

---

## Semana 20 - Aula 1

**Tópico Principal da Aula:** Estruturas de Repetição: Laço de Repetição do-while

Subtítulo/Tema Específico: Vantagens e Conceito do Laço do-while e sua Emulação em Python

Código da aula: [SIS]ANO1C1B3S20A1

### Objetivos da Aula:

- Conhecer os conceitos sobre o desenvolvimento e execução prática de programas computacionais utilizando estruturas de repetição.
- Desenvolver sistemas computacionais utilizando ambiente de desenvolvimento.
- Resolver problemas computacionais com estratégias criativas.

### Recursos Adicionais:

- Caderno para anotações.
- Acesso ao laboratório de informática e/ou internet.

### Exposição do Conteúdo:

**Referência do Slide:** 09 - Ponto de Partida

- **Definição:** O laço (ou loop) do-while é uma estrutura de controle que executa um bloco de código ao menos uma vez e, ao final dessa execução, avalia uma condição. Se a condição for verdadeira, o bloco de código é executado novamente. Esse ciclo se repete até que a condição se torne falsa. A principal vantagem do do-while é a garantia de que o código será executado no mínimo uma vez.
- **Aprofundamento/Complemento:** Embora seja comum em linguagens como C++, Java e C#, o laço do-while não existe nativamente em Python. A filosofia de Python, expressa em "The Zen of Python", preza pela clareza e simplicidade, e os desenvolvedores da linguagem consideraram que a funcionalidade do do-while poderia ser facilmente replicada com um while True e uma instrução break, evitando a necessidade de adicionar uma nova sintaxe à linguagem.
- **Exemplo Prático:** O do-while é particularmente útil em menus interativos, onde as opções devem ser exibidas ao usuário pelo menos uma vez antes que ele decida sair. Outro uso comum é na validação de entradas, onde o programa solicita um dado ao usuário e continua solicitando até que a entrada seja válida.

**Referência do Slide:** 10 - Ponto de partida

- **Definição:** O laço do-while é adequado para situações onde a condição de continuação do loop depende de um valor que é calculado ou modificado dentro do próprio loop. Isso simplifica a lógica, pois não é necessário inicializar variáveis com valores artificiais apenas para garantir a primeira entrada no loop, o que poderia ser necessário em um while tradicional. Em certos casos, seu uso pode reduzir a

duplicação de código, pois a ação que precisa ocorrer antes da verificação já está dentro do laço.

- **Exemplo Prático:** Imagine um programa que lê dados de um sensor. Ele precisa ler o primeiro dado para então decidir se continua ou não a leitura (por exemplo, se o valor lido está dentro de uma faixa esperada). Com um `do-while`, a lógica seria: "leia o dado; enquanto o dado for válido, continue lendo".

#### Referência do Slide: 13 e 14 - Construindo o conceito

- **Definição:** Para emular o comportamento de um `do-while` em Python, a abordagem padrão é utilizar um loop `while True:`, que cria um laço infinito, e dentro dele, colocar uma estrutura condicional (`if`) que execute uma instrução `break` para sair do laço quando a condição de parada for atendida. Isso garante que o bloco de código anterior ao `if` seja executado pelo menos uma vez.
- **Aprofundamento/Complemento:** A principal diferença para um laço `while` tradicional é que o `while` comum testa a condição *antes* de executar o bloco de código pela primeira vez. Se a condição for inicialmente falsa, o bloco nunca é executado. Na emulação do `do-while`, o bloco sempre executa ao menos uma vez, e o teste da condição é feito *no final* do bloco.
- video: **Curso Python #15 - Interrompendo repetições while**
- [https://youtu.be/1OFp\\_-R2B2A?si=CQp8HRtsrv0dm19j](https://youtu.be/1OFp_-R2B2A?si=CQp8HRtsrv0dm19j)
- **Exemplo Prático:**

```
atividades_praticas_S20 > exemplo_de_codigo_repeticao_do_while_2.py > ...
1  sexo = str(input("Informe seu sexo (M/F): ")).strip().upper()
   [0]
2  while True:  # Aqui é um exemplo de validação de entrada
3      if sexo in 'MF':
4          print(f"Sexo {sexo} registrado com sucesso!")
5      else:
6          print("Sexo inválido. Tente novamente.")
7          break
8
```

```
atividades_praticas_S20 > exemplo_de_codigo_repeticao_do_while.py > ...
1  while True:  # Aqui é um exemplo de menu interativo
2      print("\n---Menu---")
3      print("1.Ver Saldo")
4      print("2.Sair")
5
6      escolha = input("Escolha uma opção:")
7      if escolha == '1':
8          print("Seu Saldo é R$ 100.00")
9
10         # condição de saída
11
12     elif escolha == '2':
13         print("Saindo...")
14         break
15
```

---

## Semana 20 - Aula 2

**Tópico Principal da Aula:** Estruturas de Repetição: Laço de Repetição do-while

Subtítulo/Tema Específico: Estruturas de Repetição e o Uso de break/continue

Código da aula: [SIS]ANO1C1B3S20A2

### Objetivos da Aula:

- Conhecer os conceitos sobre o desenvolvimento e execução prática de programas computacionais utilizando estruturas de repetição.
- Migrar sistemas, implementando rotinas e estruturas de dados mais eficazes.
- Resolver problemas computacionais com estratégias criativas.

### Recursos Adicionais:

- Caderno para anotações.
- Acesso ao laboratório de informática e/ou internet.

### Exposição do Conteúdo:

#### Referência do Slide: 08 - Ponto de Partida

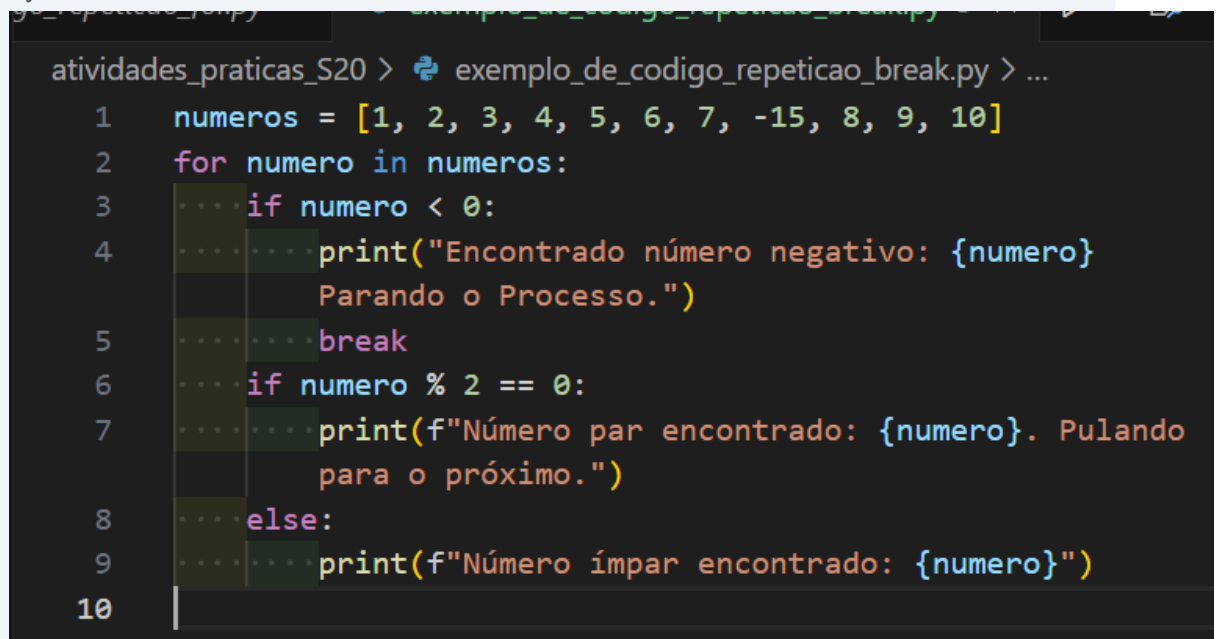
- **Definição:** Existem três tipos principais de laços de repetição em programação: **for**, **while**, e **do-while**.
  - **Laço for:** Ideal para percorrer uma sequência com um número conhecido de iterações, como uma lista, uma string ou uma faixa de números (**range**).
  - **Laço while:** Executa um bloco de código repetidamente enquanto uma condição for verdadeira. É usado quando o número de iterações não é conhecido de antemão.
  - **Laço do-while:** Garante que o bloco de código seja executado pelo menos uma vez antes de verificar a condição. Não é nativo em Python, mas pode ser emulado.
- **Aprofundamento/Complemento:** A escolha do laço correto afeta a clareza e a eficiência do código. Usar um **for** para iterar sobre os elementos de uma lista é considerado mais "Pythônico" e legível do que usar um **while** com um contador manual.

#### Referência do Slide: 13, 14 e 15 - Construindo o conceito

- **Definição:** Na emulação do **do-while**, as declarações **break** e **continue** são ferramentas estratégicas.
  - **break:** É usado para sair (interromper) o loop imediatamente, independentemente de onde o fluxo de execução esteja dentro do laço. É a peça-chave para finalizar um loop **while True**.
  - **continue:** É usado para pular o restante do código na iteração atual e passar diretamente para a próxima iteração do loop.
- **Exemplo Prático:** No exemplo dos slides, o código processa uma lista de números.

- Se encontra um número negativo, o `break` interrompe todo o processo.
- Se encontra um número par, o `continue` faz com que o `print("Processando o número...")` seja ignorado, e o loop avança para o próximo número da lista.

- Python



```
atividades_praticas_S20 > exemplo_de_codigo_repeticao_break.py > ...
1  numeros = [1, 2, 3, 4, 5, 6, 7, -15, 8, 9, 10]
2  for numero in numeros:
3      if numero < 0:
4          print("Encontrado número negativo: {numero}")
5          print("Parando o Processo.")
6          break
7      if numero % 2 == 0:
8          print(f"Número par encontrado: {numero}. Pulando para o próximo.")
9          continue
10     print(f"Número ímpar encontrado: {numero}")
```

#### Referência do Slide: 16, 17 e 18 - Construindo o conceito

- **Definição:** Ao emular um `do-while`, é importante considerar o design do código. Se a lógica se tornar muito complexa dentro de um único loop `while True:`, pode ser um sinal de que o código precisa ser refatorado. Refatorar significa reestruturar o código para torná-lo mais claro, eficiente e fácil de manter, sem alterar seu comportamento externo.
- **Aprofundamento/Complemento:** Uma boa prática de refatoração é extrair lógicas complexas para dentro de funções. No exemplo do menu, em vez de ter um grande bloco `if/elif/else` dentro do `while`, o código é refatorado em três funções: `mostrar_menu()`, `processar_escolha(escolha)` e `executar_menu()`. Isso torna o loop principal (`executar_menu`) muito mais limpo e legível.

---

## Semana 20 - Aula 3

**Tópico Principal da Aula:** Estruturas de Repetição: Laço de Repetição do-while

Subtítulo/Tema Específico: Aplicação de Estruturas de Repetição em Listas

Código da aula: [SIS]ANO1C1B3S20A3

### Objetivos da Aula:

- Conhecer os conceitos sobre o desenvolvimento e execução prática de programas computacionais utilizando estruturas de repetição.
- Desenvolver sistemas computacionais utilizando ambiente de desenvolvimento.

- Compreender os conceitos e a utilização de listas em Python e a iteração sobre elas.

#### Recursos Adicionais:

- Caderno para anotações.
- Acesso ao laboratório de informática e/ou internet.

#### Exposição do Conteúdo:

##### Referência do Slide: 07 - Ponto de partida

- **Definição:** Python, embora não tenha o laço `do-while` nativo, oferece maneiras flexíveis de emular seu comportamento. A essência do `do-while` é executar um bloco de código *antes* de testar a condição, garantindo pelo menos uma execução. A comunidade Python desenvolveu padrões para replicar essa lógica de forma eficaz, adaptando-a às melhores práticas da linguagem.
- **Aprofundamento/Complemento:** O relacionamento entre listas e laços de repetição é fundamental em programação. Laços são a principal ferramenta para acessar, processar e manipular cada elemento contido dentro de uma lista.

##### Referência do Slide: 10 - Construindo conceito

- **Definição:** Uma **lista** em Python é uma coleção ordenada e mutável de elementos. "Ordenada" significa que os itens mantêm uma posição definida (índice), e "mutável" significa que podemos alterar, adicionar e remover itens da lista depois que ela foi criada. As listas são definidas por colchetes `[]`, com os elementos separados por vírgulas.
- **Exemplo Prático:**
- Python

```
# Uma lista de strings
```

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

```
# Uma lista de números
```

```
idades = [19, 25, 42, 33]
```

```
# Uma lista mista (não é uma prática comum, mas é possível)
```

```
dados_usuario = ["Ana", 30, 1.65, True]
```

##### Referência do Slide: 11 e 12 - Situação Ser sempre +Contexto

- **Definição:** A situação-problema descreve a tarefa de analisar uma lista de feedbacks de clientes. Cada feedback é uma tupla no formato `(pontuação, categoria, comentário)`. O objetivo é usar Python para filtrar e agrupar os feedbacks críticos (com pontuação baixa).
- **Aprofundamento/Complemento:** Esta tarefa é um exemplo clássico de processamento de dados onde laços de repetição são essenciais. Para resolver o problema, é necessário:
  - Iterar sobre cada tupla na lista de feedbacks usando um laço `for`.

- Dentro do laço, usar uma estrutura condicional (if) para verificar se a pontuação é baixa (filtragem).
- Se a pontuação for baixa, armazenar o feedback de forma organizada, por exemplo, em um dicionário onde as chaves são as categorias (agrupamento).
- **Exemplo Prático:**
- Python

```
atividades_praticas_S20 > exemplo_de_codigo_repeticao_lista.py > ...
1  feedbacks = [
2      (2, "Usabilidade", "O botão de sair é difícil de
        achar"),
3      (5, "Performance", "O app é muito rápido!"),
4      (1, "Bugs", "O Aplicativo fechou sozinho"),
5      (3, "Usabilidade", "Gostaria de um modo escuro")
6  ]
7
8  feedbacks_criticos = []
9
10 for feedback in feedbacks:
11     pontuacao = feedback[0]
12     if pontuacao < 3:
13         feedbacks_criticos.append(feedback)
14     print("FEEDBACK criticos encontrados.")
15     print(feedbacks_criticos)
16
```

---

## Semana 20 - Aula 4

**Tópico Principal da Aula:** Estruturas de Repetição: Laço de Repetição do-while

**Subtítulo/Tema Específico:** Relacionando Dicionários e Estruturas de Repetição

**Código da aula:** [SIS]ANO1C1B3S20A4

### Objetivos da Aula:

- Conhecer os conceitos e a utilização de dicionários em Python.
- Compreender a utilização das estruturas de repetição para iteração dos dicionários em Python.
- Resolver problemas computacionais com estratégias criativas.

### Recursos Adicionais:

- Caderno para anotações.
- Acesso ao laboratório de informática e/ou internet.

## Exposição do Conteúdo:

### Referência do Slide: 07 - Ponto de partida

- **Definição:** Dicionários em Python são coleções não ordenadas (em versões de Python anteriores à 3.7) de pares chave-valor. Cada chave em um dicionário deve ser única e é usada para acessar seu valor correspondente. Dicionários são definidos por chaves {}.
- **Aprofundamento/Complemento:** A combinação de dicionários com laços de repetição é uma técnica poderosa para manipulação de dados. Podemos usar um laço `for` para iterar sobre as chaves, os valores, ou os pares chave-valor de um dicionário, permitindo análises complexas.

### Referência do Slide: 08 - Ponto de partida

- **Definição:** Dicionários são extremamente eficazes para agrupar e organizar dados de listas. Por exemplo, ao processar uma lista de feedbacks, podemos usar um dicionário para agrupar todos os comentários pertencentes à mesma categoria. Outra aplicação comum é a contagem de frequência de elementos em uma lista, onde cada elemento se torna uma chave no dicionário e seu valor é a contagem de ocorrências.
- **Exemplo Prático (Agrupamento):**
- Python

```
feedbacks = [
    (2, "Usabilidade", "Botão difícil de achar."),
    (1, "Bugs", "App fechou sozinho."),
    (4, "Usabilidade", "Interface limpa."),
    (2, "Bugs", "Erro ao salvar.")
]

feedbacks_agrupados = {}

for pontuacao, categoria, comentario in feedbacks:
    if categoria not in feedbacks_agrupados:
        feedbacks_agrupados[categoria] = [] # Cria a lista se a categoria não existe
    feedbacks_agrupados[categoria].append(comentario) # Adiciona o comentário à lista da categoria

print(feedbacks_agrupados)
# Saída: {'Usabilidade': ['Botão difícil de achar.', 'Interface limpa.'], 'Bugs': ['App fechou sozinho.', 'Erro ao salvar.']}
```

-

## Referência do Slide: 12 - Construindo o conceito

- **Definição:** Para aprofundar o conhecimento em dicionários, é importante conhecer seus métodos principais, como `.keys()` (retorna as chaves), `.values()` (retorna os valores) e `.items()` (retorna os pares chave-valor).
- **Aprofundamento/Complemento:** A iteração sobre um dicionário pode ser feita de várias formas:
- Python

```
atividades_praticas_S20 > exemplo_de_codigo_repeticao_dicionario.py > ...  
1  contatos = {"Ana": "111-222", "Beto": "333-444"}  
2  
3  # Iterar sobre as chaves (comportamento padrão)  
4  for nome in contatos:  
5      print(nome) # Ana, Beto  
6  
7  # Iterar sobre os valores  
8  for telefone in contatos.values():  
9      print(telefone) # 111-222, 333-444  
10  
11 # Iterar sobre os pares (chave, valor)  
12 for nome, telefone in contatos.items():  
13     print(f"{nome}: {telefone}")
```

## Referência do Slide: 13 - Colocando em prática

- **Definição:** A situação-problema apresenta dados de vendas em uma lista de tuplas (`região`, `produto`, `quantidade_vendida`, `data_da_venda`) e pede uma forma de organizar esses dados para análise.
- **Aprofundamento/Complemento:** Esta é uma oportunidade ideal para usar dicionários aninhados. Uma estrutura eficaz seria um dicionário onde as chaves são as regiões. O valor para cada região seria outro dicionário, onde as chaves são os produtos e os valores são o total de vendas daquele produto naquela região.
- **Exemplo Prático:**



atividades\_praticas\_S20 > exemplo\_de\_codigo\_repeticao\_dicionario\_2.py > ...

```
1  vendas = [  
2      ("Sul", "Produto A", 10, "2024-01-10"),  
3      ("Norte", "Produto B", 5, "2024-01-11"),  
4      ("Sul", "Produto A", 15, "2024-01-12"),  
5      ("Sul", "Produto C", 20, "2024-01-12")  
6  ]  
7  
8  analise_vendas = {}  
9  
10 for regioao, produto, quantidade, data in vendas:  
11     if regioao not in analise_vendas:  
12         analise_vendas[regiao] = {}  
13  
14     if produto not in analise_vendas[regiao]:  
15         analise_vendas[regiao][produto] = 0  
16  
17     analise_vendas[regiao][produto] += quantidade  
18  
19 print(analise_vendas)  
20 # Saída: {'Sul': {'Produto A': 25, 'Produto C': 20},  
21         'Norte': {'Produto B': 5}}
```

● Python