Adaptação 1

Trecho Original 1:

Imagine que você começou a trabalhar numa empresa de desenvolvimento de software. Lá, você conseguiu um contrato para desenvolver o sistema de uma clínica veterinária e foi alocado para construir a parte do sistema que faz o cadastro do cliente e do animal dele. Então, seu chefe chegou na sua mesa e falou, “Eu preciso de um tipo de dado para armazenar informações sobre os clientes da clínica, estes dados são: o nome, a idade, o endereço e o telefone do cliente. Ah, também preciso de outro tipo em que possamos armazenar os dados dos pets, que são: o nome, o tipo (cão ou gato, por exemplo), a idade, o peso e a doença que ele tem no momento, ok? Faça isso e depois construa um programa que lê os dados de um cliente e de um pet, em sequência. Após fazer esta leitura, você deve apresentar os dados na tela, só para vermos se tudo foi armazenado corretamente. Então, posso contar com você?” A partir das orientações dadas pelo seu chefe, construa um Jupyter notebook que resolve os problemas apresentados por ele: a representação e o armazenamento dos dados do cliente, do pet, a entrada de dados a partir do teclado e a saída de dados textual em tela. Em seguida, exporte o arquivo do seu notebook e envie.

Trecho Adaptado 1:

Tarefa: Criar um Programa para Armazenar e Exibir Dados de Clientes e Animais de Estimação

Você foi designado para uma tarefa em seu novo emprego em uma empresa de desenvolvimento de software. Seu chefe precisa que você crie um programa para armazenar informações sobre clientes e seus animais de estimação.

O que Você Precisa Fazer:

Crie uma maneira de armazenar dados dos clientes, incluindo:
-Nome
-Idade
-Endereço
-Número de telefone
Crie uma maneira de armazenar dados dos animais de estimação, incluindo:
-Nome
-Tipo (por exemplo, cachorro ou gato)
-Idade
-Peso
-Doença atual

Escreva um programa que leia os dados de clientes e animais de estimação pelo teclado. Exiba os dados armazenados na tela para verificar se tudo foi armazenado corretamente.

Interface do programa

Adaptação 2

Trecho original 2:
Então, para adentrar nesses novos conceitos, imagine que você trabalha em um banco, como programador, e está desenvolvendo um programa que é responsável pelo saque de dinheiro. Certo dia, uma pessoa vai ao caixa, pede para sacar 5.000 reais, mas só tem saldo de 500 reais. Seu programa não testa se o valor pedido é maior ou igual ao saldo da conta, então permite que o cliente saque todo o valor pedido. Provavelmente, esse erro vai gerar um problema enorme, não é? Então, entenda como é possível resolver esse tipo de problema através de desvios condicionais simples e composto, a seguir. Assim, você poderá desenvolver códigos, que testam em diferentes tipos de cenários.

Para resolver o problema proposto anteriormente, você precisa construir o que chamamos de desvio condicional. Desvio condicional é um elemento em um código fonte que define a execução de um determinado bloco de comandos a partir de uma dada condição lógica. Nesse caso, o bloco de comandos é a sequência de passos necessária para que haja o saque de dinheiro, e a condição lógica, para que este bloco ocorra, é o teste que diz se o saldo é maior ou igual ao valor requisitado para saque. Para que o bloco seja executado e haja o saque do valor, o teste tem que resultar em verdadeiro (True). Em Python, a sintaxe, ou seja, a ordem e disposição das palavras reservadas e expressões ou comandos para esse desvio condicional simples está estruturada no código a seguir:

```
if condição:

    comando 01
```

comando 02

Trecho adaptado 2:

Compreendendo Declarações Condicionais em Python

Cenário: Você é programador em um banco e está desenvolvendo um programa que gerencia saques em dinheiro. Um cliente deseja sacar 5.000 reais, mas possui um saldo de apenas 500 reais.

Problema: Se o programa não verificar se o valor solicitado é maior ou igual ao saldo da conta, ele permitirá que o cliente saque todo o valor. Isso seria um grande problema!

Solução: Podemos resolver este problema usando declarações condicionais. Uma declaração condicional é parte do código que decide executar ou não um bloco de comandos com base em uma condição lógica.

Vamos Simplificar:
Condição Lógica: No nosso cenário, a condição é "O saldo é maior ou igual ao valor solicitado?

Bloco de Comandos: Se a resposta for True, execute o bloco de comandos para permitir o saque.

Declaração Condicional: Usamos uma instrução if para testar a condição e executar o bloco de comandos se for True.

Código Exemplo:

```
saldo = 500
valor solicitado = 5000
 if saldo >= valor_solicitado:
    print("Saque permitido”)
else:
    print("Fundos insuficientes")
```

O que Acontece: Se a condição for True, o programa imprime "Saque permitido". Se for False, imprime "Fundos insuficientes".

Ao usar declarações condicionais, podemos escrever códigos que testam diferentes cenários e tomam decisões com base em condições. Isso nos ajuda a evitar problemas como o do nosso cenário.

Ilustração do saque negado por falta de saldo.

Adaptação 3

Trecho original 3:
Já list são elementos um pouco mais complexos, mas podemos entender este tipo através de um exemplo: digamos que existem 3 baldes alinhados, um depois do outro, os quais podemos colocar qualquer coisa dentro deles. Quando eu quiser colocar ou retirar algum elemento, eu só tenho que me referir à posição do balde na sequência (primeiro, segundo ou terceiro). Portanto, listas sâo isso, elementos enumerados que você pode colocar qualquer coisa dentro, e que você acessa através da posição em uma sequência.

Trecho adaptado 3:

Compreendendo Listas em Python

O que é uma Lista?
Uma lista é uma coleção de itens que podem ser armazenados e acessados de acordo com sua posição em uma sequência.

Exemplo:
Imagine que você tem 3 caixas alinhadas, uma após a outra. Você pode colocar qualquer coisa dentro dessas caixas. Quando quiser adicionar ou remover um item, basta se referir à posição da caixa na sequência (primeira, segunda ou terceira).

Como as Listas Funcionam:

Você pode armazenar itens: Coloque qualquer tipo de dado (números, palavras, etc.)

em uma lista.

Os itens têm posições: Cada item tem uma posição específica na lista (1ª, 2ª, 3ª, etc.).

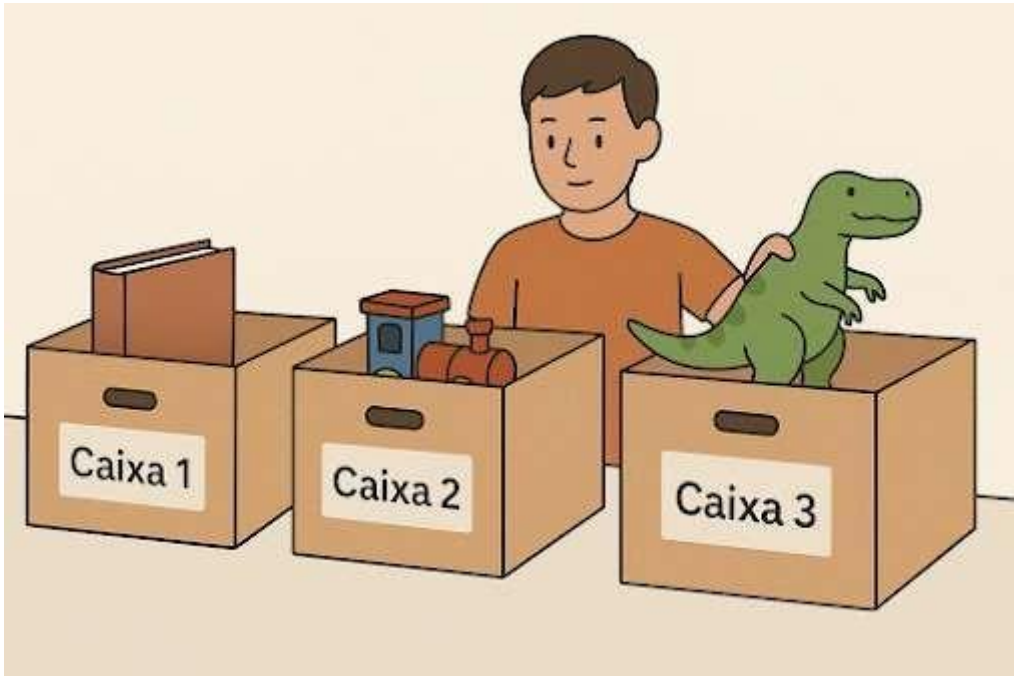
Acesse os itens pela posição: Você pode acessar um item com base em sua posição na lista.

Código de Exemplo:

```
minha lista = [1, 2, 3] # cria uma lista com 3 itens
print(minha lista[0]) # imprime o primeiro item (saída: 1)
```

Usando listas, você pode armazenar e gerenciar coleções de dados em Python.

Ilustração do conceito de listas usando caixas nomeadas guardando objetos.



Adaptação 4

Trecho original 4:

Computadores servem para automatizar tarefas, principalmente, as repetitivas. Por exemplo, a placa de vídeo do seu computador fará muitas tarefas repetidas, se você decidir jogar um jogo 3D. Estas placas são equipamentos específicos para fazer tarefas repetitivas com elementos visuais. Já uma tabela de suíte de escritório, automatiza o processo repetitivo de anotar diversos dados. Repetição é uma palavra chave aqui, então compreenda como isso pode se aplicar a programas que você poderá desenvolver em Python.

Para isso, imagine que você está desenvolvendo o sistema do caixa de um supermercado, codificando a parte que faz leitura do código de barra dos produtos comprados pelo cliente. Essa tarefa é simples: o programa lê o código do primeiro produto e registra a venda, lê o código do segundo e faz a anotação novamente, e assim por diante. Já deu para perceber que além de simples, essa tarefa é bastante repetitiva, não é? Então, faz muito sentido que você crie um código que se repita e faça este trabalho enquanto houver novos produtos para serem escaneados.

Trecho adaptado 4:

O que é Repetição?

Repetição significa realizar uma tarefa várias vezes. Os computadores são excelentes em automatizar tarefas repetitivas.

Exemplos:

- A placa gráfica de um computador executa muitas tarefas repetitivas ao rodar um jogo em 3D.
- Uma planilha automatiza o processo repetitivo de entrada de diferentes dados.

Por que a Repetição é Importante na Programação?

Na programação, a repetição nos ajuda a escrever códigos mais eficientes. Imagine que estamos desenvolvendo um sistema de checkout de supermercado que lê códigos de barras de produtos. A tarefa é simples: ler o código do primeiro produto e registrar a venda, ler o código do segundo produto e repetir o processo, e assim por diante.

Repetição no Codigo:

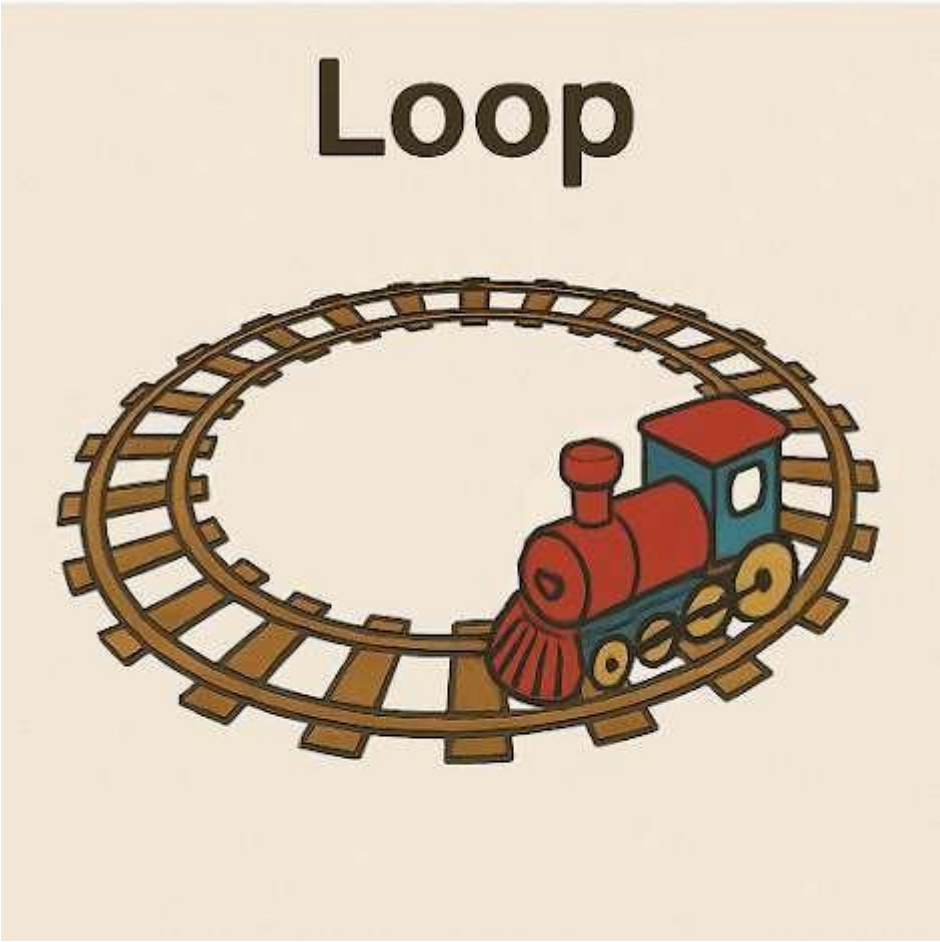
Podemos escrever um código que repita esse processo para cada novo produto escaneado. Isso faz sentido porque a tarefa é repetitiva.

Como Alcançar a Repetição em Python?

Podemos usar loops (como for ou while) para repetir um bloco de código várias vezes. Assim, conseguimos automatizar tarefas repetitivas e tornar nossos programas mais eficientes.

Vamos focar em entender como escrever códigos que repetem tarefas de forma eficiente usando Python!

Ilustração de Loops representado por um trem que anda em um trilho circular.



Adaptação 5

Trecho original 5:

Imagine que você tem um aplicativo que te ajuda a criar e manter listas de compras. Internamente, ele pode guardar os seus itens em uma lista de strings, em uma variável chamada compras, sem nenhuma ordenação especial. Se você pedir ao aplicativo que adicione 'farinha' à lista, por causa de uma receita que você irá preparar, o mínimo que o aplicativo deveria fazer é verificar se o item 'farinha' já não faz parte desta lista, certo? Em Python, isto poderia ser feito com a expressão 'farinha' in compras, que resulta em True ou False.

Trecho adaptado 5:

Usando Listas para Armazenar Dados
Imagine que você tem um aplicativo que ajuda a criar e gerenciar listas de compras. O aplicativo armazena seus itens em uma lista de strings chamada compras, sem nenhuma ordem específica.

Vamos supor que você queira adicionar 'farinha' à lista por causa de uma receita que vai preparar. O aplicativo deveria, pelo menos, verificar se 'farinha' já está na lista, certo?

Em Python, isso pode ser feito usando a expressão 'farinha' in compras, que retorna True ou False.

Aqui está um exemplo:

```
compras = ['leite','ovos','pão']
print(’farinha’ in compras) # Saída: False
compras.append('farinha')
print('farinha' in compras) # Saída: True
```

Esse código mostra como verificar se um item já está na lista e, em seguida, adicioná-lo caso ainda não esteja

Ilustração do conceito de listas usando aplicativo.

Adaptação 6

Trecho original 6:

Funções em programação se assemelham bastante à ideia matemática de entidades que transformam valores ou a linhas de produção que combinam diferentes materiais para fabricar um produto. Essa descrição talvez parece familiar, porque programas fazem justamente isso. Com funções em Python, é possível dividir programas em partes menores que podem ser recombinadas para criar outros programas ou, simplesmente, para reduzir a complexidade do programa original.

Trecho adaptado 6:

O que são Funções na Programação?

Funções na programação são como máquinas que recebem valores e produzem novos valores. Elas podem ser comparadas a linhas de montagem que combinam diferentes materiais para criar um produto.

Como as Funções Funcionam?

Em Python, funções são blocos de código que realizam uma tarefa específica. Elas recebem entradas (chamadas de argumentos) e retornam saídas.

Exemplo:

```
def cumprimentar(nome):
    print("Olá, " + nome + "!")
cumprimentar("João") # Saída: Olá, João!
cumprimentar("Maria") # Saída: Olá, Maria!
```

Neste exemplo, a função cumprimentar recebe um nome como entrada e imprime uma mensagem de saudação.

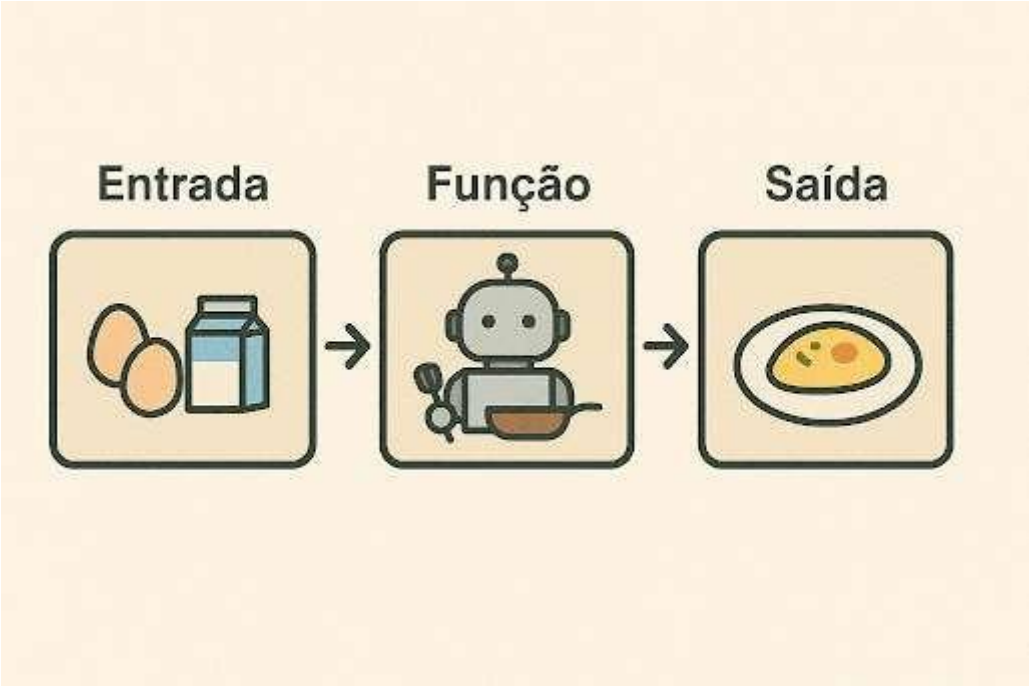
Por que as Funções são úteis?

As funções ajudam a tornar os programas mais organizados e reutilizáveis. Elas permitem que você:

- Divida programas complexos em partes menores e mais gerenciáveis
- Reutilize código em diferentes partes do seu programa
- Crie novos programas combinando funções existentes

Ao usar funções, é possível escrever códigos mais eficientes e eficazes.

Ilustração de como funciona uma função.



Adaptação 7

Trecho original 7:

Jogos de simulação são muito famosos e divertidos. Em SimCity, por exemplo, o jogador brinca de gerenciar uma cidade; em Tamagoshi, o jogador tem um pequeno dispositivo que simula toda a vida de um animal de estimação. Se baseando nesses passatempos, que tal fazer uma pequena simulação, como se estivesse construindo uma pequena cópia do Tamagoshi? Para esta oficina, a partir do que aprendeu até aqui, você deverá construir uma classe para abstrair um animal qualquer. Nessa abstração, inclua os atributos nome, espécie e fome. Por exemplo, nome e espécie podem ser passados como parâmetro no construtor dessa classe; já a fome deve ser inicializada com valor 0, pois ele é o atributo indicador numérico que armazenará o quanto de fome o animal tem em um dado momento. Assim, 0 (zero) fome significa que ele está satisfeito, ou seja, não tem fome nenhuma. Agora, se tiver 1, significa que ele tem um pouquinho de fome; 2, um pouco mais; e assim por diante.

Trecho adaptado 7:

Jogos de Simulação

Os jogos de simulação são populares e divertidos. No SimCity, você gerencia uma cidade, enquanto no Tamagotchi, você cuida de um animal de estimação virtual.

Vamos Criar uma Simulação Simples

Vamos criar uma simulação simples utilizando o que aprendemos até agora. Construiremos uma classe para representar um animal.

A Classe Animal

Nossa classe Animal terá três atributos:

- **name**: o nome do animal
- **species**: a espécie do animal
- **hunger**: um valor numérico que indica o quão faminto o animal está

Inicializando os Atributos

Quando criarmos um objeto Animal, passaremos o nome e a espécie como parâmetros. 0 atributo **hunger** começará em 0, indicando que o animal não está com fome.

O que Significa Hunger?

- **0 hunger** significa que o animal está cheio e não está com fome.
- **1 hunger** significa que o animal está um pouco faminto.
- **2 hunger** significa que o animal está mais faminto.

E assim por diante.

Ilustração da ideia do jogo para mostrar como funcionam as classes.