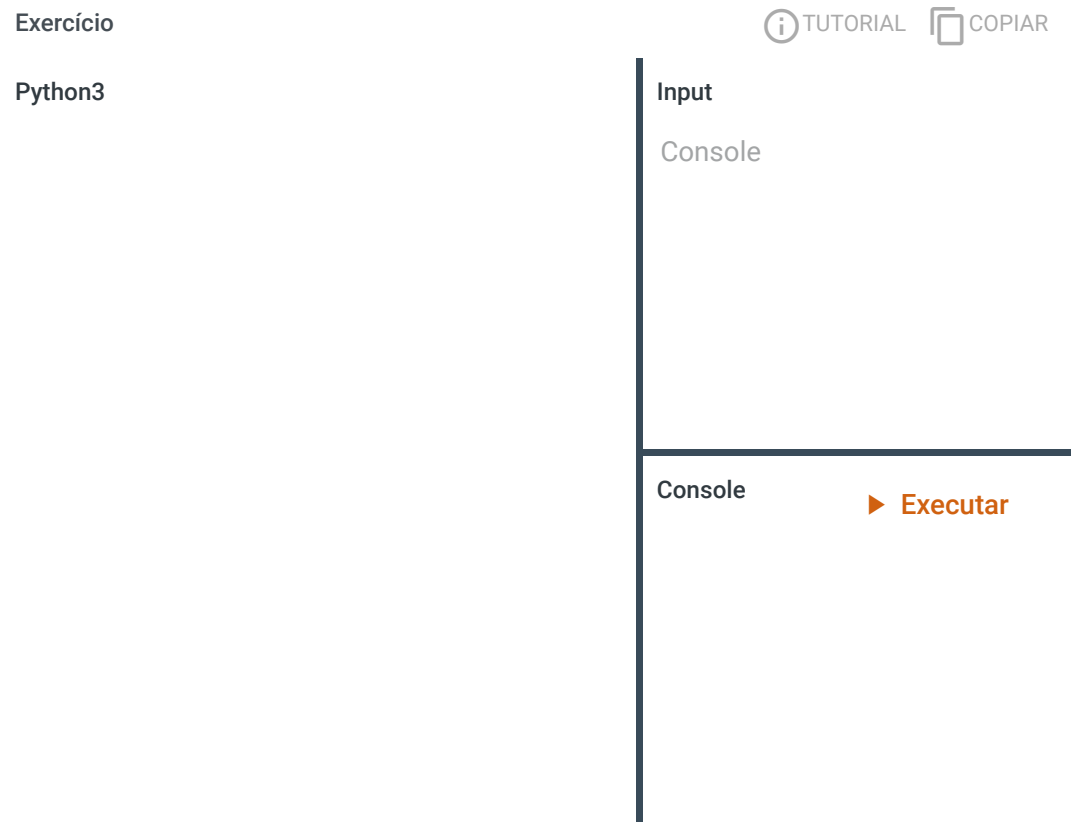
1. Abra os arquivos chamados erro1, erro2, erro3, erro4 e erro5.

2. Tente executar e veja os erros que eles contêm.
3. Realize as correções nos arquivos e execute o exercício.

Você pode obter aqui os arquivos-fonte do Python gerado no vídeo com as correções.

Atividade 2

Corrija os erros e execute o exercício a seguir.



Digite sua resposta aqui

Chave de resposta

Veja a seguir o código da solução da atividade.

Python

Tratamento de exceções e eventos

Vamos agora dominar o tratamento de exceções e eventos em Python! Você aprenderá a utilizar blocos try-except para lidar com exceções de forma eficaz. Além disso, vai descobrir como capturar e responder a eventos em seus programas, como cliques de mouse de tecla.

Neste vídeo, falaremos sobre como implementar o tratamento de exceções e eventos na linguagem de programação Python. Acompanhe!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.

Captura e manipulação de

exceções

Para evitar que os programas sejam interrompidos quando uma exceção é levantada, é possível planejar um comportamento alternativo. Assim, o programa não será interrompido e a exceção poderá ser tratada. Chamamos esse processo de **captura da exceção**.

Vamos considerar um exemplo de programa que solicita ao usuário, com a função **input**(), um número inteiro. Embora essa função trate a entrada do usuário como **string**, é possível utilizá-la em conjunto com a função eval() para que os dados inseridos sejam avaliados como números.

O próximo emulador mostra uma implementação simples desse exemplo.

Exercício

TUTORIAL COPIAR

Python3

Input

Digite cada input em uma linha.

Console

Executar

Mas o que aconteceria se o usuário digitasse uma palavra em vez de números? Faça essa experiência e digite uma palavra, como, por exemplo, dois, no emulador anterior e clique em Executar.

Veja que o programa foi encerrado com uma exceção sendo levantada. Uma forma de fazer a captura e a manipulação de exceções é usar o par de instruções try/except. Conheça!

Bloco try

É executado primeiramente. Devem ser inseridas nele as instruções do fluxo normal do programa.

Bloco except

Só será executado se houver o levantamento de alguma exceção.

Isso permite que o fluxo de execução **continue** de maneira alternativa. O emulador seguinte mostra uma implementação possível desse exemplo (código 25 no arquivo disponibilizado).

Exercício

TUTORIAL COPIAR

Python3

Input

Digite cada input em uma linha.

Console

Executar

O formato padrão de uso do par **try/except** é:

Python

O **bloco 1** representa o fluxo normal do programa. Caso uma exceção seja levantada, o bloco 2 será executado, permitindo o tratamento adequado dela. Esse **bloco 2** é chamado de manipulador de exceção.

Comentário

Em Python, o manipulador de exceção padrão é que executa o trabalho de captura da exceção caso não haja um tratamento explícito feito pelo desenvolvedor. É esse manipulador o responsável pela exibição das mensagens de erro no console.

Captura de exceções de determinado tipo

Python permite que o bloco relativo ao except só seja executado caso a exceção levantada seja de determinado tipo. Para isso, o except precisa trazer o tipo de exceção que se deseja capturar.

O emulador a seguir traz uma possível variação do exemplo anterior com a captura apenas das exceções do tipo NameError (código 26 do arquivo disponibilizado). Observe!

Exercício

TUTORIAL COPIAR

Python3

Input

Digite cada input em uma linha.

Console

Executar

Captura de exceções de múltiplos tipos

Python permite que haja diversos tratamentos para diferentes tipos possíveis de exceção. Isso pode ser feito com mais de uma cláusula except vinculada à mesma cláusula try.

O emulador seguinte mostra um exemplo de implementação da captura de exceções de múltiplos tipos (no arquivo disponibilizado, trata-se do código 27).

Exercício

TUTORIAL COPIAR

Python3

Input

Digite cada input em uma linha.

Console

Executar

Vamos analisar o exemplo!

A instrução da **linha 5** somente será executada se a exceção levantada no bloco **try** for do tipo **ValueError** e se, na instrução da **linha 7**, a exceção for do tipo IndexError.

Agora responda:

Caso a exceção seja de outro tipo, a **linha 9** será ou não executada?

Clique em um dos botões para responder.

Não será executada. Será executada.

O tratamento completo das exceções

Confira no código a seguir a forma geral completa para lidar com as exceções em Python.

As cláusulas **else** e **finally** são opcionais, como foi possível perceber nos exemplos iniciais.

Tratamento de eventos

É similar ao tratamento de exceções. Assim como no caso das exceções ocorridas em tempo de execução, podemos tratar os eventos criados por ações externas, como as interações de usuário realizadas por meio de uma interface gráfica de usuário (GUI).

Comentário

Um evento é a notificação de que alguma coisa aconteceu, como um clique de mouse sobre um elemento botão. O tratador do evento é o segmento de código que será executado em resposta à ocorrência do evento.

Atividade 3

Você está desenvolvendo um programa em Python que precisa ler números inteiros do usuário e calcular a divisão entre eles, qual é a forma correta de tratar a exceção que pode ocorrer quando o usuário tenta dividir por zero?

A

```
try:
    result = num1 / num2
except ZeroDivisionError:
    print("Não é possível dividir por zero.")
```

```
try:
    result = num1 / num2
except ZeroDivisionError:
    print("Não é possível dividir por zero.")
```

B

```
try:
    num1 = int(input("Digite o numerador: "))
    num2 = int(input("Digite o denominador: "))
    result = num1 / num2
except ValueError:
    print("Entrada inválida.")
except ZeroDivisionError:
    print("Não é possível dividir por zero.")
```

```
try:
    num1 = int(input("Digite o numerador: "))
    num2 = int(input("Digite o denominador: "))
    result = num1 / num2
except ValueError:
    print("Entrada inválida.")
except ZeroDivisionError:
    print("Não é possível dividir por zero.")
```

C

```
try:
    num1 = int(input("Digite o numerador: "))
    num2 = int(input("Digite o denominador: "))
    result = num1 / num2
except ValueError:
    print("Entrada inválida.")
except ZeroDivisionError:
```

```
try:
    num1 = int(input("Digite o numerador: "))
    num2 = int(input("Digite o denominador: "))
    result = num1 / num2
except ValueError:
    print("Entrada inválida.")
except ZeroDivisionError:
    print("Erro: divisão por zero.")
```

D

```
try:
    num1 = int(input("Digite o numerador: "))
    num2 = int(input("Digite o denominador: "))
    result = num1 / num2
except ValueError, ZeroDivisionError:
    print("Erro na entrada ou divisão por zero.")
```

```
try:
    num1 = int(input("Digite o numerador: "))
    num2 = int(input("Digite o denominador: "))
    result = num1 / num2
except ValueError, ZeroDivisionError:
    print("Erro na entrada ou divisão por zero.")
```

E

```
try:
    num1 = int(input("Digite o numerador: "))
    num2 = int(input("Digite o denominador: "))
    result = num1 / num2
except (ValueError, ZeroDivisionError):
    print("Erro na operação.")
```

```
try:
    num1 = int(input("Digite o numerador: "))
    num2 = int(input("Digite o denominador: "))
    result = num1 / num2
except (ValueError, ZeroDivisionError):
    print("Erro na operação.")
```

Parabéns! A alternativa B está correta.

Essa opção demonstra o uso adequado de múltiplas exceções. A estrutura try...except é usada para capturar exceções que podem ocorrer durante a execução do código. No exemplo:

- try: tenta executar o código dentro do bloco.
- except ValueError: captura exceções de valor invápdo.
- except ZeroDivisionError: captura exceções de divisão por zero.

Essa estrutura garante que erros na entrada do usuário e na operação de divisão sejam tratados de forma clara e apropriada.

Tratamento de exceções na prática

No programa Soma numero, que fizemos no exercício anterior, se em vez de um número, for digitada uma letra, o programa gerará uma exceção e abortará a execução.

Altere o programa de forma a obrigar que o usuário digite somente números na caixa de entrada. Além disso, altere a lógica do programa para que ele receba o dividendo, o divisor e retorne o quociente da divisão entre eles.

No vídeo a seguir, praticaremos a implementação do tratamento de exceções utilizando a linguagem de programação Python com intuito de evitar outros tipos de erro. Confira!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.

Roteiro de prática

Para realizar o exercício, siga o passo a passo proposto.

1. Crie um arquivo chamado dividenumero.
2. Altere o código da operação a ser realizada.
3. Faça o tratamento dos erros.

Confira, agora, o código fonte gerado no vídeo.

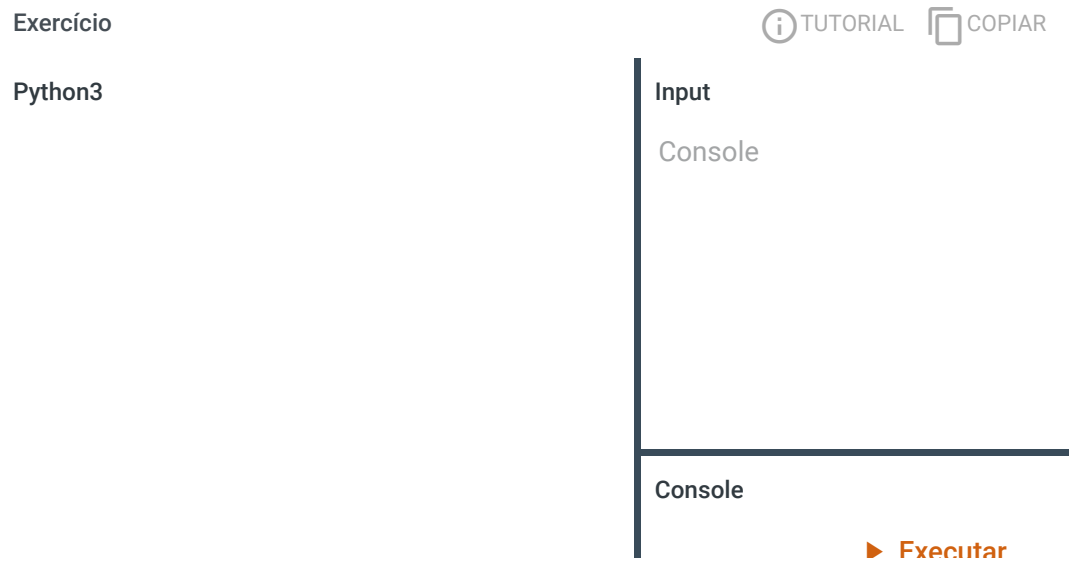
Código fonte

Python

Atividade 4

Analise o programa. Se em vez de um número, for digitada uma letra, o programa gerará uma exceção e irá abortar a execução.

Altere o programa de forma a obrigar que o usuário digite somente números na caixa de entrada.



Digite sua resposta aqui

Chave de resposta

Veja a seguir o código da solução da atividade.

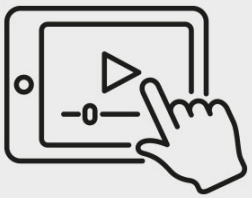
Python

Tratamento de eventos na prática

A captura de eventos é fundamental para a programação de interfaces gráficas. Com esse tipo de captura, podemos mapear o movimento do mouse, os cliques, as teclas pressionadas no teclado etc.

Neste vídeo, praticaremos a implementação do tratamento de eventos, a partir de exercícios de captura e utilizando a linguagem de programação Python. Confira!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Roteiro de prática

Para realizar a captura e o tratamento de eventos, siga as orientações.

Capturando os movimentos do mouse

Inicie realizando as etapas a seguir.

1. Crie um arquivo do Python chamado eventomouse.
2. Importe o tkinter.
3. Crie uma função para mapear as coordenadas do mouse.
4. Crie uma janela.
5. Crie um rótulo pedindo para movimentar o mouse.
6. Faça a ligação da captura do movimento com a função de mapeamento das coordenadas.
7. Rode a janela principal.

Confira, agora, o código fonte gerado no vídeo.

Código fonte

Python

Capturando o clique do botão esquerdo do mouse

Agora, siga o passo a passo.

1. Crie um arquivo do Python chamado eventomouse2.

2. Importe o tkinter.
3. Crie uma função para capturar o clique.
4. Crie uma janela.
5. Crie um rótulo pedindo para movimentar o mouse.
6. Faça a ligação do clique do mouse à função de captura.
7. Rode a janela principal.

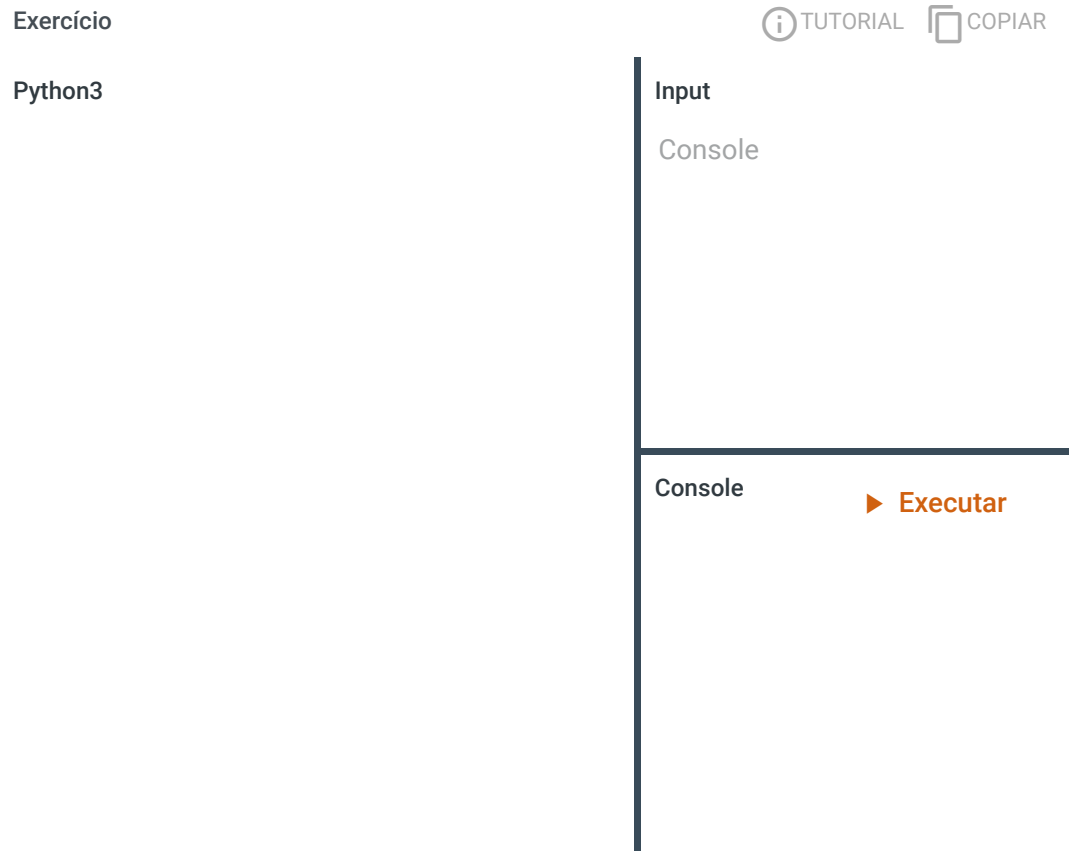
Confira, agora, o código fonte gerado no vídeo.

Código fonte

Python

Atividade 5

Altere o programa eventomouse2 para capturar o clique do botão direito ao invés do botão esquerdo.



Digite sua resposta aqui

Chave de resposta

Veja a seguir o código da solução da atividade.

Python

O que você aprendeu neste conteúdo?

- Estruturas de decisão e repetição da linguagem Python.
- Uso de parâmetros, procedimentos e funções da linguagem Python.
- Importação de funções e módulos utilizados pela linguagem Python.
- Uso de pacotes externos utilizados pela linguagem Python.
- Tratamento de exceções e eventos.

Explore +

Para ter desafios mais complexos e exercícios para treinar, recomendamos uma visita ao website Python Brasil.

Acesse também o site das bibliotecas apresentadas e busque pela documentação para que você possa conhecer todas as funcionalidades disponíveis.

Referências

PERKOVIC, L. **Introdução à computação usando Python**: um foco no desenvolvimento de aplicações. Rio de Janeiro: LTC, 2016.

SEBESTA, R. W. **Conceitos de linguagens de programação**. 11. ed. São Paulo: Bookman, 2018.

Material para download

Clique no botão abaixo para fazer o download do conteúdo completo em formato PDF.

Download material

O que você achou do conteúdo?

Relatar problema