CorpLang - Documentação e Gramática

Visão Geral

CorpLang é uma linguagem de programação projetada especificamente para ambientes corporativos, com foco em análise de dados, machine learning e automação de processos empresariais.

Características Principais

- **Sintaxe Limpa**: Sintaxe similar a JavaScript/C, fácil de aprender
- **Orientada a Dados**: Operações nativas para manipulação de datasets
- **IA Integrada**: Comandos built-in para machine learning
- **Tipagem Dinâmica**: Flexibilidade para análise exploratória
- **Extensível**: Arquitetura modular para expansão

Gramática Formal (EBNF)

```ebnf
```

```
program = { statement } ;

statement = var_declaration
          | function_declaration
          | assignment
          | if_statement
          | while_statement
          | return_statement
          | dataset_operation
          | model_operation
          | expression_statement ;

var_declaration = "var" identifier "=" expression ";" ;

function_declaration = "function" identifier "(" parameter_list ")" "{" { statement } "}" ;

parameter_list = [ identifier { "," identifier } ] ;

assignment = identifier "=" expression ";" ;

if_statement = "if" "(" expression ")" "{" { statement } "}"
               [ "else" "{" { statement } "}" ] ;

while_statement = "while" "(" expression ")" "{" { statement } "}" ;

return_statement = "return" [ expression ] ";" ;

dataset_operation = "dataset" identifier identifier [ "(" argument_list ")" ] ";" ;

model_operation = "model" identifier identifier [ "(" argument_list ")" ] ";" ;

expression_statement = expression ";" ;

expression = or_expression ;

or_expression = and_expression { "or" and_expression } ;

and_expression = equality_expression { "and" equality_expression } ;

equality_expression = comparison_expression { ( "==" | "!=" ) comparison_expression } ;

comparison_expression = additive_expression { ( "<" | ">" | "<=" | ">=" ) additive_expression } ;

additive_expression = multiplicative_expression { ( "+" | "-" ) multiplicative_expression } ;

multiplicative_expression = unary_expression { ( "*" | "/" | "%" ) unary_expression } ;
```

```
unary_expression = ( "not" | "-" ) unary_expression | primary_expression ;

primary_expression = number
                   | string
                   | boolean
                   | identifier [ "(" argument_list ")" ]
                   | "(" expression ")" ;

argument_list = [ expression { "," expression } ] ;

identifier = letter { letter | digit | "_" } ;

number = digit { digit } [ "." digit { digit } ] ;

string = "\"" { character } "\"" | "'" { character } "'" ;

boolean = "true" | "false" ;

letter = "a" | "b" | ... | "z" | "A" | "B" | ... | "Z" ;

digit = "0" | "1" | ... | "9" ;

character = any_character_except_quote ;
```

Palavras-Chave

Controle de Fluxo

- `if`, `else` - Estruturas condicionais
- `while` - Loops
- `for`, `in` - Iteração (futuro)
- `return` - Retorno de função

Declarações

- `var` - Declaração de variável
- `function` - Declaração de função

Operadores Lógicos

- `and` - E lógico
- `or` - Ou lógico
- `not` - Negação lógica

Valores Booleanos

- `true` - Verdadeiro
- `false` - Falso

Específicos para Dados

- `dataset` - Operações com conjuntos de dados
- `model` - Operações com modelos de ML
- `predict` - Predições
- `train` - Treinamento
- `analyze` - Análise de dados

Tipos de Dados

Primitivos

- **Números**: `123`, `45.67`, `-89`
- **Strings**: `"texto"`, `'texto'`
- **Booleanos**: `true`, `false`
- **Null**: `null` (futuro)

Compostos (futuro)

- **Arrays**: `[1, 2, 3]`
- **Objects**: `{nome: "João", idade: 30}`
- **Datasets**: Estruturas especializadas para dados tabulares

Operações com Datasets

Carregamento

```
corplang

dataset load vendas("vendas.csv");
dataset load clientes("clientes.json");
```

Análise

```
corplang

dataset analyze vendas;
dataset analyze clientes;
```

Filtros

corplang

```
dataset filter vendas;
dataset filter clientes;
```

Salvamento

corplang

```
dataset save vendas;
dataset save clientes;
```

Operações com Modelos

Criação

corplang

```
model create previsao_vendas(type="linear_regression");
model create segmentacao_clientes(type="clustering");
```

Treinamento

corplang

```
model train previsao_vendas(dataset="vendas");
model train segmentacao_clientes(dataset="clientes");
```

Predição

corplang

```
model predict previsao_vendas(input="novo_produto");
model predict segmentacao_clientes(input="novo_cliente");
```

Exemplos Práticos

1. Análise de Vendas

corplang

```
# Carregamento de dados
dataset load vendas("vendas_2024.csv");
dataset analyze vendas;

# Função para calcular crescimento
function calcular_crescimento(vendas_atual, vendas_anterior) {
    return ((vendas_atual - vendas_anterior) / vendas_anterior) * 100;
}

# Análise de crescimento
var crescimento = calcular_crescimento(150000, 120000);
print("Crescimento de vendas:", crescimento, "%");

# Modelo de previsão
model create previsao_vendas(type="linear_regression");
model train previsao_vendas(dataset="vendas");
```

2. Segmentação de Clientes

corplang

```
# Carregamento de dados de clientes
dataset load clientes("base_clientes.csv");
dataset analyze clientes;

# Modelo de segmentação
model create segmentacao(type="kmeans");
model train segmentacao(dataset="clientes");

# Classificação de novos clientes
var classificacao = 0.85;
if (classificacao > 0.8) {
    print("Cliente premium");
} else {
    if (classificacao > 0.6) {
        print("Cliente padrão");
    } else {
        print("Cliente básico");
    }
}
```

3. Análise de RH

corplang

```
# Sistema de análise de recursos humanos
dataset load funcionarios("funcionarios.csv");
dataset analyze funcionarios;

function avaliar_promocao(desempenho, tempo_empresa) {
    var score = (desempenho * 0.7) + (tempo_empresa * 0.3);
    return score;
}

# Modelo de predição de turnover
model create turnover_model(type="classification");
model train turnover_model(dataset="funcionarios");

var score_promocao = avaliar_promocao(8.5, 3.2);
print("Score de promoção:", score_promocao);
```

Funções Built-in

Funções de Saída

- `print(...)` - Imprime valores no console

Funções de Utilidade

- `len(objeto)` - Retorna o tamanho de strings ou arrays
- `type(objeto)` - Retorna o tipo do objeto

Roadmap de Desenvolvimento

Versão 1.1

- [ ] Estruturas de dados (arrays, objetos)
- [ ] Loops `for` e `for...in`
- [ ] Tratamento de exceções (`try/catch`)
- [ ] Módulos e imports

Versão 1.2

- [ ] Operações avançadas de dataset (joins, aggregations)
- [ ] Conectores para bancos de dados
- [ ] Visualizações básicas
- [ ] Validação de tipos

Versão 1.3

- [ ] Integração com APIs REST

- [ ] Operações assíncronas
- [ ] Sistema de plugins
- [ ] IDE integrada

Versão 2.0

- [ ] Compilação para LLVM
- [ ] Otimizações de performance
- [ ] Paralelização automática
- [ ] Integração com Kubernetes

Arquitetura Técnica

Componentes Principais

1. **Lexer**: Análise léxica e tokenização
2. **Parser**: Análise sintática e geração de AST
3. **Interpreter**: Execução do código
4. **Environment**: Gerenciamento de escopo
5. **Built-ins**: Funções e operações nativas

Padrões de Design

- **Visitor Pattern**: Para percorrer e executar a AST
- **Strategy Pattern**: Para diferentes tipos de operações
- **Factory Pattern**: Para criação de modelos e datasets
- **Observer Pattern**: Para monitoramento de execução

Extensibilidade

- **Plugin System**: Para adicionar novos tipos de operações
- **Custom Functions**: Para funções específicas do domínio
- **Connectors**: Para integração com sistemas externos
- **Serialization**: Para persistência de estado

Integração Corporativa

Conectores Nativos

- **SQL Databases**: PostgreSQL, MySQL, SQL Server
- **NoSQL**: MongoDB, Elasticsearch
- **Data Lakes**: HDFS, S3
- **APIs**: REST, GraphQL, gRPC

Segurança

- **Authentication**: OAuth2, JWT
- **Authorization**: RBAC (Role-Based Access Control)
- **Encryption**: TLS/SSL para comunicação
- **Audit**: Logs de execução e acesso

Monitoramento

- **Métricas**: Prometheus, Grafana
- **Logs**: ELK Stack
- **Tracing**: Jaeger, Zipkin
- **Alertas**: Configuração automática

Exemplo de Arquivo de Configuração

json

```
{
  "corplang": {
    "version": "1.0",
    "environment": "production",
    "security": {
      "authentication": "oauth2",
      "encryption": true
    },
    "connectors": {
      "database": {
        "type": "postgresql",
        "host": "localhost",
        "port": 5432,
        "database": "corporate_db"
      },
      "ml_backend": {
        "type": "tensorflow",
        "gpu_enabled": true
      }
    },
    "monitoring": {
      "metrics": true,
      "logging": true,
      "tracing": true
    }
  }
}
```

Conclusão

CorpLang representa uma base sólida para desenvolvimento de uma linguagem de programação corporativa. Com sua arquitetura modular e foco em análise de dados, oferece um caminho claro para evolução e integração com o stack tecnológico empresarial moderno.

A implementação atual fornece as funcionalidades essenciais para começar a trabalhar com análise de dados e machine learning, enquanto a arquitetura permite expansão gradual para atender necessidades corporativas mais complexas.