

# **ABAP Moderno: Escreva Menos, Realize Mais.**

**Um Guia Prático para Refatorar seu  
Código e sua Mentalidade.**

# O Desafio Diário: O Código que Conhecemos

```
" Classic ABAP Example  
REPORT z_classic_report.
```

```
DATA: gv_counter      TYPE i,  
      gs_flight       TYPE spfli,  
      gt_flights      TYPE STANDARD TABLE OF spfli,  
      gv_message       TYPE string,  
      gv_airline_name  TYPE c LENGTH 20,  
      gv_connection_id TYPE c LENGTH 4.
```

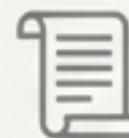
```
START-OF-SELECTION.
```

```
  SELECT * FROM spfli INTO TABLE gt_flights UP TO 10 ROWS.
```

```
  LOOP AT gt_flights INTO gs_flight.  
    gv_airline_name = gs_flight-carrname.  
    gv_connection_id = gs_flight-connid.
```

```
    CONCATENATE 'Voo:' gv_airline_name gv_connection_id  
    INTO gv_message SEPARATED BY space.
```

```
    WRITE: / gv_message.  
  ENDLOOP.
```



## Verbosidade:

Declarações no topo, lógica de execução muito abaixo.



## "Sobe e Desce":

Constante rolagem na tela para verificar tipos de variáveis.



## Lógica Dispersa:

Uso de múltiplos comandos para tarefas simples como concatenação ou preenchimento de estruturas.



## Performance:

Risco de cópias de memória desnecessárias em loops com tabelas grandes.

# A Filosofia Moderna: Declare Onde Você Usa

## Contexto

Unir a definição da variável ao seu primeiro uso, aumentando drasticamente a legibilidade e a manutenção do código.

## Concisão

Substituir blocos de código procedurais por expressões funcionais poderosas que expressam a intenção, não os passos.

## Segurança

Deixar o compilador inferir os tipos de dados, eliminando erros manuais de declaração e garantindo a compatibilidade.

## Performance

Incentivar práticas que favorecem o uso de referências (ponteiros) em vez de cópias de valor, especialmente em processamento de grandes volumes de dados.

# Revolução nº 1: O Fim da Seção `DATA`

## ANTES

### Clássico

```
DATA: ls_result TYPE zcomplex_structure,  
      lt_flights TYPE STANDARD TABLE OF flight_schedule.
```

```
lo_objeto->get_complex_data(  
  IMPORTING  
    es_result = ls_result ).
```

```
SELECT * FROM flight_schedule  
INTO TABLE lt_flights.
```

## DEPOIS

### Moderno

```
DATA(ls_result) = lo_objeto->get_complex_data( ).
```

```
SELECT * FROM flight_schedule  
INTO TABLE @DATA(lt_flights).
```

Deixe o compilador fazer o trabalho pesado. O sistema infere a estrutura exata do retorno do método ou da tabela do dicionário. Menos código, zero consultas à SE11.

# Performance e Precisão: Pare de Copiar, Comece a Apontar

## Cenário 1: WORK AREA (Cópia de Valor)

```
LOOP AT lt_tabela INTO DATA(ls_copia).
```



Lento para tabelas grandes. Cada linha é copiada para uma nova área de memória.

## Cenário 2: FIELD-SYMBOL (Referência)

```
LOOP AT lt_tabela ASSIGNING  
FIELD-SYMBOL(<fs_linha>).
```



Extremamente rápido. Cria apenas um ponteiro para a linha original, permitindo modificações diretas na tabela.

**\*\*Atenção:** Uma variável declarada inline dentro de um `LOOP` ou `IF` continua a existir após o bloco terminar. O escopo no ABAP é o método inteiro. Mantenha a disciplina para evitar bugs com valores residuais!

# Revolução nº 2: `CONCATENATE` Ficou no Passado

## ANTES

### Clássico Inter SemiBold

```
DATA: lv_nome      TYPE string VALUE 'Ana',  
      lv_sobrenome  TYPE string VALUE 'Silva',  
      lv_completo   TYPE string.
```

```
CONCATENATE 'Prezado cliente:' lv_nome  
             lv_sobrenome  
INTO lv_completo SEPARATED BY space.
```

## DEPOIS

### Moderno Inter SemiBold

```
DATA(lv_nome) = 'Ana'.  
DATA(lv_sobrenome) = 'Silva'.  
  
DATA(lv_completo) =  
|Prezado cliente: { lv_nome } { lv_sobrenome }|.
```

Intuitivo e poderoso. Espaços são respeitados literalmente, e qualquer expressão ABAP válida pode ser colocada entre chaves {...}.

# O Poder Oculto dos Templates: Formatação e Lógica Embutida



## Datas e Moedas Inter SemiBold

```
|Data para o usuário: {  
{ sy-datum DATE = USER }|
```

```
|Salário formatado: {  
{ lv_salary CURRENCY = 'BRL'  
NUMBER = USER }|
```

Adapta-se automaticamente ao perfil do usuário logado, simplificando a internacionalização.



## Conversão ALPHA Inter SemiBold

```
|Material para DB: {  
{ lv_matnr ALPHA = IN }|
```

```
|Material para Tela: {  
{ lv_db_format ALPHA = OUT }|
```

Essencial para chaves de banco de dados, sem chamar functions.



## Lógica Condisional Inter SemiBold

```
|Status: {  
COND #(  
WHEN lv_nota > 7  
THEN 'Aprovado'  
ELSE 'Reprovado' )  
} }|
```

Execute decisões lógicas diretamente onde o resultado será usado.

# Revolução nº 3: Construa Resultados, Não Apenas Execute Lógica

***"Transforme blocos de código procedurais em expressões funcionais elegantes."***



# COND e SWITCH: A Evolução Inteligente do IF/CASE

## Antes - IF

```
IF lv_idade < 12.  
    lv_fase = 'Criança'.  
ELSEIF lv_idade >= 12 AND lv_idade < 18.  
    lv_fase = 'Adolescente'.  
ELSE.  
    lv_fase = 'Adulto'.  
ENDIF.
```

## Depois - COND

```
DATA(lv_fase) = COND string(  
    WHEN lv_idade < 12 THEN 'Criança'  
    WHEN lv_idade < 18 THEN 'Adolescente'  
    ELSE 'Adulto' ).
```

## Antes - CASE

```
CASE lv_status.  
    WHEN 'S'.  
        lv_cor_semaforo = 'Verde'.  
    WHEN 'E'.  
        lv_cor_semaforo = 'Vermelho'.  
    WHEN OTHERS.  
        lv_cor_semaforo = 'Amarelo'.  
ENDCASE.
```

## Depois - SWITCH

```
DATA(lv_cor_semaforo) = SWITCH string( lv_status  
    WHEN 'S' THEN 'Verde'  
    WHEN 'E' THEN 'Vermelho'  
    ELSE 'Amarelo' ).
```

A intenção fica mais clara. O código descreve o **\*resultado desejado\***, não os passos para alcançá-lo.

# Ferramentas Avançadas: Variáveis Temporárias e Construção Atômica

## ‘LET’ - Evite Poluir seu Código

### Problema

Precisa de uma variável auxiliar apenas para um cálculo intermediário dentro de uma condição?

```
DATA(lv_discount) = COND i(
    LET media = ( lv_compra1 + lv_compra2 ) / 2 IN
    WHEN media > 1000 THEN 20
    ELSE 10 ).
```

### Benefício

A variável `media` só existe dentro desta operação, mantendo o escopo do método limpo.

## ‘VALUE’ - Construção Instantânea

### Problema

Preenchendo uma estrutura ou tabela interna, campo a campo, linha a linha?

```
" Para estruturas
DATA(ls_user) = VALUE ty_user( id = 1 name = 'João' ).

" Para tabelas (adicionando à existente)
lt_history = VALUE #( BASE lt_history ( ... ) ).
```

### Benefício

Cria e preenche dados complexos de forma atômica, legível e segura.

# Síntese: A Calculadora Robusta em Ação

```
METHOD if_oo_adt_classrun~main.  
    " 1. Setup com Declarações Inline  
    DATA(lv_num1) = 10. ←  
    DATA(lv_num2) = 5.  
    DATA(lv_operation) = 'DIV'.  
    DATA lt_history TYPE tt_log.  
  
    " 2. Lógica principal com SWITCH  
    DATA(lv_result) = SWITCH decfloat34( lv_operation ←  
        WHEN 'DIV' THEN  
            " 3. COND Aninhado para segurança  
            COND #( WHEN lv_num2 <> 0  
                  THEN lv_num1 / lv_num2 ←  
                  ELSE 0 )  
        WHEN 'SUM' THEN lv_num1 + lv_num2  
        ELSE -1 ).  
  
    " 4. Saída com String Template e formatação  
    out->write( |Resultado: { lv_result NUMBER = USER }| ). ←  
  
    " 5. Adição ao histórico com VALUE  
    lt_history = VALUE #( BASE lt_history  
        ( operation = lv_operation  
          val1      = lv_num1 ←  
          val2      = lv_num2  
          result    = lv_result ) ).  
  
ENDMETHOD.
```

## 1. Declarações Inline:

Variáveis declaradas no momento do uso, com tipo inferido.

## 2. `SWITCH` para Lógica Principal:

Seleciona a operação de forma clara e funcional.

## 3. `COND` Aninhado para Segurança:

Evita a divisão por zero de forma elegante, sem um `IF` separado.

## 4. `String Template` com Formatação:

Formata o número de saída para o usuário final diretamente na string.

## 5. `VALUE` para Histórico:

Adiciona uma nova linha à tabela de forma atômica e legível, sem `APPEND`.

# Seu Novo "Cheat Sheet" ABAP

**Declaração:**

```
DATA: lv_val TYPE i.
```



```
DATA(lv_val) = 10.
```

**Concatenação:**

```
CONCATENATE a b INTO c.
```



```
c = |{ a } { b }|
```

**Atribuição Lógica:**

```
IF/ELSE ... ENDIF.
```



```
DATA(x) = COND #( ... ).
```

**Construção de Estrutura:**

```
ls_data-campo = val.
```



```
ls_data = VALUE #( ... ).
```

**Verificar Tabela:**

```
READ TABLE ...; IF sy-subrc = 0.
```



```
IF line_exists( ... ).
```

**Conversão Alpha:**

```
CALL FUNCTION...
```



```
|{ var ALPHA = IN }|
```

Transforme estes conceitos em hábitos diários.

# Teste seu Entendimento: As Regras do Jogo

**O operador `COND` pode substituir qualquer comando `IF`?**

**Resposta:** Não. `COND` é um operador para *retornar um valor*. Ele substitui a lógica `IF` usada para atribuição. Para controle de fluxo (chamar métodos, sair de loops), o comando `IF` tradicional ainda é necessário.

---

**Qual o risco de uma variável declarada com `DATA(...)` dentro de um `LOOP`?**

**Resposta:** A variável continua acessível após o `DATA(...)` dentro de um loop.

**Resposta:** A variável continua acessível após o `ENDLOOP`. Seu escopo é o método inteiro, não o loop. Se não for limpa (`CLEAR`), ela pode carregar um valor residual para iterações ou lógicas futuras, causando bugs.

# Sua Jornada Apenas Começou: Próximos Passos



**Pratique:** Comece aplicando as novas sintaxes em pequenos relatórios ou classes de utilidade. Crie um ambiente seguro para experimentar.



**Refatore:** Escolha um programa antigo e aplique um conceito por vez. String Templates (`|...|`) e `line_exists()` são ótimos pontos de partida com baixo risco e alto impacto na legibilidade.



**Compartilhe:** Discuta estas técnicas com sua equipe. Promova um código mais limpo e padronizado nos *code reviews*.



**Pense Moderno:** Antes de escrever um bloco `IF` para atribuir um valor, pergunte-se: 'Posso resolver isso com `COND`?' Crie o hábito.

# Código Limpo é Código Profissional.

A modernização do ABAP não é apenas sobre sintaxe, é sobre clareza, eficiência e a qualidade do software que construímos.

Repositório do Exemplo: [[github.com/exemplo/abap-moderno](https://github.com/exemplo/abap-moderno)]  
Recursos Adicionais: [[blog.empresacomabap.com.br](http://blog.empresacomabap.com.br)]