Este resumo aborda os conceitos essenciais de Algoritmos e Lógica de Programação, com exemplos práticos em Python, baseados no conteúdo do material.

1. Algoritmos e Estruturas de Dados Básicas

Um **algoritmo** é um conjunto de passos claros, bem definidos e em ordem lógica, usado para resolver um problema ou realizar uma tarefa. Pense nele como uma receita de bolo ou instruções de montagem. O computador precisa dessas **instruções** para saber o que fazer.

Um algoritmo funciona recebendo uma **entrada**, seguindo os **passos pré-definidos**, e entregando uma **saída** (o resultado final).

2. Variáveis e Tipos de Dados

Variáveis

Uma **variável** é um nome que aponta para um valor armazenado na memória do computador, funcionando como uma "caixa" para guardar informações que podem ser usadas depois.

Em Python, **toda variável é um objeto**. Isso significa que ela é uma referência a uma estrutura de dados que armazena tanto o valor quanto seus comportamentos (métodos).

- **Boas Práticas:** Use nomes descritivos (ex: `nome_completo` em vez de `nc`) para facilitar a compreensão do código. Variáveis devem começar com uma letra ou *underscore* (`_`) e são *case sensitive*.

Tipos Primitivos Básicos

Python lida com quatro tipos de dados básicos:

- **int (Inteiro):** Números sem parte decimal (ex: $-4, 0, 1987$)[10].
- **float (Flutuante):** Números reais, com parte decimal (ex: $-15.2, 3.14$)[11111111].
- **str (String):** Sequência de caracteres (texto), delimitada por aspas (simples ou duplas)[12].
- **bool (Booleano):** Valores lógicos **True** (Verdadeiro) ou **False** (Falso)[13].

```
# Exemplos de Variáveis e Tipos de Dados
idade = 25              # int
peso = 72.5             # float
nome = 'Ana'            # str
acordado = True         # bool

print(f"Nome: {nome}, Idade: {idade}")
```

3. Comandos de Entrada e Saída

Funções `print()` e `input()`

- **`print()`**: Usada para exibir mensagens e resultados na tela (saída).
- **`input()`**: Usada para pedir dados ao usuário (entrada). Por padrão, a entrada do `input()` é sempre lida como uma *string* (`str`).

```
# Exemplo de Entrada e Saída
nome = input('Qual o seu nome? ') # Recebe como string
idade = int(input('Qual a sua idade? ')) # Converte a entrada para
int
cidade = 'Goiânia'

# Saída formatada com f-string (Python 3.6+)
print(f'Olá, {nome}. Você tem {idade} anos e mora em {cidade}.')
```

4. Operadores Aritméticos

Os operadores realizam cálculos. Se você usa o `input()` para números, lembre-se de converter a *string* para `int()` ou `float()` antes de operar.

Operador	Descrição	Exemplo	Resultado
`+`	Adição	5 + 3	8
`-`	Subtração	7 - 2	5
`*`	Multiplicação	4 * 3	12
`/`	Divisão	10 / 4	2.5
`//`	Divisão Inteira	10 // 3	3
`%`	Módulo (Resto)	10 % 3	1
`**`	Exponenciação	2 ** 3	8

Ordem de Precedência (PEMDAS)

A ordem em que as operações são calculadas é fixa, seguindo a regra **PEMDAS**:

1. **P**arênteses
2. **E**xpoentes
3. **M**ultiplicação
4. **D**ivisão

5. **A**dição
6. **S**ubtração

```
# Exemplo de Operadores e Precedência
resultado = 5 + (6 - 2) * 3**2 # 5 + 4 * 9 => 5 + 36
print(f"O resultado é: {resultado}") # Saída: 41
```

5. Condicionais (Tomada de Decisão)

Condicionais são "perguntas" que o programa faz para decidir qual bloco de código executar.

- **`if` (Se):** O comando inicial que testa uma condição.
- **`else` (Se não/Outra):** O "plano B" que é executado se o `if` não for verdadeiro.
- **`elif` (Se não se):** Usado para testar condições adicionais se o `if` inicial for falso.

Estrutura Completa

Você só pode ter um **`if`** e um **`else`** em um mesmo bloco, mas pode ter quantos **`elif`** quiser. O Python testa a ordem de cima para baixo e para no primeiro que for verdadeiro.

```
# Exemplo de Condicionais (if, elif, else)
nota = float(input("Digite sua nota: "))

if nota == 10:
    print("Nota máxima! Parabéns!")
elif nota >= 7:
    print("Aprovado")
elif nota >= 5:
    print("Recuperação")
else: # Executado se nota < 5
    print("Reprovado")
```

Condicionais Aninhadas

Ocorre quando um `if` está dentro de outro `if`. Você só testa a segunda condição se a primeira for verdadeira. É útil quando uma condição depende da outra.

```
# Exemplo de Condicionais Aninhadas
passou_ano = input("Passou de ano? (s/n): ").strip().lower()

if passou_ano == "s":
    passou_vestibular = input("Passou no vestibular? (s/n):
").strip().lower()
    if passou_vestibular == "s":
        print("🎉 Começará a faculdade no próximo ano.")
    else:
        print("Vou precisar tentar o vestibular novamente.")
else:
    print("Preciso estudar mais para passar de ano.")
```

6. Repetição/Laços

Estruturas de repetição executam um bloco de código várias vezes.

Estrutura `while`

O `while` repete um bloco de código **ENQUANTO** (só se) uma condição for atendida (`True`). É comumente usado quando o número de repetições é **desconhecido**.

```
# Exemplo de while (Contador)
n = 1
while n < 10: # A condição precisa se tornar False para parar
    print(n)
    n += 1 # n = n + 1 (incrementa o contador) [cite: 753, 754,
755, 756]
# Saída: 1 a 9
```

Estrutura `for`

O `for` repete um bloco de código percorrendo uma **sequência** (como uma lista ou um intervalo de números). Esse processo é chamado de **iteração**.

Função `range()`

Gera uma sequência de números inteiros para o `for`. Ela inclui o **`start`** e exclui o **`stop`**.

- **`range(stop)`:** De 0 até (stop - 1).
- **`range(start, stop)`:** De *start* até (*stop* - 1).
- **`range(start, stop, step)`:** De *start* até (*stop* - 1), pulando de *step* em *step*.

```
# Exemplo de for e range
print("Contagem ímpar de 1 a 10:")
for i in range(1, 11, 2): # Start=1, Stop=11, Step=2
    print(i)
# Saída: 1, 3, 5, 7, 9
```

Ferramentas de Controle

- **Contador:** Variável que conta o número de repetições (ex: `i = 0`, `i += 1`).
- **Acumulador:** Variável que acumula ou soma valores variáveis ao longo do *loop* (ex: `soma += num`).
- **`break`:** Palavra-chave que funciona como uma "porta de saída" para o laço, encerrando-o imediatamente, mesmo que a condição do *loop* ainda seja `True`.

```
# Exemplo de Acumulador e break
soma = 0 # Acumulador
num = int(input("Digite um número (ou 0 para parar): "))

while num != 0:
    soma += num # Acumula o valor digitado
    if soma > 50:
        print("Soma ultrapassou 50! Parando...")
```

```
        break # Sai do loop
    num = int(input("Digite outro número (ou 0 para parar): "))

print("Soma total:", soma)
```

7. Listas

Uma **Lista** é uma coleção de dados fundamental em Python, permitindo armazenar vários valores em uma única variável, definidos por colchetes [ ]

Operação	Explicação	Exemplo de Código
Acessar	Elementos são acessados pelo **índice** (a posição, começando em 0)	`lista[0]`
Modificar	Altera um elemento em uma posição específica	`lista[1] = 'kiwi'`
append()	Adiciona um elemento **ao final** da lista	`lista.append('uva')`
insert()	Adiciona um elemento em uma **posição específica**	`lista.insert(1, 'melancia')`
remove()	Remove o **primeiro valor** correspondente	`lista.remove('banana')`

del	Remove um elemento por **índice/posição**	**del lista[2]**
.sort()	**Ordena** a lista original (no local)	**lista.sort()**
sorted()	Retorna uma **nova lista ordenada**	**nova_lista = sorted(lista)**
max() / min()	Retorna o maior/menor elemento (numérico ou alfabético)	**max(numeros)**

Scripts de Operações com Listas em Python

Usaremos a lista `candidatos` como nossa lista base para os exemplos.

```
# Lista base para os exemplos
candidatos = ['Ana', 'Bernardo', 'Carla', 'David', 'Eva',
'Felipe']
print(f"Lista Inicial: {candidatos}")
print("-" * 30)
```

1. Acessar e Modificar Elementos

Para **acessar** um elemento, usamos o índice (posição), que começa em 0. Para **modificar**, atribuímos um novo valor ao índice específico.

```
# Acessar um elemento (o segundo item, índice 1)
primeiro_nome = candidatos[0]
print(f"Acessando o primeiro elemento: {primeiro_nome}")

# Modificar um elemento (o terceiro item, índice 2)
```

```
candidatos[2] = 'Carol'
print(f"Lista após modificação do índice 2: {candidatos}")
print("-" * 30)
```

2. Adicionar Elementos

Método	Descrição
`.append()`	Adiciona o elemento **ao final** da lista.
`.insert()`	Adiciona o elemento em uma **posição específica** (posição, valor).

```
# Adicionar no final com .append()
candidatos.append('Giovana')
print(f"Lista após .append('Giovana'): {candidatos}")

# Adicionar em uma posição específica com .insert(posição, valor)
candidatos.insert(3, 'Hugo') # Adiciona 'Hugo' na posição 3
print(f"Lista após .insert(3, 'Hugo'): {candidatos}")
print("-" * 30)
```

3. Remover Elementos

Método	Descrição
`.remove()`	Remove o primeiro elemento que possui o **valor** especificado[5].

`del`	Remove um elemento pelo seu **índice/posição**[6].

```
# Remover por valor com .remove()
candidatos.remove('David')
print(f"Lista após .remove('David'): {candidatos}")

# Remover por índice com del
del candidatos[4] # Remove o item que está no índice 4 ('Eva' ou o que sobrou após remoção)
print(f"Lista após del candidatos[4]: {candidatos}")
print("-" * 30)
```

4. Ordenação

Método	Descrição
`.sort()`	Modifica a lista **original** no local (ordem crescente/alfabética)[7].
`sorted()`	Cria e retorna uma **nova lista ordenada**, sem alterar a original[8].

```
# Ordenar a lista original com .sort()
candidatos.sort()
print(f"Lista após .sort() (Ordem Alfabética): {candidatos}")

# Ordenar a lista em ordem decrescente (do maior para o menor)
candidatos.sort(reverse=True)
print(f"Lista após .sort(reverse=True): {candidatos}")

# Criar uma nova lista ordenada sem alterar a original (sorted())
```

```
nova_lista = sorted(candidatos)
print(f"Nova lista criada com sorted(): {nova_lista}")
print("-" * 30)
```

5. Funções Auxiliares (`max` e `min`)

As funções `max()` e `min()` retornam o maior e o menor valor, respectivamente. Para *strings* (textos), o critério é a ordem alfabética.

```
# Exemplo com números
numeros = [15, 8, 42, 2, 70]
maior_numero = max(numeros)
menor_numero = min(numeros)

print(f"Lista de Números: {numeros}")
print(f"Maior número (max): {maior_numero}")
print(f"Menor número (min): {menor_numero}")

# Exemplo com strings (usando a lista atual de candidatos)
maior_nome = max(candidatos)
menor_nome = min(candidatos)

print(f"Lista de Nomes: {candidatos}")
print(f"Maior nome (último em ordem alfabética): {maior_nome}")
print(f"Menor nome (primeiro em ordem alfabética): {menor_nome}")
```