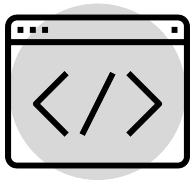
escola britânica de artes criativas & tecnologia

DE OLHO NO CÓDIGO

escola britânica de artes criativas & tecnologia

Lógica de programação com Python



- Conheça o Python
- Descubra variáveis e operadores
- Interaja com usuário
- Conheça estruturas condicionais
- Utilize listas de valores
- Aplique estruturas de repetição
- Atalhos

Conheça o Python

escola britânica de artes criativas & tecnologia

A **lógica de programação** envolve a capacidade de pensar na **sequência de ações de forma lógica e eficiente** para resolver um problema. Essa habilidade de pensar em sequência é útil não apenas na programação, mas em muitos aspectos da vida cotidiana.

Os conteúdos vistos neste módulo são conceitos básicos de lógica de programação usando **Python**. À medida que você avançar no curso, aprenderá a traduzir ideias em código de uma forma que seja fácil de entender.

Não se preocupe em dominar códigos em Python agora - na fase 4 da Jornada de Aprendizagem, você se aprofundará nesta linguagem para analisar dados.

O mais importante agora é entender **como os problemas podem ser resolvidos de forma lógica**.

Conheça o Python

Um dos conceitos da lógica de programação é a **sequência**. Ele se refere à ideia de que um programa é executado passo a passo, seguindo uma **ordem específica**, assim como você segue uma lista de tarefas quando está cozinhando ou resolvendo um problema.

Por exemplo: imagine que você deseja fazer um programa que calcule a média de duas notas. A sequência de ações poderia ser:

1. Pedir ao usuário para inserir a primeira nota;
2. Armazenar a primeira nota em uma variável;
3. Pedir ao usuário para inserir a segunda nota;
4. Armazenar a segunda nota em outra variável;
5. Calcular a média das duas notas;
6. Exibir o resultado da média na tela.

Essas etapas são executadas em ordem, uma após a outra, sem pular nenhuma. É importante seguir essa sequência para que o programa funcione corretamente.

Descubra operadores e variáveis

escola britânica de artes criativas & tecnologia

Pense em **variáveis** como "caixas" onde você pode armazenar informações. Elas são usadas para guardar dados, como números, textos ou qualquer tipo de informação que seu programa precisa manipular.

Por exemplo, você pode ter uma variável chamada "idade" para armazenar a sua idade ou uma variável chamada "nome" para armazenar o seu nome. As variáveis são essenciais para que você possa trabalhar com dados em um programa.

Veja um exemplo simples usando Python:

```
idade = 15  # Neste caso, a variável "idade" armazena o valor 15.
nome = "João"  # A variável "nome" armazena o texto "João".
```

Descubra operadores e variáveis

escola britânica de artes criativas & tecnologia

Veja porque as variáveis são importantes:

- **Armazenamento de dados**: Elas permitem que você armazene informações para serem usadas mais tarde no programa. Isso é útil quando você precisa lembrar ou manipular dados.
- **Manipulação de dados**: Você pode realizar operações matemáticas, combinar textos ou realizar outras ações em variáveis para realizar cálculos ou criar resultados.
- **Legibilidade**: Dar nomes significativos às variáveis torna seu código mais fácil de entender. Por exemplo, usar "idade" em vez de "a" é mais claro para qualquer pessoa que leia seu código.
- **Reutilização:** Você pode usar o valor armazenado em uma variável várias vezes em seu programa, evitando a necessidade de repetir a mesma informação.

Interaja com o usuário

Entrada e Saída se referem à forma como um programa interage com o mundo exterior, seja para receber informações (entrada) ou para fornecer informações (saída).

- **Entrada** (Input): Esta é a informação que um programa recebe do mundo exterior. Pode ser fornecida pelo usuário ou lida de um arquivo, sensor, rede, entre outras fontes. Em termos de programação, a entrada pode incluir números, textos, cliques do mouse, toques na tela, comandos de teclado, e assim por diante.

  Exemplo: Se você está fazendo um programa que calcula a média de notas, a entrada seria as notas inseridas pelo usuário.

- **Saída** (Output): Esta é a informação que um programa fornece ao mundo exterior, geralmente na forma de resultados visuais, sonoros ou escritos. A saída pode ser exibida na tela, salva em um arquivo, enviada a um dispositivo de impressão ou transmitida por uma rede.

  Exemplo: No programa de cálculo de média, a saída seria a média calculada exibida na tela.

Interaja com o usuário

A capacidade de lidar com entrada e saída é crucial na programação, porque os programas frequentemente precisam interagir com os usuários ou com outros sistemas. Além disso, entender como receber informações e comunicar resultados é fundamental para tornar um programa útil e compreensível.

Em Python, você pode usar funções como **input()** para obter entrada do usuário e **print()** para exibir saída na tela.

Por exemplo:

```
nome = input("Digite seu nome: ")  # Obtém o nome do usuário
print("Olá, " + nome + "!")
```

Neste exemplo, o programa recebe o nome do usuário como entrada e, em seguida, produz uma saída de saudação na tela.

Conheça estruturas condicionais

As **condicionais** são como **decisões** que você toma em sua vida diária. Por exemplo, se está chovendo, você leva um guarda-chuva.

Em Python, você pode dizer ao programa para fazer algo "se" uma condição for verdadeira.

Veja a seguir um exemplo de utilização de condicionais:

Conheça estruturas condicionais

Imagine que você está desenvolvendo um programa básico para verificar se um usuário pode acessar um sistema mediante digitação de senha. Veja como você pode resolver:

```
# Verificação de senha simples

# Definir a senha permitida
senha_permitida = "12345"

# Solicitar a senha do usuário
senha_usuario = input("Digite a senha: ")

# Verificar se a senha está correta
if senha_usuario == senha_permitida:
    print("Acesso permitido!")
else:
    print("Senha incorreta. Acesso negado.")
```

Para testar os códigos deste material no seu Colab corretamente, lembre-se de manter os espaçamentos, quando houverem. Caso contrário, o código dará erro.

Lembre: as condicionais “If” e “Else” realizam ações diferentes, dependendo se a ação é verdadeira ou falsa.

Utilize listas de valores

- **Listas** são estruturas de dados que permitem **armazenar coleções de elementos**. Esses elementos podem ser números, textos, outros tipos de dados ou até mesmo outras listas. Elas são uma maneira flexível e poderosa de organizar dados em um programa.

  Veja um exemplo simples usando Python:

  ```
  frutas = ["maçã", "banana", "laranja", "uva"]
  ```

  Neste exemplo, "frutas" é uma lista que contém quatro elementos, cada um representando um tipo de fruta.

- Você pode acessar esses elementos individualmente usando sua posição na lista. Em Python, a contagem começa em 0, então para acessar a primeira fruta (maçã), você usaria `frutas[0]`.

Utilize listas de valores

escola britânica de artes criativas & tecnologia

Veja porque as listas são importantes na lógica de programação:

- **Armazenamento de múltiplos valores**: Listas permitem que você agrupe vários valores relacionados em uma única estrutura de dados. Isso é útil quando você precisa lidar com coleções de informações, como uma lista de nomes, números, produtos, etc.
- **Acesso e manipulação de elementos**: Você pode adicionar, remover, substituir e acessar elementos individualmente em uma lista, o que é fundamental para muitas tarefas de programação.
- **Iteração**: Você pode usar loops para percorrer todos os elementos de uma lista, o que é útil quando precisa realizar a mesma operação em cada item da lista.
- **Ordenação**: Você pode classificar os elementos em uma lista, o que é útil para organizar dados em ordem crescente ou decrescente.
- **Listas complexas**: Listas podem conter outras listas, permitindo a criação de estruturas de dados complexas para lidar com dados mais complexos.

Utilize listas de valores

escola britânica de artes criativas & tecnologia

Por exemplo, imagine que você está começando a coletar dados de feedback de clientes e deseja organizar esses dados em uma lista simples e um dicionário. Veja como seu código pode ficar:

```
# Organizando dados de feedback de clientes
# Lista de feedbacks
feedbacks = ["Ótimo serviço!", "Produto excelente", "Atendimento rápido"]
# Dicionário com informações de um cliente
cliente = {
    "nome": "Maria",
    "idade": 28,
    "feedback": "Muito satisfeita"
}
# Imprimir os feedbacks da lista
print("Feedbacks dos clientes:")
for feedback in feedbacks:
    print("- " + feedback)
# Imprimir informações do cliente
print("\nInformações do cliente:")
print("Nome:", cliente["nome"])
print("Idade:", cliente["idade"])
print("Feedback:", cliente["feedback"])
```

\n é utilizado para quebrar linha.

Aplique estruturas de repetição

Os **loops** são uma parte fundamental da lógica de programação, pois permitem que você **repita uma ou mais ações várias vezes**. Eles são como instruções de repetição em um programa, permitindo que você economize tempo e esforço, evitando a repetição manual de código.

Entender como usar loops de maneira eficaz é uma habilidade importante na lógica de programação, pois permite que você resolva problemas complexos de forma mais rápida e consistente. Além disso, os loops são uma das estruturas de controle mais poderosas que um programador tem à disposição.

Existem dois tipos comuns de loops, veja a seguir:

Aplique estruturas de repetição

escola britânica de artes criativas & tecnologia

- **Loop While**: Este tipo de loop executa um conjunto de instruções repetidamente enquanto uma determinada condição for verdadeira. Assim que a condição se tornar falsa, o loop para.

```
contador = 0
while contador < 5:
  print("Isso é a iteração número", contador)
  contador += 1
```

Neste exemplo, o loop while continua a imprimir a mensagem enquanto a variável `contador` for menor que 5.

- **Loop For**: O loop for é usado para iterar sobre uma sequência, como uma lista ou um intervalo de números. Ele percorre todos os elementos da sequência.

```
frutas = ["maçã", "banana", "laranja"]
for fruta in frutas:
  print("Eu gosto de", fruta)
```

Neste exemplo, o loop for percorre a lista de frutas e imprime uma mensagem para cada uma delas.

Aplique estruturas de repetição

escola britânica de artes criativas & tecnologia

Suponha que você tenha uma lista de notas de alunos e deseje calcular a média e classificá-los com base nessa média, veja uma possibilidade:

```
# Lista de notas dos alunos
notas = [85, 92, 78, 90, 88]

# Calcula a média das notas
soma = 0
for nota in notas:
    soma += nota
media = soma / len(notas)

# Classifica os alunos com base na média
alunos_aprovados = []
alunos_reprovados = []

for nota in notas:
    if nota >= media:
        alunos_aprovados.append(nota)
    else:
        alunos_reprovados.append(nota)

# Imprime os resultados
print("Notas dos alunos:", notas)
print("Média das notas:", media)
print("Alunos aprovados:", alunos_aprovados)
print("Alunos reprovados:", alunos_reprovados
```

'append' é utilizado em listas para adicionar um elemento ao final da lista. Aqui, foi utilizado para adicionar as notas dos alunos às listas alunos_aprovados e alunos_reprovados com base na comparação com a média.

Atalhos

Reveja os atalhos mostrados nas aulas deste módulo:

- **Ctrl M Y**
  Converte em código
- **Ctrl M M**
  Converte em texto
- **Ctrl M A**
  Código Acima
- **Ctrl M B**
  Código Abaixo
- **Ctrl M I**
  Interromper execução
- **Esc**
  Tirar o foco da célula

Atalhos

Reveja os atalhos mostrados nas aulas deste módulo:

- **Ctrl M J**
  Mover célula baixo
- **Ctrl Enter**
  Executa célula
- **Ctrl Espaço**
  Mostra sugestões de funções
- **Ctrl M k**
  Mover célular acima
- **Alt Enter**
  Executa a célula e inseri uma nova
- **Shit Enter**
  Executa a célula e seleciona a próxima

escola britânica de artes criativas & tecnologia

Bons estudos!