# ParkingScript

UMA LINGUAGEM PARA FACILITAR A ADMINISTRAÇÃO DOS ESTACIONAMENTOS


Luiza Coutinho

# Motivação e Problema

## Controle de Vagas Inteligente

Tarefas de controle de vagas, embora comuns, frequentemente exigem scripts complexos ou monitoramento manual repetitivo. A linguagem ParkingScript nasceu da necessidade de automatizar esse processo usando uma sintaxe direta e intuitiva, focada exclusivamente no domínio de estacionamentos.

**Estacionamentos de condomínios, shoppings e empresas enfrentam desafios comuns:**

- Controle inconsistente das vagas disponíveis
- Risco de superlotação
- Falta de alertas claros para porteiros/manobristas
- Processos repetitivos de entrada/saída
- Lógica operacional difícil de padronizar

**A ParkingScript surge para:**

- Simplificar o controle de vagas com uma linguagem intuitiva, em português
- Reduzir erros humanos na operação
- Facilitar automações em sistemas de cancela
- Criar regras claras e reproduzíveis para operação do estacionamento
- A cancela consegue captar  informações diminuindo checagens manuais e possíveis conflitos e possibilidades de superlotação.

**Exemplo de código na linguagem PS**

```
ps

se vagas == 0 {
   alarme
}
entrada 1
```

# Ideia Central

Uma linguagem simples, dedicada a um único objetivo: controlar vagas e automatizar o funcionamento do estacionamento.

# Características

## Como a ParkingScript funciona?

É uma DSL (Domain-Specific Language) com comandos diretos para controlar vagas. Toda a lógica gira em torno da variável implícita: **vagas**.

### 1. Configuração

vagas N
Define a quantidade inicial de vagas livres.

### 2. Ações

entrada N → diminui vagas
saida N → aumenta vagas
alarme → imprime alerta (PRINT)

### 3. Controle de fluxo

enquanto vagas > 0 { ... }
Executa repetidamente até as vagas acabarem.

### 4. Finalização

parar
Encerra o programa.

## Como funciona internamente?

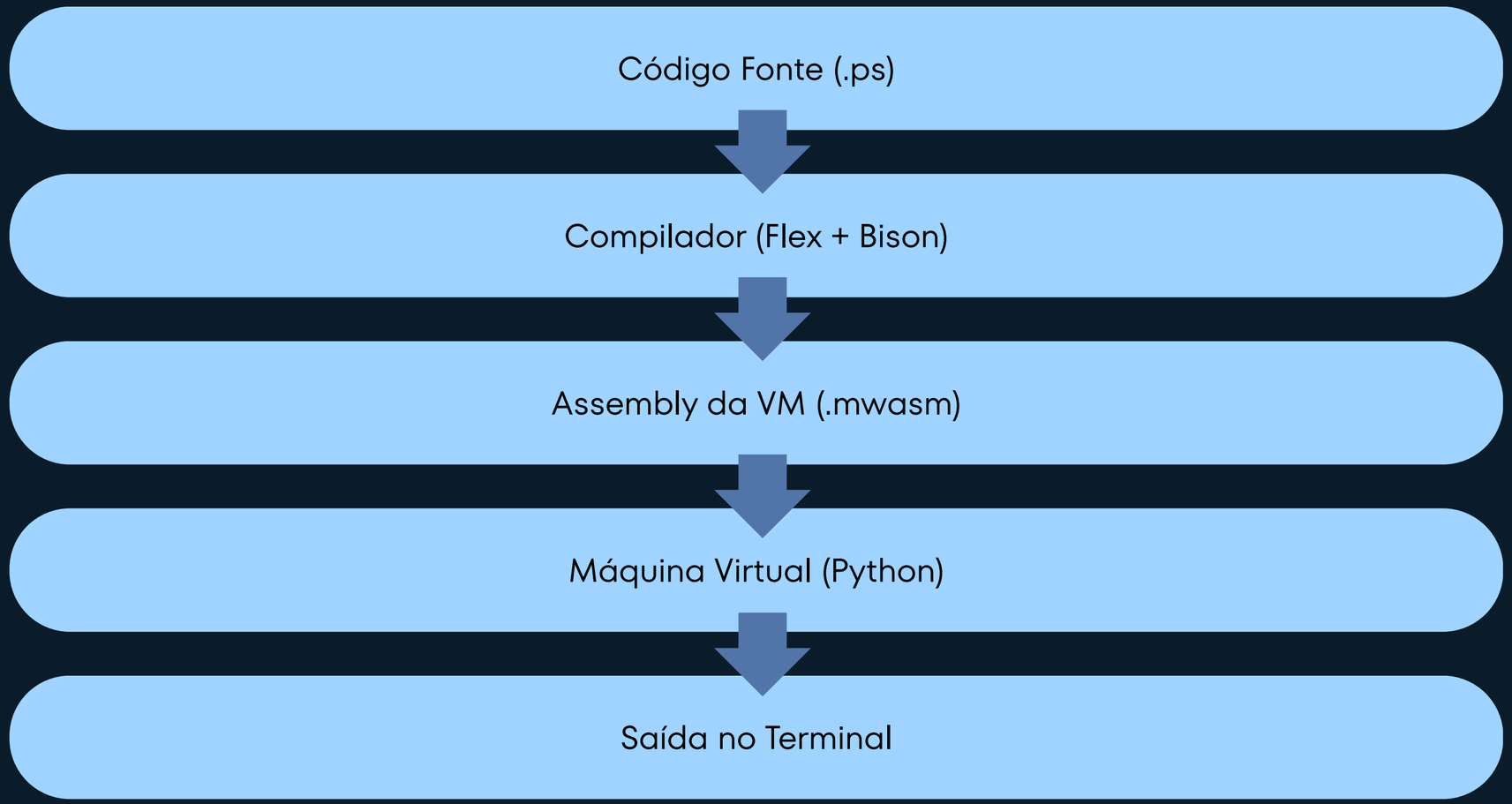
A linguagem é compilada via Flex + Bison para assembly da MicrowaveVM, que possui:

- Registrador TIME → representa as vagas
- Registrador POWER → contador auxiliar
- Instruções: SET, INC, DECJZ, GOTO, PRINT, HALT
- Execução real em VM Python

É literalmente uma linguagem de verdade, compilada e executada em máquina virtual.

# Arquitetura do Compilador à VM

**ParkingScript é um sistema completo:**
**O código-fonte é compilado (Flex/Bison) gerando um assembly customizado que é executado pela MicrowaveVM.**



# Traduções VM x ParkingScript

**Traduções principais:**

| Comando ParkingScript | Descrição | Código MicrowaveVM (Assembly) |
|---|---|---|
| `vagas N` | Inicializa o número de vagas. | `SET TIME N` |
| `entrada N` | Tenta diminuir `TIME` em N (decrementa até 0). | `SET POWER N`<br>`Lentrada:`<br>`DECJZ POWER Lentrada_end`<br>`DECJZ TIME Lentrada`<br>`GOTO Lentrada`<br>`Lentrada_end:` |
| `saida N` | Aumenta `TIME` em N. | `SET POWER N`<br>`Lsaida:`<br>`DECJZ POWER Lsaida_end`<br>`INC TIME`<br>`GOTO Lsaida`<br>`Lsaida_end:` |
| `alarme` | Imprime o valor atual de `TIME`. | `PRINT` |
| `enquanto vagas > 0` | Laço enquanto `TIME` > 0. | `Lloop:`<br>`DECJZ TIME Lend`<br>`INC TIME`<br>`... corpo do while ...`<br>`GOTO Lloop`<br>`Lend:` |
| `parar` | Encerra o programa. | `HALT` |

# Operando um Estacionamento

**Um cenário real: dono de estacionamento de pequeno porte precisa controlar a entrada de carros que entram em seu estacionamento pela cancela.**
**O script abaixo automatiza todo o processo, garantindo alertas e evitando superlotação.**

```
ps

vagas 5

entrada 2      // 2 carros entraram
saida 1        // 1 carro saiu
alarme         // printa situação atual

enquanto vagas > 0 {
   entrada 2   // tenta entrar 2 carros
   alarme      // alerta a cada iteração
}

parar
```

1. **vagas 5 → Estacionamento começa com 5 vagas**
2. **entrada 2 → 3 vagas restantes**
3. **saida 1 → 4 vagas restantes**
4. **alarme → imprime 4**
5. **Loop enquanto vagas > 0**
    a. **1ª iteração: entrada 2 → vagas = 2 → alarme imprime 2**
    b. **2ª iteração: entrada 2 → vagas = 0 → alarme imprime 0**
    c. **3ª iteração: condição falsa → loop termina**
6. **parar → HALT**

# Operando um Estacionamento

**Um cenário real: dono de estacionamento de pequeno porte precisa controlar a entrada de carros que entram em seu estacionamento pela cancela.**
**O script abaixo automatiza todo o processo, garantindo alertas e evitando superlotação.**

```
ps

vagas 5

entrada 2      // 2 carros entraram
saida 1        // 1 carro saiu
alarme         // printa situação atual

enquanto vagas > 0 {
   entrada 2   // tenta entrar 2 carros
   alarme      // alerta a cada iteração
}

parar
```

- **Define vagas iniciais**
- **Simula entrada de veículos**
- **Evita superlotação”**
- **Loop até esgotar vagas**

# Saída real da MicrowaveVM

```yaml
TIME: 4
TIME: 2
TIME: 0
BEEEEEPP!
Final state: {'TIME': 0, 'POWER': 0}
```

**Interpretação da saída:**
**TIME: 4 → após entrada/saída iniciais**
**TIME: 2 e TIME: 0 → dentro do loop**
**BEEEP! → programa finalizou (HALT)**

# Controle Inteligente com Condicionais

```ps

se vagas == 0 {
    alarme
} senao {
    entrada 1
}
```

**ParkingScript permite criar scripts que reagem ao estado do estacionamento, evitando erros e garantindo fluidez.**

1. Identifica vagas cheias
2. Executa ação alternativa automaticamente

# O Poder da Automação com Loops

```
ps

vagas 6

enquanto vagas > 0 {
    entrada 2
    alarme
}
```

**Para estacionamentos com alta rotatividade, a linguagem suporta loops que automatizam entradas e verificações.**

1. Repetição

# Gramática (EBNF)

**A gramática EBNF define as regras principais da ParkingScript, facilitando a implementação de comandos e estruturas, como a simplificação do loop "enquanto".**

```
Programa      = { Declaracao | Comando } ;

Declaracao    = "vagas" Numero ;          (* define a capacidade inicial *)

Comando       = Entrada
              | Saida
              | Alarme
              | Enquanto
              | Parar ;

Entrada       = "entrada" Numero ;        (* simula entrada de N carros *)
Saida         = "saida" Numero ;          (* simula saída de N carros  *)
Alarme        = "alarme" ;                (* emite sinal sonoro/print  *)
Parar         = "parar" ;                 (* encerra execução          *)

Enquanto      = "enquanto" Condicao "{" { Comando } "}" ;

Condicao      = "vagas" Operador Numero ;

Operador      = ">" | "<" | "==" ;

Numero        = Digit { Digit } ;
Digit         = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
```

# Experimente Você Mesmo!

**Requisitos**

Flex, Bison, GCC, Make, Python 3

**Instalação (Ubuntu/Debian)**

```
sudo apt-get install flex bison gcc make python3
```

**Compilação**

```
make
```

**Gerar Assembly a partir do ParkingScript**

```
./parking_parser src/parking_test.ps out/parking_test.mwasm
```

**Executar na VM Microwave**

```
cd MicrowaveVM
python3 main.py ../logcomp_aps/parking-script/out/parking_test.mwasm
```

# ParkingScript