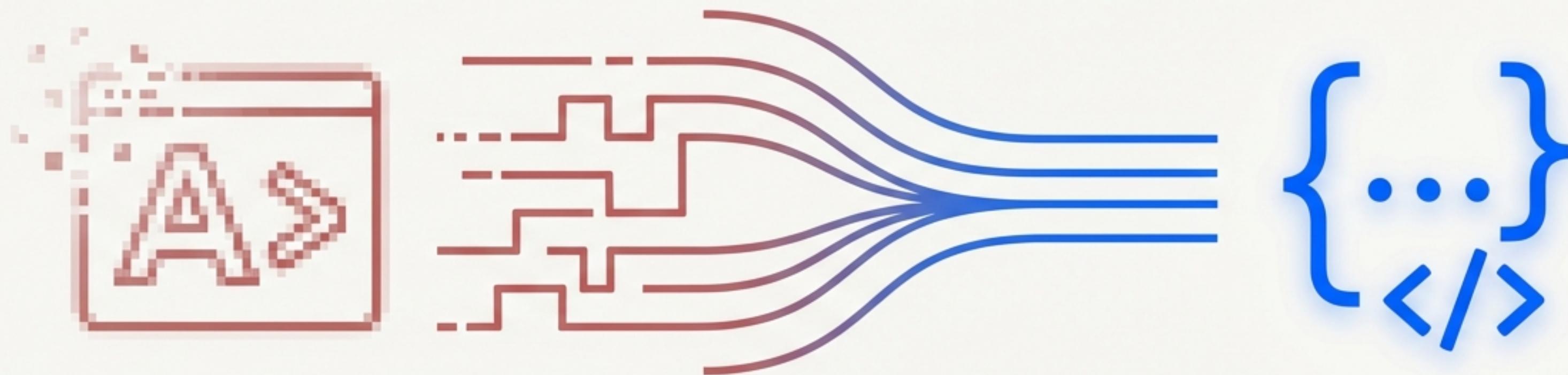


A Evolução do Processamento de Strings em ABAP

De Comandos Obsoletos a Expressões Fluentes e Seguras



Um guia prático para modernizar seu código, aumentar a legibilidade e eliminar erros em tempo de execução.

O Mundo Antigo: O Código que Nos ACOSTUMAMOS a Escrever

Comandos clássicos como `TRANSLATE` e `SEARCH` eram a base da manipulação de strings. Eles funcionavam, mas com um custo em legibilidade e segurança.

Código Verboso e Destruutivo

```
DATA lv_text TYPE string VALUE ' some text '.

" 1. Converter para maiúsculas
TRANSLATE lv_text TO UPPER CASE.

" 2. Buscar por uma substring
SEARCH lv_text FOR 'TEXT'.

IF sy-subrc = 0.
  " ... lógica complexa aqui ...
ENDIF.

" 3. Quebrar a string
SPLIT lv_text AT space INTO ...
```

Principais Desafios



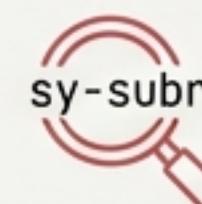
Modificação Destruitiva: Comandos alteram a variável original, tornando o rastreamento do estado difícil.



Impossibilidade de Encadeamento: Cada operação requer uma nova linha de código.



Risco de Dumps: Sintaxes como `lv_text+0(5)` podem causar erros fatais se o offset for inválido.



Dependência do `sy-subrc`: Lógica de controle verbose e propensa a erros.

A Nova Abordagem: Funções, Expressões e Encadeamento

A partir do ABAP 7.40, a manipulação de strings foi reinventada. O novo paradigma é baseado em funções que retornam resultados, permitindo um código mais limpo, seguro e declarativo.



Programação Funcional

Funções como `to_upper()` e `substring()` retornam um novo valor, preservando o original. Isso permite o encadeamento ('chaining'), onde o resultado de uma função é a entrada da próxima.



Segurança Integrada

Funções modernas são mais robustas, tratando casos extremos (como offsets inválidos) de forma graciosa, sem causar dumps.



Legibilidade Aprimorada

O código se torna mais próximo da linguagem natural, descrevendo 'o que' você quer, e não 'como' fazê-lo passo a passo.

Bloco 1: Funções Embutidas — As Substitutas Diretas

⚖ Maiúsculas / Minúsculas

📁 ANTIGO

```
TRANSLATE lv_text TO UPPER CASE.
```

Nota: Modifica `lv_text` diretamente.

⚙ MODERNO

```
lv_text = to_upper( lv_text ).
```

Nota: Retorna uma nova string. Variações: `to_lower()`, `to_mixed()`.

↳ Substrings

📁 ANTIGO

```
lv_part = lv_text+0(5).
```

Nota: Rígido e arriscado. Causa dump se o offset for inválido.

⚙ MODERNO

```
lv_part = substring( val = lv_text off = 0 len = 5 ).
```

Nota: Seguro, legível e permite offsets dinâmicos.

⌫ Limpeza de Espaços

📁 ANTIGO

```
CONDENSE lv_text NO-GAPS.
```

*Nota: Remove *todos* os espaços.*

⚙ MODERNO

```
lv_text = condense( val = lv_text ).
```

*Nota: Comportamento de *Trim* + Redução de espaços internos.*

Um Detalhe Crucial: Entendendo o Novo `condense()`

A função `condense()` é mais inteligente que seu antecessor. Seu comportamento padrão é diferente do antigo `NO-GAPS`, e entender essa diferença é vital para evitar bugs.

`CONDENSE ... NO-GAPS` (O Legado)

```
DATA lv_string TYPE string VALUE ' joão da silva '.
CONDENSE lv_string NO-GAPS.
" lv_string agora é 'joãodasilva'
```

Explicação: Remove **todos** os espaços, unindo as palavras.

`condense()` (A Função Moderna)

```
DATA(lv_raw) = ' joão da silva '.
DATA(lv_condensed) = condense( lv_raw ).
" lv_condensed é 'joão da silva'
```

- 1. Remove espaços no início e no fim (Trim).
- 2. Reduz múltiplos espaços internos a um único espaço.

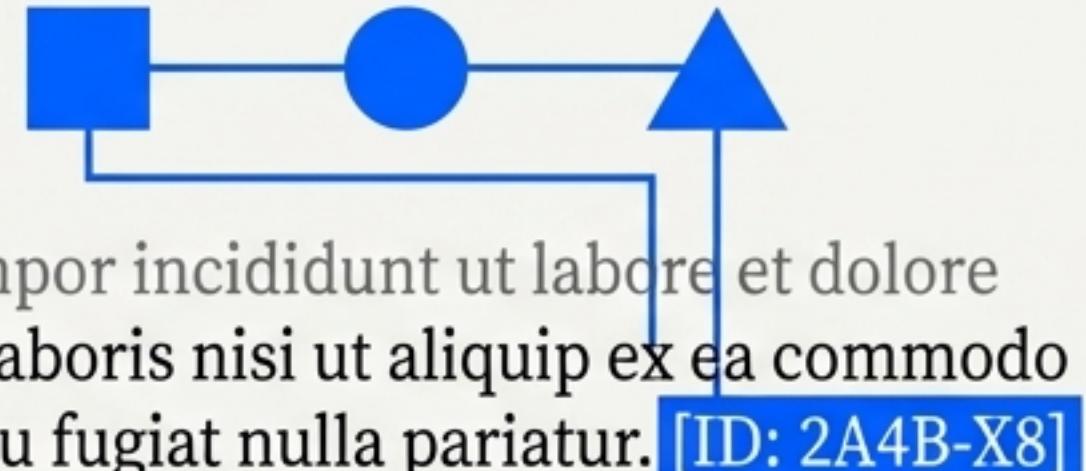
Takeaway: Como replicar o `NO-GAPS`?

Para remover todos os espaços, use a função replace():

```
DATA(lv_no_gaps) = replace( val = lv_raw sub = `` with = `` occ = 0 ).
```

Bloco 2: O Poder das Expressões Regulares (Regex)

E quando precisamos validar um padrão complexo, como um e-mail ou CPF, em vez de um texto fixo? Para isso, o ABAP moderno oferece um robusto motor de Expressões Regulares (Regex).



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse us*e* eu fugiat nulla pariatur. [ID: 2A4B-X8] Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

O que é Regex?

"Regex é uma linguagem formal para descrever padrões de texto."

****Analogia**:** Pense nisso como um 'localizar e substituir' superpoderoso, onde você define as **regras** do que procurar, não apenas o texto exato.

Casos de Uso Comuns em ABAP

- Validação:** Garantir que um campo (e-mail, telefone, CPF) segue um formato obrigatório.
- Extração:** Capturar partes específicas de uma string, como o número de uma nota fiscal dentro de um texto longo.
- Limpeza (Higienização):** Remover todos os caracteres que não correspondem a um padrão (ex: manter apenas números).

Regex em Ação: Validando Formatos Complexos com `matches`

Precisamos verificar se um e-mail inserido pelo usuário é sintaticamente válido antes de salvá-lo.

JOAO.SILVA@SAP.COM



Válido

```
DATA(lv_email) = 'JOAO.SILVA@SAP.COM'.
```

```
IF matches( val      = lv_email  
            regex = '^@[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$' ).  
    out->write( 'Formato de e-mail válido!' ).
```

```
ELSE.
```

```
    out->write( 'Formato inválido.' ).
```

```
ENDIF.
```

**[a-z]{2,}: O TLD (ex: .com, .br).

**[a-z]{...}: O TLD (ex: /. domínio).

^@[a-zA-Z0-9._%+-]+: O nome do usuário.

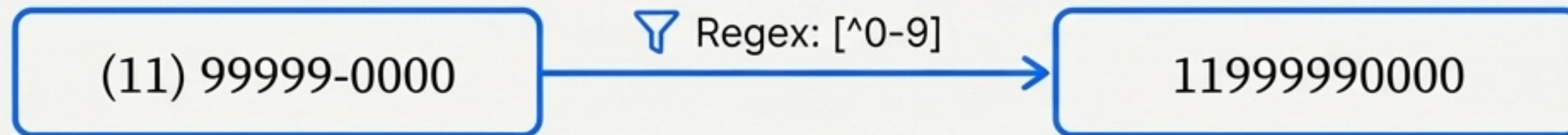
**\.: O ponto do domínio.

@: O separador literal.

Com uma única linha, realizamos uma validação que seria extremamente complexa e frágil com comandos legados como `SEARCH` ou `CP`.

Regex em Ação: Higienizando Dados com `replace`

Recebemos um telefone de um sistema externo com formatação `(11) 99999-0000`. Precisamos salvar apenas os dígitos `11999990000` no banco de dados.



```
DATA(lv_phone) = '(11) 99999-0000'.  
" Regex '[^0-9]' encontra qualquer caractere que NÃO (^) seja um dígito (0-9).  
" A função substitui os caracteres encontrados por uma string vazia ('').  
DATA(lv_clean) = replace(val = lv_phone  
                      regex = '[^0-9]'  
                      with = ''  
                      occ = 0 ). " occ = 0 para todas as ocorrências  
" Resultado: lv_clean = '11999990000'
```

Padrão-chave

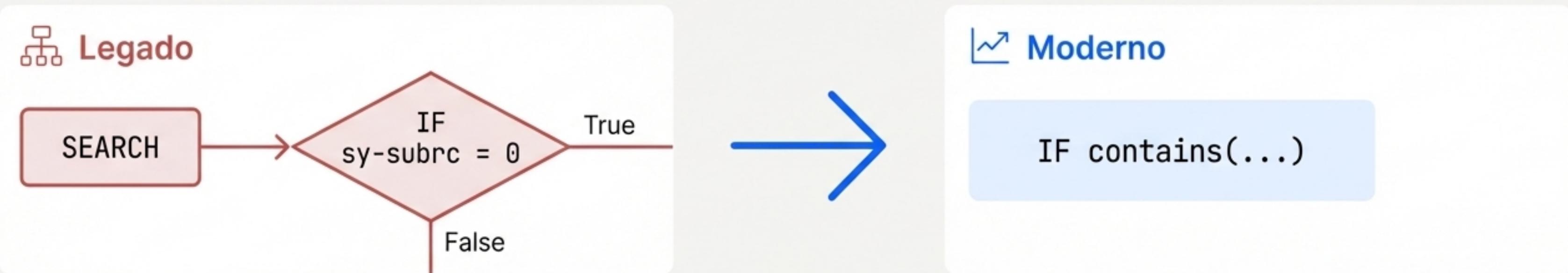
[^...]: O acento circunflexo ^ dentro de colchetes [] atua como uma **negação**. [^0-9] significa 'Encontre qualquer caractere que não esteja no intervalo de 0 a 9'. Uma técnica poderosa para limpeza de dados.

Bloco 3: Funções Predicativas — Escrevendo Condições Mais Inteligentes

As funções predicativas são projetadas para um propósito: retornar um valor booleano (verdadeiro/falso) para serem usadas diretamente em expressões lógicas ('IF', 'CHECK', 'COND', etc.).

O Fim do `sy-subrc`

Elas eliminam o padrão antigo de "Executar comando -> Verificar `sy-subrc` -> Tomar decisão". A verificação e a decisão acontecem no mesmo lugar.



As Funções Essenciais

`contains(val = ... sub = ...)` Verifica se uma string **contém** uma substring. Substitui 'SEARCH'.

`starts_with(...) / ends_with(...)` Verifica se a string **começa** ou **termina** com um padrão.

`matches(val = ... regex = ...)` Verifica se a string corresponde a uma expressão regular. Substitui 'CP'.

Exemplo Prático: Higienização Completa de Dados de Cadastro

Uma classe utilitária recebe dados 'sujos' de uma fonte externa e os padroniza para o formato SAP.

```
" Dados de Entrada (Sujos)
DATA(lv_raw_name) = ' joão da silva '.
DATA(lv_raw_email) = 'JOAO.SILVA@SAP.COM'.
DATA(lv_raw_sku) = 'MAT-1234-BR'.

" 1. Padronização de Nome (Encadeamento de Funções) ①
DATA(lv_name) = to_upper( condense( val = lv_raw_name ) ).

" 2. Padronização de Email
DATA(lv_email) = to_lower( lv_raw_email ).

" 3. Extração de Informação com Regex ②
" Padrão \d+ busca uma sequência de um ou mais dígitos
find( val = lv_raw_sku regex = '\d+' sub = DATA(lv_extracted_num) ).

" 4. Validação Lógica com Predicados ③
DATA(lv_type) = COND string(
    WHEN starts_with( val = lv_raw_sku sub = 'MAT' ) THEN 'Material Físico'
    WHEN starts_with( val = lv_raw_sku sub = 'SRV' ) THEN 'Serviço'
    ELSE 'Desconhecido'
).
```

① Encadeamento em ação: `condense()` é executado primeiro, seu resultado alimenta `to_upper()`.

② Regex `\d+` extrai apenas a parte numérica do SKU.

③ `starts_with()` torna a lógica do `COND` limpa e legível, sem variáveis auxiliares.

O Resultado: Dados Transformados, Limpos e Estruturados

A aplicação combinada das técnicas modernas resulta em dados padronizados, prontos para serem utilizados de forma consistente no sistema.

Antes Inter SemiBold

```
lv_raw_name: ' joão da silva '
```

```
lv_raw_email : 'JOAO.SILVA@SAP.COM'
```

```
lv_raw_sku: 'MAT-1234-BR'
```

Depois Inter SemiBold

```
Nome Limpo: 'JOÃO DA SILVA'
```

```
Email Limpo: 'joao.silva@sap.com'
```

```
Número do Material: '1234'
```

```
Tipo de Produto: 'Material Físico'
```

Código mais curto, mais expressivo e menos propenso a erros.

Referência Rápida: Tabela Comparativa Antigo vs. Moderno

Use esta tabela como um guia para substituir comandos legados em seu código.

Operação	Comando Legado (Evitar)	Função Moderna (Usar)
Maiúsculas	TRANSLATE x TO UPPER CASE	x = <code>to_upper(x)</code>
Tamanho	DESCRIBE FIELD... ou STRLEN	<code>strlen(x)</code>
Busca	SEARCH x FOR 'ABC'	<code>find(val = x sub = 'ABC')</code>
Confere Padrão	CP (Contains Pattern)	<code>matches(val = x regex = ...)</code>
Concatenar	CONCATENATE a b INTO c	c = \ \{ a }\{ b }\ \ (String Templates)
Substituir	REPLACE 'A' WITH 'B' INTO x	x = <code>replace(val = x sub = 'A' with = 'B')</code>

Glossário Técnico: A Linguagem do ABAP Moderno

Built-in Functions (Funções Embutidas)

Funções nativas da linguagem ABAP (como `strlen`, `to_upper`) que podem ser usadas em qualquer posição de operando, substituindo comandos procedurais antigos.

Regex (Regular Expressions)

Uma linguagem formal para descrever padrões de texto. Usada para validação complexa e operações de busca/substituição avançadas.

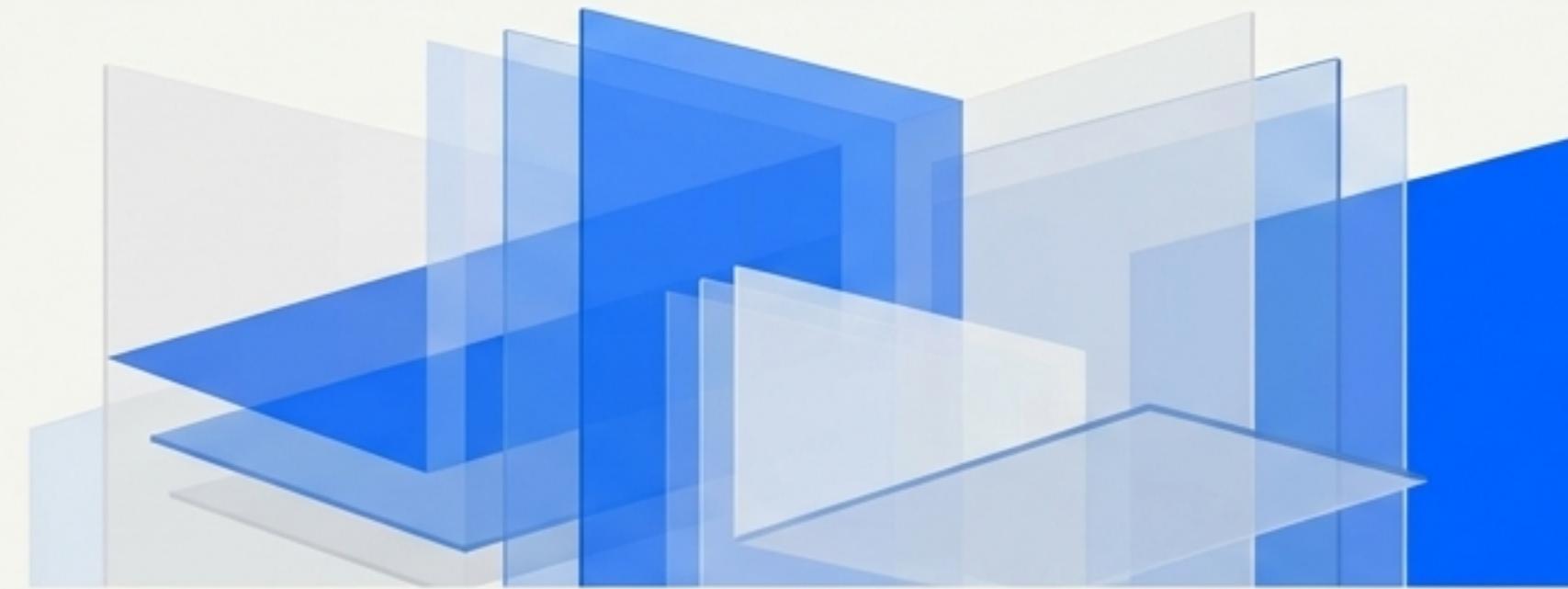
Chaining (Encadeamento)

A capacidade de chamar uma função sobre o resultado de outra, como `func1(func2(var))`. Permite escrever transformações complexas em uma única linha legível.

Predicative Function (Função Predicativa)

Uma função que retorna um valor verdade (booleano) para uso direto em expressões lógicas. Exemplos: `contains`, `matches`, `line_exists`.

Adote a Fluência: Escreva Código Que Você se Orgulhará de Manter



Modernizar a manipulação de strings em ABAP não é apenas sobre usar novas funções. É sobre adotar uma mentalidade focada em:

- ✓ **Clareza:** O código deve expressar sua intenção diretamente.
- ✓ **Segurança:** O código deve ser resiliente a dados inesperados.
- ✓ **Eficiência:** O código deve ser conciso e fácil de manter para o próximo desenvolvedor.

O futuro do ABAP é expressivo e seguro. Comece a construir com ele hoje.