# Documentação da Linguagem Zyntex

## Propósito, Objetivo e Paradigma

Zyntex é uma linguagem de programação multi-paradigma criada com o propósito de oferecer aos estudantes de programação uma forma prática e direta para experimentar e consolidar seu entendimento dos conceitos fundamentais dos paradigmas imperativo (como variáveis, atribuições e controle de fluxo) e funcional (como funções de primeira classe, imutabilidade e funções de ordem superior), permitindo que estudantes escrevam e executem pequenos programas. A combinação desses paradigmas visa permitir que os estudantes comparem diferentes abordagens para a resolução de problemas e apreciem as vantagens de cada estilo.

## Elementos Principais (Tokens)

### Tipos de Dados

A linguagem Zyntex suporta os seguintes tipos de dados primitivos e compostos:

| Tipo de Dado | Sintaxe | Descrição |
|---|---|---|
| Número inteiro | `int` | Representa números inteiros |
| Número decimal | `float` | Representa números de ponto flutuante |
| Caractere | `char` | Representa um único caractere |
| Cadeia de caracteres | `string` | Representa sequências de caracteres |
| Boolean | `bool` | Representa valores verdadeiro/falso |
| Array | `arr` | Representa arrays de elementos |
| Lista | `list` | Representa listas dinâmicas |
| Função | `function` | Representa funções como objetos de primeira classe |
| Nulo | `null` | Representa valor nulo |

| Tipo de Dado | Sintaxe | Descrição |
|---|---|---|
| Vazio (void) | `void` | Representa ausência de valor de retorno |
| Dicionários | `dict` | Representa estruturas chave-valor |

## Palavras-chave

As palavras-chave da linguagem Zyntex são:

| Função | Sintaxe | Descrição |
|---|---|---|
| Definir função | `fx` | Declara uma nova função |
| Definir variável | `vx` | Declara uma nova variável |
| Condicional 'if' | `zf` | Estrutura condicional if |
| Condicional 'else' | `zl` | Estrutura condicional else |
| Laço 'for' | `fr` | Estrutura de repetição for |
| Laço 'while' | `wl` | Estrutura de repetição while |
| Impressão | `out` | Comando de saída/impressão |
| Entrada de dados | `in` | Comando de entrada de dados |
| Retorno de função | `rx` | Comando de retorno |

## Operadores

### Operadores Lógicos

| Função | Sintaxe | Descrição |
|---|---|---|
| AND | `&&` | Operação lógica E |
| OR | `\|\|` | Operação lógica OU |
| NOT | `!` | Operação lógica NÃO |

### Operadores de Atribuição

| Função | Sintaxe | Descrição |
|---|---|---|
| Atribuição de valor | `:=` | Atribui valor a uma variável |
| Atribuição de tipo | `=>` | Atribui tipo a uma variável |

### Operadores Unários

| Função | Sintaxe | Descrição |
|---|---|---|
| Incremento | `+>` | Incrementa valor em 1 |
| Decremento | `->` | Decrementa valor em 1 |

### Operadores Aritméticos

| Função | Sintaxe | Descrição |
|---|---|---|
| Soma | `++` | Operação de adição |
| Subtração | `--` | Operação de subtração |
| Multiplicação | `**` | Operação de multiplicação |
| Divisão | `//` | Operação de divisão |
| Radiciação | `rd` | Operação de radiciação |
| Exponenciação | `exp` | Operação de exponenciação |

### Operadores Relacionais

| Função | Sintaxe | Descrição |
|---|---|---|
| Igual a | `=?` | Comparação de igualdade |
| Diferente de | `!=?` | Comparação de diferença |
| Menor que | `<<` | Comparação menor que |
| Maior que | `>>` | Comparação maior que |
| Menor ou igual | `<=` | Comparação menor ou igual |

| Função | Sintaxe | Descrição |
|---|---|---|
| Maior ou igual | `>=` | Comparação maior ou igual |

### Símbolos

| Função | Sintaxe | Descrição |
|---|---|---|
| Comentário | `##` | Indica início de comentário |
| Bloco de código | `{}` | Delimitadores de bloco |

### Delimitadores Adicionais

A linguagem também utiliza os seguintes delimitadores:
- `(` `)` - Parênteses para agrupamento e parâmetros de função
- `;` - Ponto e vírgula para separação de comandos
- `,` - Vírgula para separação de elementos

## Gramática Formal da Linguagem Zyntex

### Notação BNF (Backus-Naur Form)

```
<programa> ::= <declaracao>*

<declaracao> ::= <declaracao_funcao> | <declaracao_variavel> | <comando>

<declaracao_funcao> ::= "fx" <identificador> "(" <lista_parametros>? ")" "=>"
<tipo> "{" <comando>* "}"

<declaracao_variavel> ::= "vx" <identificador> "=>" <tipo> (":=" <expressao>)?

<lista_parametros> ::= <parametro> ("," <parametro>)*

<parametro> ::= <identificador> "=>" <tipo>

<comando> ::= <atribuicao> | <condicional> | <laco> | <chamada_funcao> |
<retorno> | <entrada> | <saida>

<atribuicao> ::= <identificador> ":=" <expressao>

<condicional> ::= "zf" "(" <expressao> ")" "{" <comando>* "}" ("zl" "{" <comando>*
"}")?

<laco> ::= <laco_for> | <laco_while>
```

<laco_for> ::= "fr" "(" <atribuicao> ";" <expressao> ";" <expressao> ")" "{" <comando>* "}"

<laco_while> ::= "wl" "(" <expressao> ")" "{" <comando>* "}"

<retorno> ::= "rx" <expressao>?

<entrada> ::= "in" <identificador>

<saida> ::= "out" <expressao>

<expressao> ::= <expressao_logica>

<expressao_logica> ::= <expressao_relacional> (("&&" | "||") <expressao_relacional>)*

<expressao_relacional> ::= <expressao_aritmetica> (("=?" | "!=?" | "<<" | ">>" | "<=" | ">=") <expressao_aritmetica>)*

<expressao_aritmetica> ::= <termo> (("++" | "--") <termo>)*

<termo> ::= <fator> (("**" | "//" | "rd" | "exp") <fator>)*

<fator> ::= <numero> | <identificador> | <chamada_funcao> | "(" <expressao> ")" | <expressao_unaria>

<expressao_unaria> ::= ("+>" | "->" | "!") <fator>

<chamada_funcao> ::= <identificador> "(" <lista_argumentos>? ")"

<lista_argumentos> ::= <expressao> ("," <expressao>)*

<tipo> ::= "int" | "float" | "char" | "string" | "bool" | "arr" | "list" | "function" | "null" | "void" | "dict"

<identificador> ::= <letra> (<letra> | <digito> | "_")*

<numero> ::= <digito>+ ("." <digito>+)?

<letra> ::= "a" | "b" | ... | "z" | "A" | "B" | ... | "Z"

<digito> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

# Exemplos de Código Zyntex

## Exemplo 1: Programa Simples com Variáveis e Operações

```
 ## Programa simples de cálculo
vx numero1 => int := 10;
vx numero2 => int := 5;
vx resultado => int;

resultado := numero1 ++ numero2;
out resultado;

resultado := numero1 -- numero2;
out resultado;

resultado := numero1 ** numero2;
out resultado;
```

## Exemplo 2: Função com Parâmetros

```
 ## Função para calcular área de retângulo
fx calcular_area(largura => float, altura => float) => float {
    vx area => float;
    area := largura ** altura;
    rx area;
}

vx l => float := 5.0;
vx h => float := 3.0;
vx resultado => float;

resultado := calcular_area(l, h);
out resultado;
```

## Exemplo 3: Estruturas Condicionais

```
 ## Programa com condicionais
vx idade => int;
out "Digite sua idade: ";
in idade;

zf (idade >= 18) {
    out "Você é maior de idade";
} zl {
```

```
    out "Você é menor de idade";
}
```

### Exemplo 4: Estruturas de Repetição

```
 ## Programa com laços
vx contador => int;

## Laço for
fr (contador := 1; contador <= 10; contador +>) {
    out contador;
}

## Laço while
vx numero => int := 1;
wl (numero <= 5) {
    out numero;
    numero +>;
}
```

### Exemplo 5: Programa Complexo com Função Recursiva

```
 ## Cálculo de fatorial usando recursão
fx fatorial(n => int) => int {
    zf (n <= 1) {
        rx 1;
    } zl {
        rx n ** fatorial(n -> 1);
    }
}

vx num => int;
out "Digite um número: ";
in num;

vx resultado => int := fatorial(num);
out "Fatorial de ";
out num;
out " é ";
out resultado;
```

# Características Especiais da Linguagem

## Paradigma Imperativo

- Variáveis mutáveis declaradas com `vx`
- Atribuições com `:=`
- Estruturas de controle tradicionais ( `zf` , `zl` , `fr` , `wl` )
- Comandos sequenciais

## Paradigma Funcional

- Funções como objetos de primeira classe
- Suporte a recursão
- Funções podem ser passadas como parâmetros
- Tipo `function` para representar funções

## Sistema de Tipos

- Tipagem estática com declaração explícita
- Operador `=>` para especificação de tipos
- Tipos primitivos e compostos
- Suporte a arrays, listas e dicionários

## Comentários

- Comentários de linha única iniciados com `##`
- Comentários são ignorados pelo analisador léxico