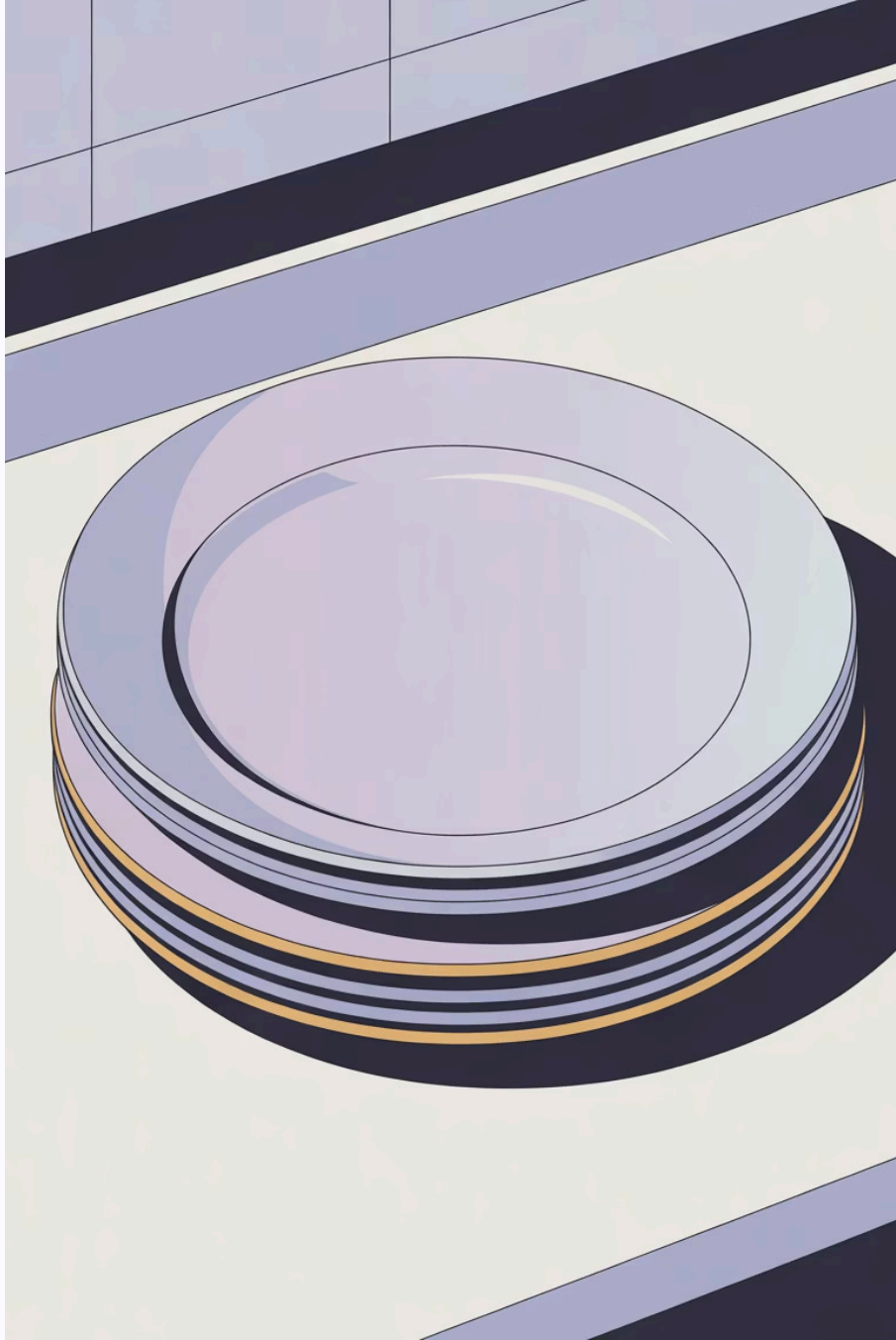
Listas como Pilhas em Python

Domine estruturas de dados fundamentais e aprenda a manipular pilhas, filas e realizar operações essenciais com listas

Explorar Pilhas

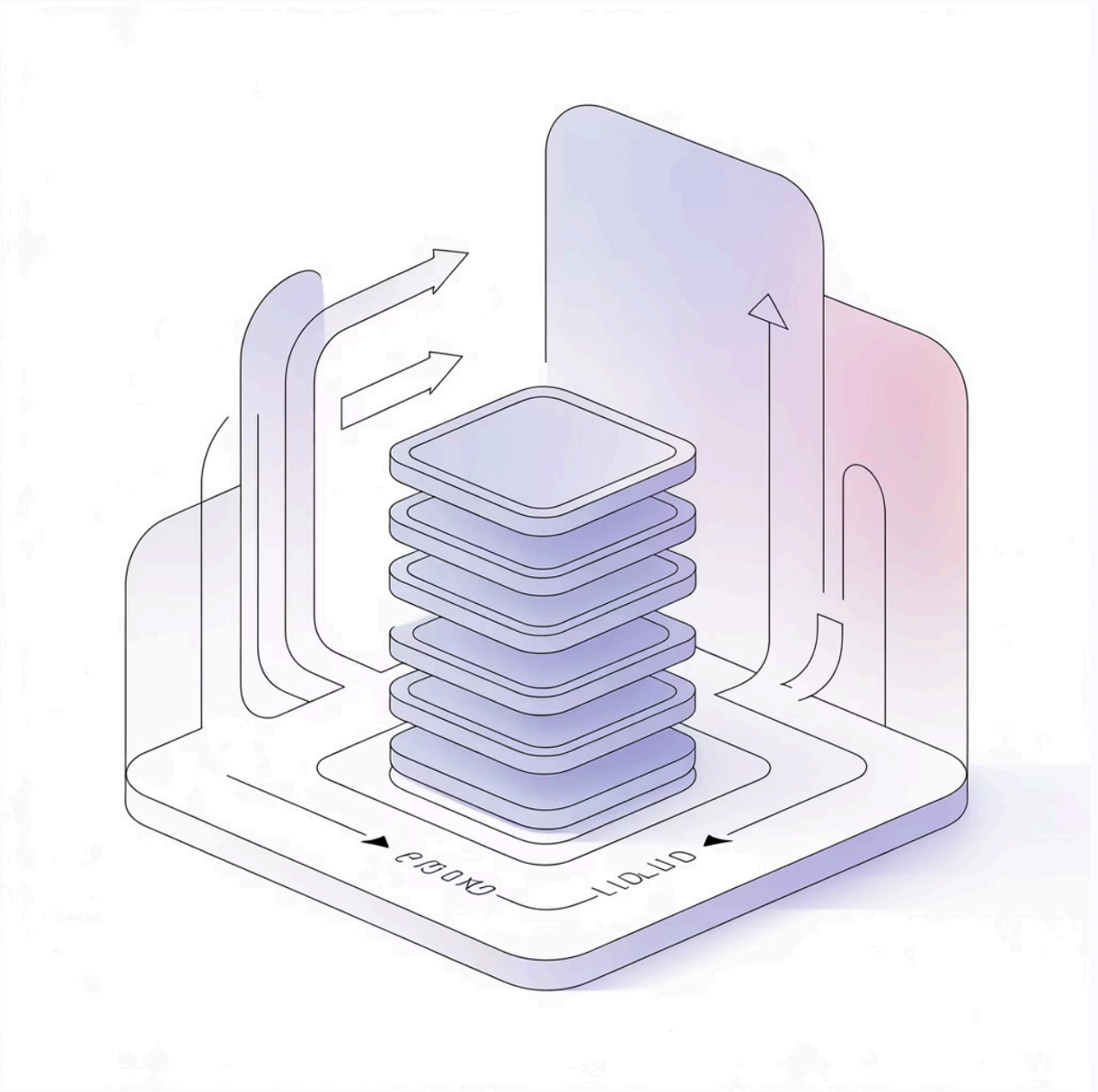
Ver Pesquisa



O Conceito de Pilha

Uma pilha tem uma política de acesso bem definida: novos elementos são adicionados ao topo. A retirada de elementos também é feita pelo topo.

Na pilha, o último elemento a chegar é o primeiro a sair (**LIFO - Last In First Out**). É como uma pilha de pratos para lavar - retiramos o prato do topo e novos pratos são empilhados sobre os anteriores.



Listas como Filas (FIFO)

Uma lista pode ser utilizada como fila se obedecermos a certas regras de inclusão e eliminação de elementos. Em uma fila, a inclusão é sempre realizada no fim, e as remoções são feitas no início. Dizemos que o primeiro a chegar é o primeiro a sair (FIFO - First In First Out).

Cliente Chega

Entra no fim da fila

Aguarda na Fila

Espera sua vez

Sai da Fila

Atendimento concluído

É Atendido

Primeiro da fila

Método pop(): Removendo e Retornando

Para retirarmos um cliente da fila e atendê-lo, poderíamos fazer del fila[0], porém, isso apagaria o cliente da fila. Se quisermos retirá-lo da fila e, ao mesmo tempo, obter o elemento retirado, podemos utilizar o método pop.

O método pop retorna o valor do elemento e o exclui da fila. Passamos 0 como parâmetro para indicar que queremos excluir o primeiro elemento.

```
fila = [1, 2, 3, 4, 5]
atendido = fila.pop(0)
# atendido = 1
# fila = [2, 3, 4, 5]
```

Fila Original

[1, 2, 3, 4, 5]

pop(0)

Remove primeiro

Retorna 1

Valor removido

Nova Fila

[2, 3, 4, 5]

Simulação de Fila de Banco

Vamos imaginar uma lista inicial com 10 clientes. Se outro cliente chegar, realizaremos um append para que ele seja inserido no fim da fila. Para atendermos um cliente, usamos pop(0) para retirá-lo da fila e obter seu valor.

```
último = 10
fila = list(range(1, último+1))
while True:
    print("\nExistem %d clientes na fila" % len(fila))
    print("Fila atual:", fila)
    print("Digite F para adicionar um cliente ao fim da fila,")
    print("ou A para realizar o atendimento. S para sair.")
    operação = input("Operação (F, A ou S):")
    if operação == "A":
        if(len(fila)) > 0:
            atendido = fila.pop(0)
            print("Cliente %d atendido" % atendido)
        else:
            print("Fila vazia! Ninguém para atender.")
    elif operação == "F":
        último += 1
        fila.append(último)
    elif operação == "S":
        break
    else:
        print("Operação inválida! Digite apenas F, A ou S!")
```

Operações Principais com Listas

Criar

Inicializar listas vazias ou com valores

```
L = []
L = [1, 2, 3]
```

Acessar

Ler e modificar elementos por índice

```
valor = L[0]
L[0] = 10
```

Adicionar

Inserir novos elementos

```
L.append(4)
L.extend([5, 6])
```

Remover

Excluir elementos da lista

```
del L[0]
L.pop(0)
```

Fatiar

Extrair partes da lista

```
L[1:3]
L[:5]
```

Medir

Obter tamanho da lista

```
tamanho = len(L)
```

Implementando uma Pilha de Pratos

O programa simula uma pia de cozinha cheia de pratos, demonstrando as operações fundamentais de uma pilha.

01

Inicialização

Cria uma lista com 5 pratos numerados de 1 a 5

02

Empilhar (E)

Adiciona um novo prato ao topo da pilha usando append()

03

Desempilhar (D)

Remove o último prato usando pop(-1) para lavar

04

Verificação

Confere se a pilha está vazia antes de desempilhar

Código: Pilha de Pratos

```
prato = 5
pilha = list(range(1,prato+1))
while True:
    print("Existem %d pratos na pilha" % len(pilha))
    print("Pilha atual:", pilha)
    operação = input("Operação (E, D ou S):")
    if operação == "D": # Desempilhar
        if(len(pilha))>0:
            lavado = pilha.pop(-1)
            print("Prato %d lavado" % lavado)
        else:
            print("Pilha vazia!")
    elif operação == "E": #Empilhar
        prato+=1
        pilha.append(prato)
    elif operação == "S": #Sair
        break
```

A diferença entre pilhas e filas está no elemento escolhido para retirar. Em pilhas, retira-se o último elemento passando -1 ao método pop.

Pilhas no Navegador Web

Histórico de Navegação

O histórico do browser funciona como uma pilha. Cada novo link adiciona uma página ao histórico. Ao clicar em voltar, o browser utiliza a última página que entrou no histórico.

Na realidade, o sistema é mais complexo: permite voltar várias vezes e depois avançar. Funciona como duas pilhas - uma à esquerda e outra à direita.

Mecânica das Duas Pilhas

Voltar

Desempilha elemento da pilha esquerda e empilha na direita

Avançar

Desempilha elemento da pilha direita e empilha na esquerda

Novo Link

Adiciona à pilha esquerda e apaga a pilha direita

Alternativa: Essas operações podem ser simuladas com uma lista simples e uma variável contendo a posição atual no histórico. A cada novo endereço, adiciona-se um elemento e atualiza-se a posição.

Exercício: Validação de Parênteses

Desafio

Faça um programa que leia uma expressão com parênteses e verifique se foram abertos e fechados na ordem correta usando pilhas.

Exemplos:

- `(())` → OK
- `()()(()())` → OK
- `())` → Erro

01

Empilhar

Adicione à pilha ao encontrar "("

02

Desempilhar

Remova da pilha ao encontrar ")"

03

Verificar

Confirme que o topo é "("

04

Validar

Pilha vazia = expressão correta

Pesquisa Sequencial em Listas

Podemos pesquisar se um elemento está em uma lista verificando do primeiro ao último elemento se o valor procurado está presente.

Comparação

Compara todos os elementos da lista com o valor procurado

Interrupção

Para ao encontrar o primeiro elemento igual ao procurado

Validação

Usa variável booleana para verificar se o valor foi encontrado

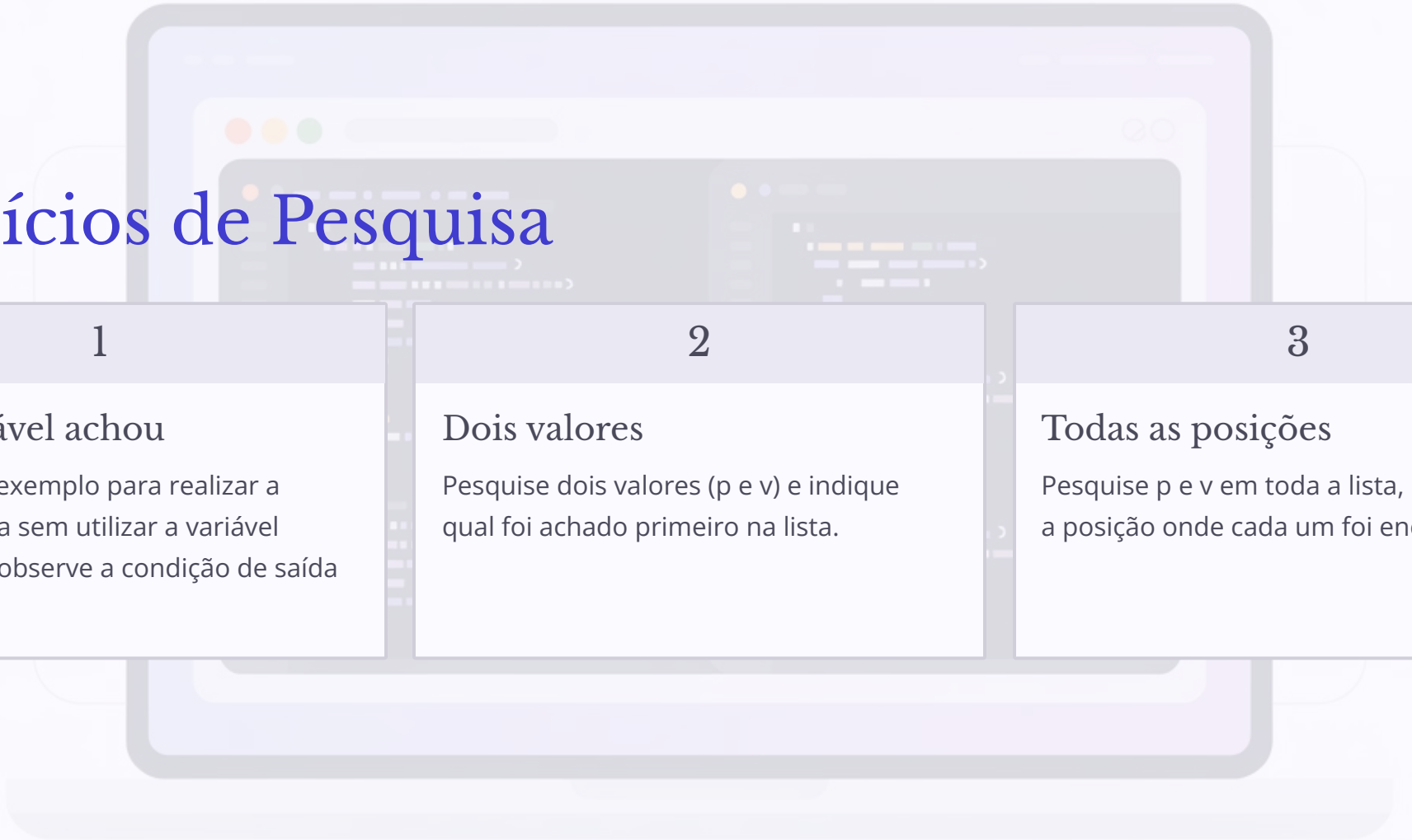
Código: Pesquisa Sequencial

```
L=[15,7,27,39]
p=int(input("Digite o valor a procurar:"))
achou=False

x=0
 while x<len(L):
    if L[x]==p:
       achou=True
       break
    x+=1
 if achou:
    print("%d achado na posição %d" % (p,x))
 else:
    print("%d não encontrado" % p)
```

A variável achou é do tipo booleano e será True apenas se algum elemento for igual ao valor procurado. Ela é marcada como True dentro do if com a condição de pesquisa, antes do break.

Exercícios de Pesquisa

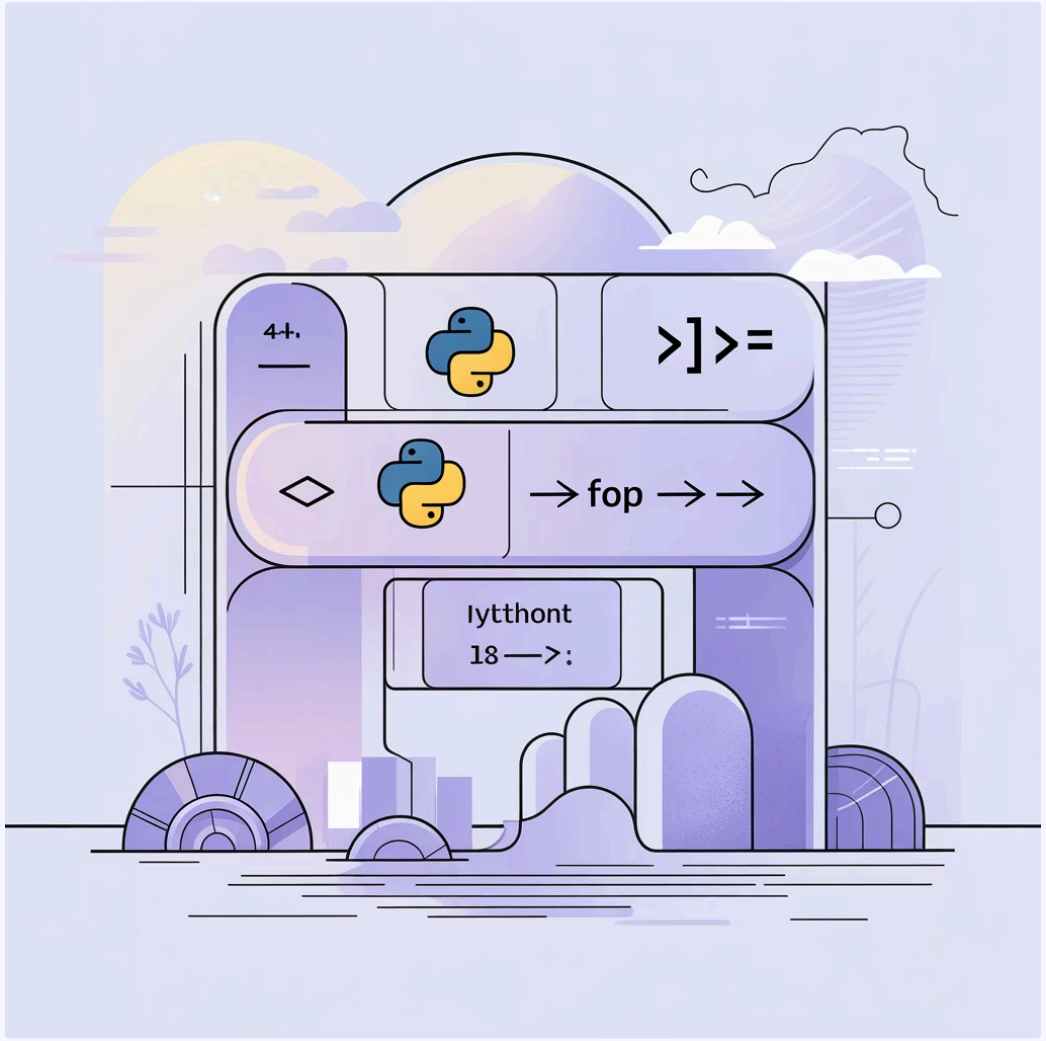


1	2	3
Sem variável achou	**Dois valores**	**Todas as posições**
Modifique o exemplo para realizar a mesma tarefa sem utilizar a variável achou. Dica: observe a condição de saída do while.	Pesquise dois valores (p e v) e indique qual foi achado primeiro na lista.	Pesquise p e v em toda a lista, informando a posição onde cada um foi encontrado.

A Instrução For em Python

Python apresenta uma estrutura de repetição especialmente projetada para percorrer listas. A instrução `for` funciona de forma parecida a `while`, mas a cada repetição utiliza um elemento diferente da lista.

A cada repetição, o próximo elemento da lista é utilizado, repetindo-se até o fim da lista. É ideal quando queremos processar os elementos um a um.



For vs While: Comparação

Usando For

```
L=[8,9,15]
for e in L:
    print(e)
```

Mais simples e direto para percorrer listas sequencialmente

Usando While

```
L=[8,9,15]
x=0
    while x<len(L):
        e=L[x]
        print(e)
        x+=1
```

Requer controle manual do índice

Quando usar cada um: Use `for` para processar elementos sequencialmente. Use `while` quando não souber quantas repetições serão necessárias ou quando manipular índices de forma não sequencial.

Pesquisa com For e Break

A instrução `break` também interrompe o `for`, permitindo sair da repetição antes do fim da lista.

```
L=[7,9,10,12]
p=int(input("Digite um número a pesquisar:"))
for e in L:
    if e == p:
        print("Elemento encontrado!")
        break
else:
    print("Elemento não encontrado.")
```

Break

Interrompe a busca após encontrar o primeiro elemento

Else do For

Executado apenas se todos os elementos forem visitados sem usar break

A Função Range

A função `range` gera listas simples de forma eficiente. Ela não retorna uma lista propriamente dita, mas um gerador (*generator*).

Um Parâmetro

range(10) gera números de 0 a 9

Dois Parâmetros

range(5,8) gera 5, 6 e 7 (início e fim)

Três Parâmetros

range(0,10,2) gera pares entre 0 e 10 (com saltos)

Exemplos Práticos com Range

Básico

```
for v in range(10):
    print(v)
```

Imprime de 0 a 9

Com Intervalo

```
for v in range(5,8):
    print(v)
```

Imprime 5, 6 e 7

Com Saltos

```
for t in range(3,33,3):
    print(t)
```

Múltiplos de 3

Para transformar um gerador em lista, utilize: L=list(range(100,1100,50))

A Função Enumerate

Com `enumerate` podemos ampliar as funcionalidades de `for` facilmente, obtendo tanto o índice quanto o valor.

Sem Enumerate

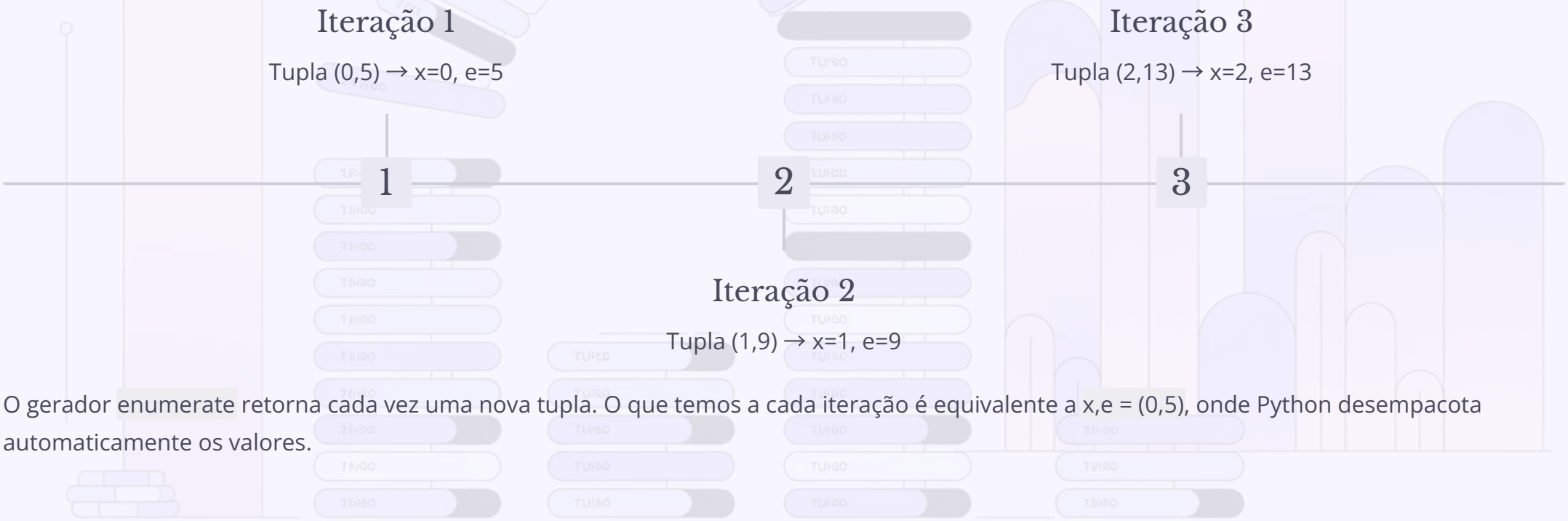
```
L=[5,9,13]
x=0
for e in L:
    print("[%d] %d" % (x,e))
    x+=1
```

Com Enumerate

```
L=[5,9,13]
for x, e in enumerate(L):
    print("[%d] %d" % (x,e))
```

A função `enumerate` gera uma tupla onde o primeiro valor é o índice e o segundo é o elemento. Python permite o desempacotamento de valores da tupla, atribuindo cada elemento a uma variável.

Como Enumerate Funciona



O gerador enumerate retorna cada vez uma nova tupla. O que temos a cada iteração é equivalente a x,e = (0,5), onde Python desempacota automaticamente os valores.

Encontrando Máximo e Mínimo

Podemos percorrer uma lista para verificar o menor e o maior valor usando comparações simples.

```
L=[1,7,2,4]
máximo=L[0]
for e in L:
    if e > máximo:
        máximo = e
print(máximo)
```

Inicializamos `máximo` com o primeiro elemento da lista. Esse truque garante que temos um valor válido antes da primeira comparação, funcionando mesmo com valores negativos.

Exercícios de Operações com Listas

1

Menor Elemento

Altere o programa para imprimir o menor elemento da lista em vez do maior.

2

Estatísticas de Temperatura

A lista T = [-10, -8, 0, 1, 2, 5, -2, -4] contém temperaturas de Mons, Bélgica. Imprima a menor, maior e temperatura média.

Importância das Estruturas de Dados

A importância de aprender a manipular listas como filas ou pilhas é entender algoritmos mais complexos no futuro.

Pilhas (LIFO)

Último a entrar, primeiro a sair. Essencial para histórico de navegação, desfazer/refazer, e avaliação de expressões.

Filas (FIFO)

Primeiro a entrar, primeiro a sair. Fundamental para processamento de tarefas e gerenciamento de recursos.

Pesquisa

Técnicas de busca sequencial formam a base para algoritmos mais sofisticados de busca e ordenação.

Iteração

Dominar for, while, range e enumerate permite processar dados de forma eficiente e elegante.

Resumo: Dominando Listas em Python

Estrutura Versátil

Listas armazenam múltiplos valores de qualquer tipo, com tamanho dinâmico e acesso por índice começando em 0.

Operações Essenciais

Domine append(), extend(), pop(), del, len() e fatiamento para manipular listas com eficiência.

Cuidados Importantes

Lembre-se: atribuição cria referências, não cópias. Use [:] para copiar. Índices válidos vão de 0 a len(L)-1.

Listas são fundamentais em Python e permitem criar programas poderosos e flexíveis. Com prática, você dominará essa estrutura de dados essencial e poderá construir aplicações cada vez mais sofisticadas.

Praticar exercícios | Próxima lição