

APS LOGCOMP

Orientador: Raul Ikeda

Aluno: Fernanda Pereira



Motivação da APS

- Criar uma linguagem simples, expressiva e autoral
- Evitar a mesmice de copiar sintaxes como C/Java
- Pensar em comandos que facilitam a leitura e a intenção do código
- Tornar decisões e laços mais declarativos

Características da Linguagem

- Tipos explícitos: int, text, bool
- Declaração obrigatória com valor: let x: int := 10;
- Expressões condicionais: decide(condição, valor1, valor2)
- Laços numéricos: loop i in range(a, b) { ... }
- Impressão: show(...)
- Depuração: watch x; mostra valor da variável
- Retorno intermediário: yield valor;
- Concatenação: ++ entre strings ou string + int

Diferente de
um `if`
clássico,
`decide`
funciona
como uma
****expressão****.

```
loop_statement =  
    "loop" identifier "in" "range" "("  
        expression "," expression ")" " {"  
            statement } "};
```

```
if_expression =  
    "decide" "(" condition ","  
        expression_if_true ","  
        expression_if_false ")";  
    ...
```



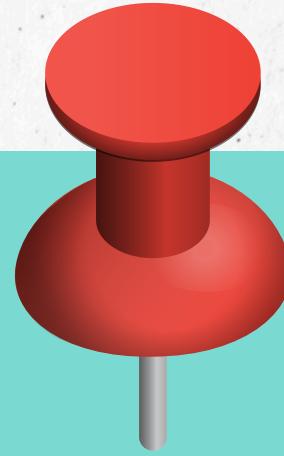
Exemplo de Código

variáveis text e int, operadores
relacionais, decide, watch e yield

```
let nome: text := "Fernanda";
show("Bem-vinda, " ++ nome);

let pontos: int := 15;
watch pontos;

let status: int := decide(pontos >= 10, 1, 0);
yield status;
```



```
Executando programa:
Bem-vinda, Fernanda
[watch] pontos = 15
[yield] 1
```

Exemplo de Código

Teste com loop + yield:
contagem de pontos

```
let pontos: int := 0;  
  
loop i in range(1, 6) {  
    let pontos: int := i * 2;  
    yield pontos;  
}
```

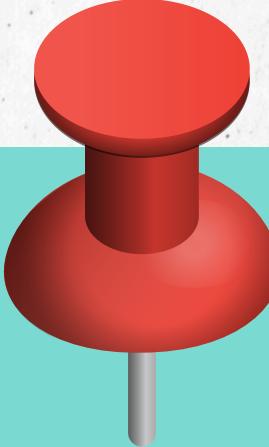


```
[yield] 2  
[yield] 4  
[yield] 6  
[yield] 8  
[yield] 10
```

Exemplo de Código

decide + loop com feedback de status

```
loop i in range(8, 12) {  
    let status: int := decide(i >= 10, 1, 0);  
    show("Tentativa " ++ i);  
    yield status;  
}
```

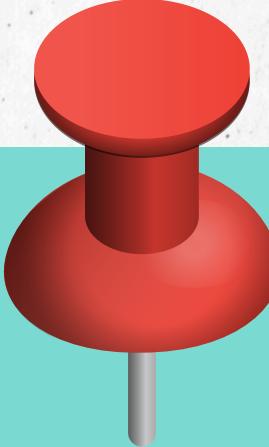


```
Tentativa 8  
[yield] 1  
Tentativa 9  
[yield] 1  
Tentativa 10  
[yield] 1  
Tentativa 11  
[yield] 1
```

Exemplo de Código

Concatenação múltipla com text + int

```
let nome: text := "Fernanda";
    let nivel: int := 3;
show("Jogadora: " ++ nome ++ " | Nível: " ++
    nivel);
```



Jogadora: Fernanda | Nível: 3

Conclusão

- Linguagem 100% autoral e funcional
- Aprendi a estruturar EBNF, criar lexer/parser e interpretar AST
- Flex + Bison + C = muito poder, mas também muita atenção
- Próximos passos: interpretar `bool` e strings reais.