EAD

EDUCAÇÃO A DISTÂNCIA

UNIDADE III

Pensamento Lógico Computacional com Python

Prof. MSc. Tarcísio Peres

Conteúdo da unidade III

- Listas: operações e métodos.
- Dicionários: criação e uso.
- Leitura de dados do teclado.
- Escrita de dados em arquivos.

Introdução às listas

- Listas são estruturas de dados fundamentais em Python, permitindo armazenar múltiplos valores em uma única variável.
- Diferentemente de variáveis que armazenam um único valor, as listas podem conter diversos elementos organizados em uma sequência ordenada.
- Os elementos dentro de uma lista são acessados por índices numéricos, começando do zero, o que possibilita manipulações eficientes.
- A principal característica das listas é sua mutabilidade, permitindo adicionar, remover e modificar elementos sem necessidade de recriação.
    - Sua flexibilidade as torna úteis para armazenar e processar conjuntos de dados variáveis, comuns em aplicações de inteligência artificial e ciência de dados.
    - O domínio das operações com listas é essencial para programadores, pois elas são amplamente utilizadas na construção de algoritmos e manipulação de dados.

Manipulação de listas – Adição e remoção de elementos

- Python oferece métodos eficientes para adicionar elementos às listas, como append() para inserir no final e insert() para adicionar em posições específicas.
- Elementos podem ser removidos pelo valor com remove() ou pela posição com pop(), possibilitando maior controle sobre a estrutura.
- O método del permite a remoção direta de elementos em uma posição específica, sem necessidade de armazenamento temporário.
- Para limpar completamente uma lista, pode-se utilizar clear(), tornando-a vazia sem destruir a variável.

  - Essas operações tornam listas ideais para armazenar conjuntos de dados dinâmicos, que precisam ser atualizados constantemente.
  - Compreender essas técnicas de manipulação é essencial para otimizar a eficiência e a organização de algoritmos baseados em listas.

Ordenação e reorganização de listas

- O método sort() organiza os elementos de uma lista em ordem crescente por padrão, mas pode ser ajustado para ordem decrescente.
- Para preservar a lista original e obter uma versão ordenada temporária, utiliza-se a função sorted().
- O método reverse() inverte a ordem dos elementos na lista, sendo útil para apresentações específicas de dados.
- Essas operações são importantes para estruturar informações de maneira lógica, facilitando buscas e análises.
    - A ordenação é frequentemente utilizada para organizar dados temporais, como eventos históricos ou sequências cronológicas.
    - O domínio dessas técnicas permite criar algoritmos eficientes que manipulam e apresentam informações de forma otimizada.

Acessando e modificando elementos em listas

- Elementos individuais podem ser acessados pelo índice, como lista[0], para obter o primeiro item.
- Índices negativos permitem acessar elementos a partir do final da lista, como lista[-1] para o último item.
- A modificação direta é possível, alterando valores em posições específicas, como lista[2] = novo valor.
- O fatiamento (slicing) permite extrair subconjuntos da lista, como lista[1:4] para obter do segundo ao quarto elemento. Ele utiliza a sintaxe lista[início:fim:passo], permitindo selecionar intervalos e definir a frequência dos elementos extraídos.
    - Compreender essas técnicas é essencial para manipular dados de maneira eficiente e evitar erros de indexação.
    - Operações de acesso e modificação são comuns na análise e no processamento de dados estruturados.

Exemplo: Listas e planejamento de viagens no tempo

- No contexto de viagens no tempo, listas podem armazenar diferentes períodos históricos a serem visitados.
- Permitem que o viajante organize sua jornada em uma sequência lógica, facilitando a navegação cronológica.
- Métodos de ordenação e inserção são úteis para reorganizar o itinerário e adicionar novos destinos conforme necessário.
- O uso de listas possibilita avaliar a distância temporal entre eventos e planejar a melhor rota de deslocamento.
    - A remoção de destinos pode ser feita com remove() para ajustes no plano de viagem sem comprometer a estrutura da lista.
    - Essa aplicação prática demonstra como listas são ferramentas versáteis para organizar informações temporais e sequenciais.

Exemplo: Listas e planejamento de viagens no tempo

```
quantidade = int(input("Quantos anos você deseja inserir na lista de destinos temporais? "))
destinos = []
for i in range(quantidade):
    ano = int(input("Insira um ano: "))
    destinos.append(ano)
print("Lista completa de destinos:", destinos)
# Abaixo demonstramos o fatiamento de listas.
# Suponhamos que o viajante deseje olhar apenas os primeiros três destinos inseridos.
# Usamos destinos[0:3] para obter um sublista começando no índice 0 até, mas não incluindo, o
índice 3.
primeiros_tres = destinos[0:3]
print("Os primeiros três destinos cadastrados:", primeiros_tres)
```

```
# Caso o viajante queira ver apenas os últimos dois destinos,
# pode-se usar índices negativos. destinos[-2:] retorna a
sublista a partir do penúltimo elemento até o final.
ultimos_dois = destinos[-2:]
print("Os dois últimos destinos inseridos:", ultimos_dois)
```

Exemplo: Listas e planejamento de viagens no tempo

```
# Também é possível pular elementos usando um passo (step).
# Suponha que o viajante queira analisar apenas destinos alternados, saltando um elemento a cada passo.
# Usando destinos[::2], obtemos todos os elementos da lista, do início ao fim, mas apenas nos índices pares.
alternados = destinos[::2]
print("Destinos alternados (pulando um a cada passo):", alternados)

# Além disso, o viajante pode querer um trecho intermediário da lista.
# Por exemplo, se houver uma longa lista de anos, podemos extrair um pedaço do meio.
# Suponhamos que haja 5 ou mais elementos e queremos pegar do segundo até o quarto destino.
# Esse fatiamento é feito com destinos[1:4], retornando elementos nos índices 1, 2 e 3.

if len(destinos) >= 5:
    trecho_intermediario = destinos[1:4]
    print("Um trecho intermediário de destinos (índices 1 a 3):", trecho_intermediario)
else:
    print("A lista não tem destinos suficientes para mostrar um trecho intermediário.")
```

Exemplo: Listas e planejamento de viagens no tempo

```
# Por fim, podemos inverter a ordem dos destinos usando o fatiamento com passo negativo.
# destinos[::-1] percorre a lista de trás para frente.
invertidos = destinos[::-1]
print("Destinos em ordem invertida:", invertidos)
```

Análise de dados em listas com funções nativas

- O método len() retorna o número total de elementos armazenados na lista, útil para contabilizar registros.
- Funções como min() e max() identificam o menor e o maior valor em uma lista numérica, fornecendo insights sobre os extremos da sequência.
- A função sum() calcula a soma dos elementos da lista, sendo útil para análises estatísticas e cálculos agregados.
- A combinação dessas funções permite extrair informações relevantes rapidamente, sem necessidade de loops manuais.

Comparação entre listas e tuplas

- Listas são mutáveis, permitindo adição, remoção e alteração de elementos, enquanto tuplas são imutáveis, garantindo integridade dos dados.
- Tuplas consomem menos memória e são mais rápidas para determinadas operações, devido à sua natureza fixa.
- O uso de listas é preferível para coleções dinâmicas, enquanto tuplas são ideais para dados que não devem ser alterados.
- A conversão entre listas e tuplas é possível com tuple(lista) e list(tupla), adaptando a estrutura conforme necessário.
- Aplicações como armazenamento de coordenadas ou registros fixos geralmente utilizam tuplas para evitar modificações acidentais.
    - A escolha entre listas e tuplas depende das necessidades específicas do programa e da manipulação esperada dos dados.

Interatividade

Qual das alternativas abaixo é correta?

a) O método append() permite inserir um elemento em qualquer posição de uma lista, deslocando os elementos subsequentes automaticamente.

b) O método sort() em Python ordena uma lista de maneira permanente, enquanto sorted() retorna uma nova lista ordenada sem modificar a original.

c) O operador in é utilizado apenas para verificar se um valor está presente em listas ordenadas, não funcionando em listas desordenadas.

d) O fatiamento de listas em Python só pode ser feito utilizando índices positivos, pois índices negativos não são suportados.

e) Tuplas e listas são idênticas em Python, pois ambas permitem adição e remoção de elementos de forma dinâmica.

Resposta

Qual das alternativas abaixo é correta?

a) O método append() permite inserir um elemento em qualquer posição de uma lista, deslocando os elementos subsequentes automaticamente.

b) O método sort() em Python ordena uma lista de maneira permanente, enquanto sorted() retorna uma nova lista ordenada sem modificar a original.

c) O operador in é utilizado apenas para verificar se um valor está presente em listas ordenadas, não funcionando em listas desordenadas.

d) O fatiamento de listas em Python só pode ser feito utilizando índices positivos, pois índices negativos não são suportados.

e) Tuplas e listas são idênticas em Python, pois ambas permitem adição e remoção de elementos de forma dinâmica.

Dicionários em Python – Estrutura e funcionalidade

- Dicionários são estruturas de dados que armazenam informações em pares de chave-valor, permitindo acesso eficiente aos dados.
- Diferentemente das listas, os dicionários não dependem de índices numéricos; em vez disso, as chaves identificam cada valor de forma única.
- Essa abordagem facilita a busca de informações, eliminando a necessidade de percorrer sequencialmente os elementos como em listas.
- As chaves em um dicionário podem ser de qualquer tipo imutável, como strings, números ou tuplas, garantindo flexibilidade.
    - A eficiência dos dicionários os torna ideais para armazenar grandes volumes de dados e acessá-los rapidamente.
    - Essa estrutura é amplamente utilizada em aplicações que exigem organização clara e recuperação ágil de informações.

Comparação entre listas e dicionários

- Listas organizam elementos em sequência, acessados por índices numéricos, enquanto dicionários permitem acesso direto via chaves.
- A busca em listas pode ser lenta, pois exige a varredura sequencial dos elementos até encontrar o desejado.
- Em dicionários, a busca é imediata, pois as chaves funcionam como identificadores diretos para os valores armazenados.
- Listas são úteis para armazenar conjuntos ordenados de dados, enquanto dicionários são ideais para relacionar informações complexas.
    - O uso adequado de cada estrutura depende do contexto e do tipo de manipulação que será realizada no programa.
    - Dicionários são frequentemente usados para representar entidades do mundo real, como registros de clientes, produtos e eventos históricos.

Chaves e valores – Organização dos dados

- A chave em um dicionário deve ser única e imutável, garantindo a consistência e a integridade dos dados armazenados.
- Os valores associados às chaves podem ser de qualquer tipo de dado, incluindo números, strings, listas e até mesmo outros dicionários.
- Essa flexibilidade permite criar estruturas de dados altamente organizadas, facilitando o armazenamento e a recuperação de informações.
- A modificação de um valor dentro de um dicionário é direta, pois basta referenciar a chave correspondente e atribuir um novo valor.
    - Essa característica torna os dicionários uma excelente escolha para armazenar configurações de sistemas e representar entidades complexas.

Eficiência na busca e acesso aos dados

- Dicionários utilizam tabelas hash para mapear chaves a valores, permitindo acessos rápidos e eficientes.
- O tempo de busca em um dicionário é constante na maioria dos casos, independentemente do tamanho da estrutura.
- Esse desempenho superior faz dos dicionários a estrutura ideal para operações que envolvem consultas frequentes a grandes conjuntos de dados.
- Diferentemente das listas, que exigem buscas sequenciais, os dicionários permitem acessar qualquer valor diretamente pela chave correspondente.
    - Essa característica os torna fundamentais em aplicações como bancos de dados em memória e indexação de informações.
    - A eficiência no acesso aos dados é um dos principais motivos pelos quais dicionários são amplamente utilizados na programação moderna.

Dicionários e o formato JSON – "Djei-sun" (/ˈdʒeɪ.sən/)

- O JSON (JavaScript Object Notation) é um dos formatos mais utilizados para a troca de dados entre sistemas remotos.
- Sua estrutura é baseada no conceito de chave-valor, o que o torna altamente compatível com os dicionários em Python.
- Em JSON, as chaves são sempre strings, enquanto os valores podem ser números, strings, listas ou outros objetos aninhados.
- O módulo json do Python permite converter facilmente dicionários para JSON e vice-versa, facilitando a integração com APIs e bancos de dados.
    - A popularidade do JSON se deve à sua simplicidade, eficiência e legibilidade, tornando-o ideal para comunicação entre aplicações.
    - O uso de dicionários para manipular JSON agiliza o desenvolvimento de sistemas distribuídos e o armazenamento de configurações.

JSON – Exemplo

- Receita de bolinho de chuva

Fonte: autoria própria.


```
{
  "receita": {
    "nome": "Bolinho de Chuva",
    "porcoes": 20,
    "tempo_preparo_minutos": 30,
    "ingredientes": [
      {
        "nome": "Farinha de trigo",
        "quantidade": 2,
        "unidade": "xícaras"
      },
      {
        "nome": "Açúcar",
        "quantidade": 3,
        "unidade": "colheres de sopa"
      },
      {
        "nome": "Ovo",
        "quantidade": 2,
        "unidade": "unidades"
      },
      {
        "nome": "Leite",
        "quantidade": 1,
        "unidade": "xícara"
      },
      {
        "nome": "Fermento em pó",
        "quantidade": 1,
        "unidade": "colher de chá"
      },
      {
        "nome": "Óleo para fritar",
        "quantidade": 500,
        "unidade": "ml"
      },
      {
        "nome": "Açúcar",
        "quantidade": 0.5,
        "unidade": "xícara"
      },
      {
        "nome": "Canela em pó",
        "quantidade": 1,
        "unidade": "colher de sopa"
      }
    ],
```

JSON – Exemplo

- Receita de bolo

```
      "modo_preparo": [
        "Em uma tigela, misture os ovos, o açúcar e o leite até obter um creme homogêneo.",
        "Adicione a farinha de trigo aos poucos, mexendo bem para não empelotar.",
        "Acrescente o fermento e misture delicadamente.",
        "Aqueça o óleo em uma panela funda.",
        "Com a ajuda de uma colher, despeje pequenas porções da massa no óleo quente.",
        "Frite até que os bolinhos fiquem dourados por igual.",
        "Retire e escorra em papel toalha.",
        "Misture o açúcar e a canela em um prato e passe os bolinhos ainda quentes nessa mistura."
        "Sirva quente."
      ],
      "observacoes": "Se desejar, pode adicionar raspas de limão ou laranja para um sabor extra."
    }
  }
```

Fonte: autoria própria.

Dicionários e o formato JSON


```
{ "receita":
        { "nome": "Bolinho de Chuva", "porcoes": 20, "tempo_preparo_minutos": 30, "ingredientes":
                [
                        { "nome": "Farinha de trigo", "quantidade": 2, "unidade": "xícaras" }, {
"nome": "Açúcar", "quantidade": 3, "unidade": "colheres de sopa" }, { "nome": "Ovo", "quantidade":
2, "unidade": "unidades" }, { "nome": "Leite", "quantidade": 1, "unidade": "xícara" }, { "nome":
"Fermento em pó", "quantidade": 1, "unidade": "colher de chá" }, { "nome": "Óleo para fritar",
"quantidade": 500, "unidade": "ml" }, { "nome": "Açúcar", "quantidade": 0.5, "unidade": "xícara" },
{ "nome": "Canela em pó", "quantidade": 1, "unidade": "colher de sopa" }
                ], "modo_preparo":     [ "Em uma tigela, misture os ovos, o açúcar e o leite até
obter um creme homogêneo.", "Adicione a farinha de trigo aos poucos, mexendo bem para não
empelotar.", "Acrescente o fermento e misture delicadamente.", "Aqueça o óleo em uma panela
funda.", "Com a ajuda de uma colher, despeje pequenas porções da massa no óleo quente.",
                                        "Frite até que os bolinhos fiquem dourados por igual.",
                                        "Retire e escorra em papel toalha.", "Misture o açúcar e a
                                        canela em um prato e passe os bolinhos ainda quentes nessa
                                        mistura.", "Sirva quente."
                                        ],
                                        "observacoes": "Se desejar, pode adicionar raspas de limão
                                        ou laranja para um sabor extra." }
                                }
```

Armazenando perfis de usuários com dicionários

- Dicionários são frequentemente usados para armazenar informações sobre usuários, como nome, idade e localização.
- Cada chave no dicionário representa um atributo específico do usuário, permitindo fácil recuperação dos dados.
- A modificação de informações é simples, bastando alterar o valor correspondente à chave desejada.
- Essa estrutura também permite a expansão dos dados armazenados, adicionando novos atributos conforme necessário.
    - Perfis de usuários podem ser enriquecidos com listas e dicionários aninhados, permitindo representações mais detalhadas.
    - Essa abordagem é utilizada em sistemas de autenticação, redes sociais e gerenciamento de cadastros em aplicações.

Dicionários – Exemplo prático

```
# Inicialmente, cria-se um dicionário vazio para armazenar as informações do viajante.
perfil_viajante = {}
# O nome será uma string, representando a identidade do viajante.
nome = input("Insira o seu nome: ")
# Agora, pede-se a idade, que é convertida para inteiro.
idade = int(input("Insira a sua idade: "))
# Por fim, solicita-se o ano de origem do viajante.
# Esse ano será igualmente transformado em inteiro, facilitando cálculos futuros, caso seja
necessário.
ano_origem = int(input("Insira o ano de origem: "))
                                    # Com as informações coletadas, insere-se cada uma no dicionário
                                    'perfil_viajante' através de pares chave-valor.
                                    # Aqui, 'nome', 'idade' e 'ano_origem' são as chaves, enquanto as
                                    variáveis que contêm os dados do usuário são os valores.
                                    perfil_viajante["nome"] = nome
                                    perfil_viajante["idade"] = idade
                                    perfil_viajante["ano_origem"] = ano_origem

                                    print("Perfil do viajante:", perfil_viajante)
                                    # Por exemplo, para imprimir apenas o nome do viajante, pode-se
                                    fazer:
                                    print("Nome do viajante:", perfil_viajante["nome"])
```

Interatividade

Qual das alternativas abaixo é correta?

a) Assim como no formato JSON, em Python, os dicionários exigem que todas as chaves sejam do tipo string, pois não permitem outros tipos de dados como chaves.

b) O método .split() é utilizado para dividir um dicionário em várias partes, separando seus pares chave-valor em listas distintas.

c) O JSON é um formato de dados estruturado baseado na organização de pares chave-valor, similar aos dicionários em Python, permitindo fácil conversão entre os dois.

d) Um dicionário em Python mantém seus elementos sempre ordenados pela ordem de inserção, independentemente da versão da linguagem utilizada.

e) A estrutura de um dicionário não permite aninhamento de outros dicionários como valores, pois apenas tipos primitivos podem ser armazenados.

Resposta

Qual das alternativas abaixo é correta?

a) Assim como no formato JSON, em Python, os dicionários exigem que todas as chaves sejam do tipo string, pois não permitem outros tipos de dados como chaves.

b) O método .split() é utilizado para dividir um dicionário em várias partes, separando seus pares chave-valor em listas distintas.

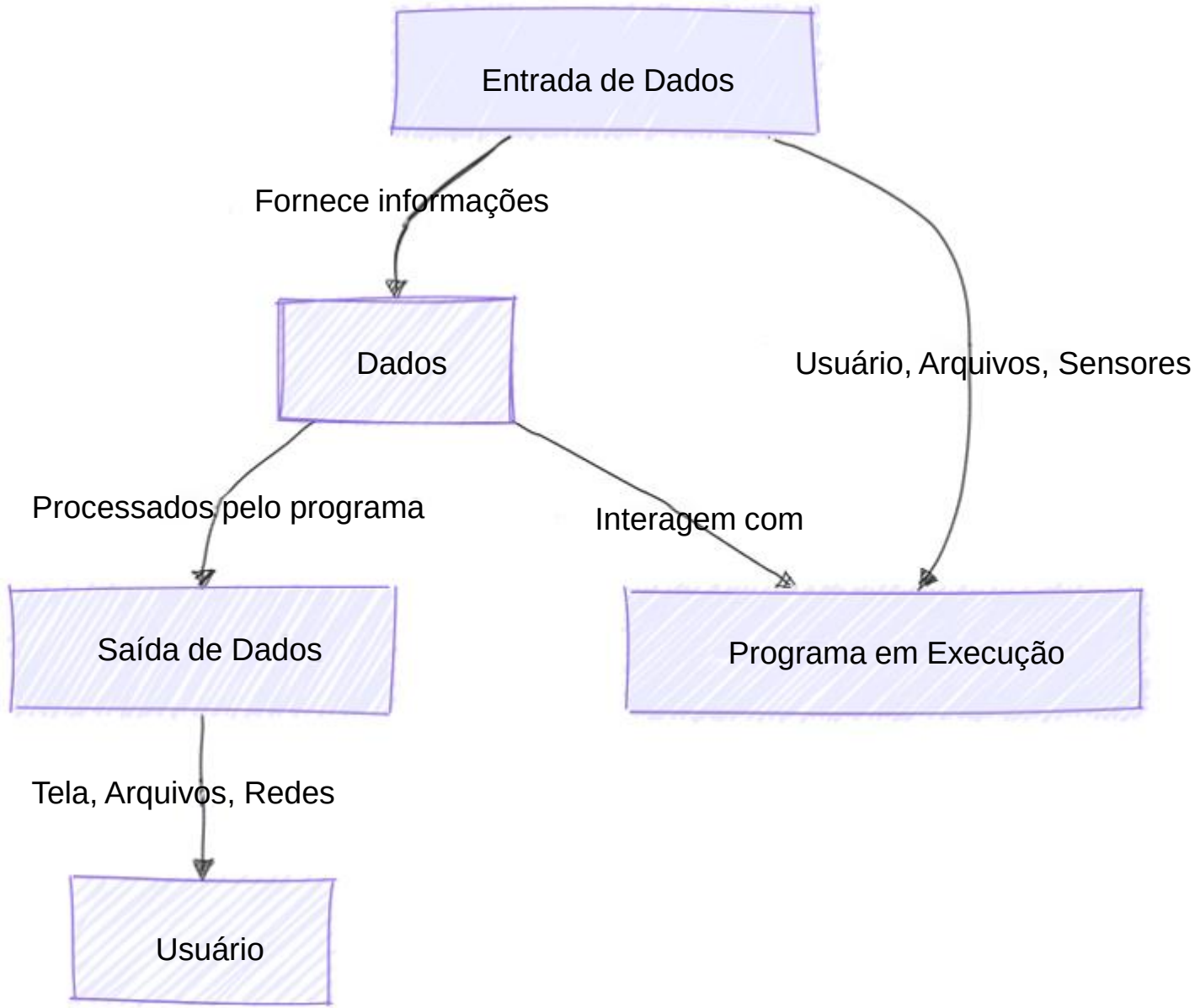
c) O JSON é um formato de dados estruturado baseado na organização de pares chave-valor, similar aos dicionários em Python, permitindo fácil conversão entre os dois.

d) Um dicionário em Python mantém seus elementos sempre ordenados pela ordem de inserção, independentemente da versão da linguagem utilizada.

e) A estrutura de um dicionário não permite aninhamento de outros dicionários como valores, pois apenas tipos primitivos podem ser armazenados.

Input e Output (I/O)

- Entrada e Saída (E/S)



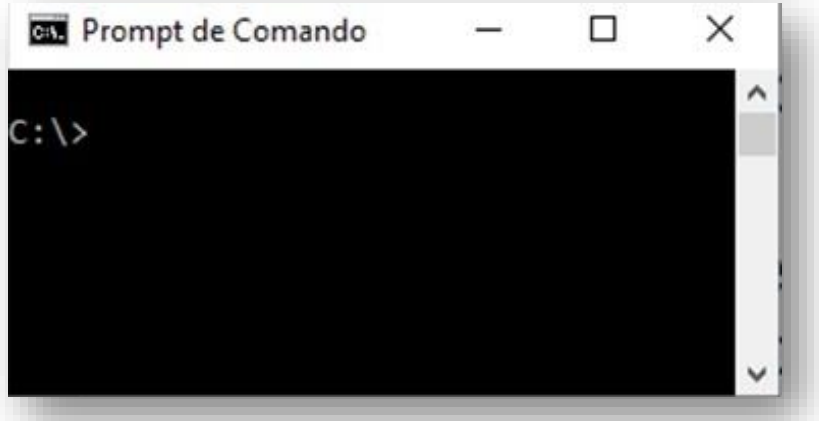
Fonte: autoria própria.

A função input() e a interação com o usuário

- A função input() permite capturar dados fornecidos pelo usuário durante a execução do programa, tornando-o interativo.
- O argumento da função input() pode ser uma mensagem exibida no console para orientar a entrada do usuário.
- Qualquer dado inserido via input() é armazenado como string, independentemente de ser um número ou caractere especial.
- Conversões são frequentemente necessárias para transformar a entrada do usuário em tipos adequados para cálculos e lógica do programa.
    - A entrada de dados em tempo de execução possibilita personalizar a experiência do usuário e adaptar os resultados conforme suas escolhas.
    - input() é amplamente utilizada em sistemas interativos, formulários, chatbots e aplicações que exigem resposta do usuário.

O console e o prompt de entrada

- O console é uma interface textual que permite interação direta entre o usuário e o sistema ou programa.
- Ele recebe comandos, exibe saídas e fornece um ambiente para execução de scripts sem interfaces gráficas.
- O termo "prompt" refere-se à indicação visual exibida no console, informando que o sistema está pronto para receber comandos.
- Diferentes sistemas operacionais possuem variações de prompt, como $ em Unix/Linux e > no Windows.
    - O console é amplamente utilizado em desenvolvimento de software, administração de sistemas e execução de scripts automatizados.



Fonte: autoria própria.

Conversão de tipos em input()

- Como input() retorna sempre uma string, conversões são necessárias quando se deseja processar números e outros tipos de dados.
- A função int() converte uma string numérica para um número inteiro, essencial para cálculos e comparações lógicas.
- A função float() transforma uma string em um número decimal, útil em contextos científicos e financeiros.
- bool() interpreta valores lógicos, convertendo strings como "True" e "False" em valores booleanos.
    - O tratamento de erros ao converter entradas é fundamental para evitar falhas no programa ao receber valores inválidos.
    - Programas interativos robustos utilizam verificações e conversões para garantir entradas coerentes e processáveis.

Exemplo: Simulando viagens no tempo com input()

- O uso de input() permite criar uma experiência interativa ao coletar o ano de destino para uma viagem no tempo.
- A diferença entre o ano atual e o ano informado determina se a viagem será ao passado, ao futuro ou se permanecerá no presente.
- Perguntas adicionais, como o objetivo da viagem e a duração da estadia, enriquecem a narrativa e tornam a experiência mais imersiva.
- A exibição de um resumo das respostas antes da execução final permite ao usuário confirmar os dados inseridos.
    - A utilização de if-elif-else possibilita estruturar a lógica e fornecer respostas dinâmicas com base na entrada do usuário.
    - Essas interações podem ser expandidas para criar jogos, simulações históricas ou assistentes inteligentes.

Simulando viagens no tempo com input()

```
from datetime import datetime
ano_atual = datetime.now().year
# Solicita o ano de destino. O usuário insere um ano, e o programa converte para inteiro.
# Caso o usuário insira um valor menor que o ano atual, significa uma viagem ao passado;
# se for maior, ao futuro; se igual, permanece no presente.
ano_destino = int(input("Insira o ano de destino da sua viagem no tempo: "))
# Solicita o objetivo da viagem. Aqui, não há uma estrutura rígida, o usuário pode digitar qualquer texto explicando suas intenções.
objetivo = input("Qual é o objetivo principal desta viagem? (ex: testemunhar um evento, encontrar alguém, estudar a cultura) ")
                                    # Pergunta se o usuário deseja retornar imediatamente após atingir o ano de destino, ou se vai permanecer um tempo lá.
                                    # Aqui, uma resposta simples como 's' ou 'n' pode determinar o comportamento.
                                    permanecer = input("Você deseja permanecer um tempo no destino antes de retornar ao presente? (s/n) ")
```

Simulando viagens no tempo com input()

```
# Exibe um resumo das informações coletadas. Isso permite ao usuário verificar se os dados inseridos fazem sentido.
print("\n--- Configuração da Viagem Temporal ---")
print(f"Ano atual: {ano_atual}")
print(f"Ano de destino: {ano_destino}")
print(f"Objetivo da viagem: {objetivo}")
# Para informar a direção temporal da viagem, verifica-se a relação entre ano_destino e ano_atual.
if ano_destino > ano_atual:
    print("Você está planejando uma viagem ao futuro.")
elif ano_destino < ano_atual:
    print("Você está preparando uma jornada ao passado.")
else:
    print("Você planeja permanecer no ano atual, talvez observando eventos atuais com outro olhar.")
```

Simulando viagens no tempo com input()

```
# Sobre a permanência, informa o usuário com base na resposta 's' ou 'n'.
if permanecer.lower() == 's':
    print("Você decidiu permanecer um tempo no destino antes de retornar. Ajuste seus recursos e aproveite a estadia.")
else:
    print("Você optou por retornar imediatamente após a chegada, minimizando sua exposição ao período histórico.")
```

Tratamento de erros na entrada do usuário

- O usuário pode inserir dados inesperados, como letras em vez de números, causando falhas na conversão de tipos.
- O uso de try-except permite capturar exceções e solicitar uma nova entrada válida sem interromper o programa.
- Mensagens de erro informativas ajudam a guiar o usuário e evitar frustrações ao preencher informações.
- Limitações como a exigência de valores dentro de um intervalo aceitável garantem maior controle sobre os dados recebidos.
    - Implementações mais avançadas podem incluir sugestões automáticas, correção de erros ou reconhecimento de padrões na entrada.
    - O tratamento adequado de erros torna programas mais robustos e confiáveis, melhorando a experiência do usuário.

Exibição de resultados com print()

- A função print() exibe informações no console e pode ser usada para fornecer feedback ao usuário.
- Permite múltiplos argumentos, separados por vírgulas, garantindo flexibilidade na exibição de dados.
- O parâmetro sep altera o separador entre os valores exibidos, permitindo personalização da saída.
- O parâmetro end controla a terminação da linha, permitindo exibições contínuas sem quebras desnecessárias.
    - O uso de f-strings (f"Texto {variavel}") torna a formatação mais clara e eficiente na construção de mensagens dinâmicas.
    - print() é uma ferramenta essencial para depuração, relatórios e interações textuais em programas Python.

Interatividade

Qual das alternativas abaixo é correta?

a) O método input() em Python sempre retorna um valor do tipo inteiro, sendo necessário convertê-lo para string quando se deseja manipular texto.

b) A função try-except permite capturar e tratar exceções, evitando que o programa seja interrompido abruptamente caso ocorra um erro durante a execução.

c) O operador == em Python é utilizado para atribuir valores a variáveis, substituindo o operador = em algumas situações específicas.

d) A função print() não permite exibir múltiplos argumentos ao mesmo tempo, sendo necessário concatenar manualmente os valores antes de exibi-los.

e) Em Python, a única forma de entrada de dados é pelo teclado, utilizando input(), pois a linguagem não permite ler informações de arquivos ou redes.

Resposta

Qual das alternativas abaixo é correta?

a) O método input() em Python sempre retorna um valor do tipo inteiro, sendo necessário convertê-lo para string quando se deseja manipular texto.

b) A função try-except permite capturar e tratar exceções, evitando que o programa seja interrompido abruptamente caso ocorra um erro durante a execução.

c) O operador == em Python é utilizado para atribuir valores a variáveis, substituindo o operador = em algumas situações específicas.

d) A função print() não permite exibir múltiplos argumentos ao mesmo tempo, sendo necessário concatenar manualmente os valores antes de exibi-los.

e) Em Python, a única forma de entrada de dados é pelo teclado, utilizando input(), pois a linguagem não permite ler informações de arquivos ou redes.

Introdução à escrita de arquivos

- A escrita de arquivos permite armazenar informações de maneira persistente, garantindo que os dados não sejam perdidos após o término da execução do programa.

- Essa funcionalidade é essencial para diversas aplicações, incluindo armazenamento de logs, geração de relatórios e configuração de sistemas.

- Em Python, a escrita de arquivos é feita por meio da função open(), que permite abrir, modificar e criar arquivos no sistema de forma eficiente.

    - O uso correto da escrita evita a necessidade de reentrada de dados, otimizando processos e garantindo maior segurança na manipulação de informações.

    - Diferentes modos de abertura de arquivos permitem a substituição de dados, adição de novos registros e a criação de arquivos do zero.

Modos de abertura de arquivos em Python

- O modo r (leitura) abre um arquivo existente para leitura, gerando um erro se o arquivo não for encontrado.
- O modo w (escrita) cria um novo arquivo ou substitui o conteúdo de um já existente, apagando qualquer informação prévia.
- O modo a (acrescentar) adiciona novos dados ao final do arquivo sem apagar o conteúdo anterior, sendo útil para logs e históricos.
- O modo x (criação exclusiva) cria um novo arquivo, mas gera um erro se o arquivo já existir, prevenindo sobrescritas acidentais.
    - Modos combinados, como r+ (leitura e escrita), permitem manipular arquivos sem apagar seu conteúdo, facilitando edições e modificações.
    - O uso adequado dos modos de abertura evita perda de dados e garante que a manipulação de arquivos seja feita com segurança e eficiência.

Trabalhando com arquivos de texto e binários

- Arquivos de texto armazenam dados legíveis por humanos, como .txt, .csv e .json, sendo amplamente utilizados em registros e relatórios.
- Arquivos binários contêm dados estruturados ou compactados, como imagens, vídeos e formatos específicos de software.
- O modo b permite manipular arquivos binários ao ser combinado com outros modos, como rb para leitura e wb para escrita.
- A manipulação correta de arquivos binários exige conhecimento sobre seu formato interno, evitando a corrupção de dados.
    - Python facilita a leitura e escrita de diferentes tipos de arquivos, possibilitando o processamento de informações diversas em múltiplos contextos.
    - A escolha entre arquivos de texto e binários depende da necessidade do programa e da forma como os dados serão utilizados.

O uso da codificação UTF-8 em arquivos

- A codificação UTF-8 é um padrão internacional amplamente utilizado para armazenar e manipular textos em diferentes idiomas.
- Em Python, ao abrir um arquivo de texto, é possível especificar a codificação usando o argumento encoding="utf-8", garantindo compatibilidade com caracteres especiais.
- A não especificação da codificação pode gerar problemas ao ler e escrever arquivos que contenham acentos, símbolos e caracteres não ASCII.
- O Unicode Consortium define o UTF-8 como a melhor prática para armazenar texto de forma portátil e sem perdas de informação.
    - Sistemas modernos adotam UTF-8 como padrão para evitar incompatibilidades entre diferentes plataformas e softwares.
    - Trabalhar com codificação correta garante que textos sejam exibidos corretamente, evitando erros de leitura e escrita.

Escrita sequencial e estruturada em arquivos

- A escrita pode ser realizada de forma sequencial, na qual os dados são adicionados linha por linha, organizando informações de maneira estruturada.
- O uso do modo a permite construir históricos de dados, como logs de sistema e diários de eventos, sem substituir informações anteriores.
- O método write() adiciona uma string ao arquivo, sendo necessário incluir \n manualmente para garantir a quebra de linha.
- O método writelines() permite escrever múltiplas linhas de uma só vez, otimizando operações que envolvem grandes volumes de dados.
    - O armazenamento estruturado facilita a leitura e análise posterior, tornando os arquivos mais úteis para consulta e recuperação de informações.
    - Em aplicações práticas, essa abordagem é usada para registrar atividades, armazenar configurações e gerar relatórios detalhados.

Exemplo: Registro de viagens temporais com arquivos

- O armazenamento de registros de viagens temporais permite documentar experiências e aprendizados em diferentes períodos históricos.
- Um programa pode solicitar ao usuário o ano visitado e uma breve descrição, salvando esses dados em um arquivo de texto para consulta futura.
- O uso da função open("registro_viagens.txt", "a", encoding="utf-8") garante que cada nova entrada seja adicionada sem sobrescrever registros anteriores.
    - O formato do registro inclui a data e a hora do momento em que a informação foi adicionada, ajudando a contextualizar a experiência do viajante.
    - A estrutura de armazenamento baseada em arquivos possibilita a criação de um diário digital de viagens no tempo, acessível a qualquer momento.

Exemplo: Registro de viagens temporais com arquivos

```
from datetime import datetime
def registrar_viagem():
    ano_visitado = input("Insira o ano visitado em sua última viagem: ")
    descricao = input("Descreva brevemente sua experiência nessa época: ")
    data_atual = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    # Abre o arquivo 'registro_viagens.txt' no modo 'a' (append). Esse modo garante
que, se o arquivo existir, o novo registro será acrescentado ao final.
    # Caso o arquivo não exista, ele será criado. Ao final da operação, o 'with'
fecha o arquivo automaticamente, garantindo a integridade dos dados.
    with open("registro_viagens.txt", "a", encoding='utf-8') as arquivo:
                                arquivo.write(f"{data_atual} - Ano visitado:
                                {ano_visitado}, Descrição: {descricao}\n")
                                print("Registro efetuado com sucesso no arquivo
                                      'registro_viagens.txt'.")
```

Exemplo: Leitura e consulta de arquivos de registro

- A leitura de registros salvos permite ao viajante revisar experiências passadas e analisar padrões históricos de suas jornadas.
- O modo r abre o arquivo para leitura, garantindo que seu conteúdo seja preservado enquanto permite a recuperação de informações.
- A função readlines() carrega todas as linhas do arquivo em uma lista, permitindo manipulação e exibição flexível dos dados armazenados.
- O uso de os.path.exists() antes de abrir um arquivo evita erros ao tentar ler um arquivo inexistente, garantindo robustez ao programa.
    - Para evitar consumo excessivo de memória, arquivos grandes podem ser lidos linha por linha usando um loop for, processando dados de forma eficiente.
    - A implementação de sistemas de consulta baseados em arquivos facilita o acesso a informações persistentes, tornando a recuperação de dados simples e intuitiva.

Exemplo: Leitura e consulta de arquivos de registro

```
import os
def exibir_registros():
    # Antes de tentar abrir o arquivo, verifica se ele existe.
if not os.path.exists("registro_viagens.txt"):
        print("Não há nenhum registro de viagens disponível.")
        return
    with open("registro_viagens.txt", "r", encoding='utf-8') as arquivo:
        # Lê todas as linhas do arquivo.
        linhas = arquivo.readlines()
                                if not linhas:
                                        print("O arquivo de registro está vazio, nenhuma
                                viagem foi registrada ainda.")
                                        return
                                print("Registros de viagens:\n")
                                for linha in linhas:
                                    print(linha.strip())
```

Uso de JSON para armazenamento de configurações

- O uso de json.dump() permite converter um dicionário Python em JSON e salvá-lo em um arquivo de forma padronizada.

```
with open(nome_arquivo, "w", encoding='utf-8') as arquivo:
    json.dump(configuracoes, arquivo, indent=4)
print(f"Configurações salvas com sucesso em {nome_arquivo}.")
```

- Para recuperar as configurações, json.load() lê o arquivo e reconstrói os dados como um dicionário, permitindo restauração rápida do estado do sistema.

```
with open(nome_arquivo, "w", encoding='utf-8') as arquivo:
    configuracoes = json.load(arquivo)
print(f"Configurações carregadas com sucesso de {nome_arquivo}.")
```

```
config_iniciais = {
    "modo_de_viagem": "gradual",
    "energia_inicial": 5000,
    "sensibilidade_controle": 0.8,
    "recursos_seguranca": True
}
```

Interatividade

Qual das alternativas abaixo é correta?

a) O modo de abertura "w" no comando open() permite escrever em um arquivo sem risco de perder os dados anteriores, pois adiciona novas informações ao final do arquivo.

b) O uso da declaração with ao abrir um arquivo em Python é opcional e não influencia no fechamento automático do arquivo após a execução do bloco de código.

c) O formato JSON é utilizado para armazenar dados estruturados de maneira padronizada, sendo amplamente compatível com diferentes linguagens de programação.

d) A função readlines() em Python permite modificar diretamente o conteúdo de um arquivo, sem a necessidade de reabri-lo em outro modo de escrita.

e) O método write() é utilizado apenas para escrita de arquivos binários, não podendo ser aplicado em arquivos de texto convencionais.

Resposta

Qual das alternativas abaixo é correta?

a) O modo de abertura "w" no comando open() permite escrever em um arquivo sem risco de perder os dados anteriores, pois adiciona novas informações ao final do arquivo.

b) O uso da declaração with ao abrir um arquivo em Python é opcional e não influencia no fechamento automático do arquivo após a execução do bloco de código.

c) O formato JSON é utilizado para armazenar dados estruturados de maneira padronizada, sendo amplamente compatível com diferentes linguagens de programação.

d) A função readlines() em Python permite modificar diretamente o conteúdo de um arquivo, sem a necessidade de reabri-lo em outro modo de escrita.

e) O método write() é utilizado apenas para escrita de arquivos binários, não podendo ser aplicado em arquivos de texto convencionais.

Referências

- ALVES, E. *Estruturas de dados com Python*. São Paulo: Novatec, 2022.
- FURTADO, A. L. *Python*: programação para leigos. Rio de Janeiro: Alta Books, 2021.
- NILO, L. E. *Introdução à programação com Python.* São Paulo: Novatec, 2019.
- RAMALHO, L. *Fluent Python*. 2. ed. O'Reilly Media, 2022.
- RAMALHO, L. *Python fluente*: programação clara, concisa e eficaz. São Paulo: Novatec, 2015.
- SWEIGART, A. *Automatize tarefas maçantes com Python*: programação prática para verdadeiros iniciantes. São Paulo: Novatec, 2015.
- VAN ROSSUM, G.; DRAKE, F. L. *Python reference manual*. PythonLabs, 2007.

ATÉ A PRÓXIMA!