

# Formal (Somewhat) Definition for Pooka Language Syntax

Leslie

May 11, 2024

## 1 Lexical Elements

### 1.1 Identifiers

`<ident>`

An identifier is a string of characters that starts with one of the below:

- A UTF-8 alphanumeric that is not an ASCII numeric
- A UTF-8 emoji
- An ASCII underscore

An identifier can then be followed by one of the below:

- Any of the above
- An ASCII numeric
- An ASCII single quote

An identifier cannot be a boolean literal or a keyword.

### 1.2 Literals

`<lit> := <int lit>|<float lit>|<string lit>|<char lit>|<bool lit>`

#### 1.2.1 Integer Literals

`<int lit>`

An integer literal starts with either a base prefix or an ASCII numeric, as listed below:

1. **0b**: Binary
  2. **0o**: Octal
  3. **0d**: Decimal
  4. **0x**: Hexadecimal
  5. Any ASCII numeric: Decimal
- For case 1 to 4:
    - It must be followed by a string of characters of that specific base.
  - For case 5:
    - It may then be followed by a string of ASCII numerics.
    - It may then be followed by the character `e`, if so, it must then be followed by a string of ASCII numerics.

If a sequence of character start with an ASCII numeric, and it otherwise satisfies the requirement of being an identifier, it is an invalid integer literal.

### 1.2.2 Floating Point Literals

`<float lit>`

A floating point literal starts with an ASCII numeric, followed by an ASCII period, followed by another string of ASCII numerics.

It may then be followed by the character `e`, if so, it must then be followed by a string of ASCII numerics.

### 1.2.3 Character Literals

`<char lit>`

A character literal starts with an ASCII single quote sign, followed by either one UTF-8 character or an string escape sequence, followed by another single quote sign.

### 1.2.4 String Literals

`<string lit>`

A string literal starts with an ASCII double quote sign, followed by a sequence of either UTF-8 characters or string escape sequence, followed by another single quote sign.

### 1.2.5 String Escape Sequence

Valid string escape sequences are:

- `\n`: Newline
- `\r`: Carriage return
- `\t`: Horizontal tab
- `\v`: Vertical tab
- `\\`: Backslash
- `\0`: Null
- `\x`, followed by 2 hexadecimal digits
- `\u{`, followed by 2 to 6 hexadecimal digits, followed by `}`

### 1.2.6 Boolean Literal

`<bool lit>`

Boolean literals are `true` or `false`.

## 1.3 Macro Directives

`<@ident>`

Macro directive starts with a `@`, it can then be followed by:

- A UTF-8 alphanumeric that is not an ASCII numeric
- A UTF-8 emoji
- An ASCII underscore
- An ASCII single quote

## 1.4 Parenthesis

A parenthesis is one of the following characters:

- {
- }
- [
- ]
- (
- )

## 1.5 Punctuations

`<punct> := <unreserved punct> | <reserved punct>`

A punctuation is a sequence of characters that satisfy **all** follow requirements:

- Is a UTF-8 punctuation character
- Is not a `<paren>`
- Is not an ASCII single quote or double quote

Some punctuations are reserved (`<reserved punct>`), the rest are not (`<unreserved punct>`).

This distinction is made to make room in the syntax for a potential future expansion of custom operators, for this reason occurrence of `<unreserved punct>` in source code is considered invalid in the current version.

- ,
- .
- =
- :=
- :
- ::
- \*
- ~
- &
- |
- ^
- >>
- <<
- >>=
- <<=
- &=
- |=
- ^=
- !
- &&

- ||
- +
- -
- /
- %
- +=
- -=
- \*=
- /=
- %=
- >
- <
- >=
- <=
- ==
- !=
- @

## 2 Syntax

### 2.1 Types

```
<ty> := <ident>
      | &<ident>
      | &mut <ident>
      | [<ty>]
      | &mut [<ty>]
      | ({<ty>},*)
      | struct \{ <pat ty pairs> \}
      | unoin \{ <pat ty pairs> \}
      | enum \{ {<ident>(<ty>)},* \}
```

### 2.2 Patterns

```
<pat> := <ident>
      | mut <ident>
      | ({<pat>},*)

<pat ty pairs> := {<pat>:<ty>},*
```

### 2.3 Expressions

```
<expr> := <ident>
        | <literal>
        | <op expr>
        | ({<expr>},*)
```

### 2.4 Assignments

```
<assign> := <pat> = <expr> ;

<expr> := <ident>
        | <literal>
        | <op expr>
        | ({<expr>},*)
```

### 2.5 Statements and Blocks

```
<stmt> := <expr> ;
        | <var decl>
        | <fn decl>
        | <typealias>
        | <newtype>
        | <if>
        | <loop>
        | <while>

<block> := \{ {<stmt>}* \};
```

### 2.6 Variable Declarations

```
<var decl> := <pat> : <ty> {= <expr>}? ;
            | <pat> {:=}|{: =} <expr> ;
```

### 2.7 Function Declarations

```
<fn decl> := <ident> :: (<pat ty pairs>) {= <expr> ;}|{<block>}|{;}
```

## 2.8 Type/Typealias Statements

`<newtype> := type <ident> = <ty> ;`

`<typealias> := typealias <ident> = <ty> ;`

## 2.9 If

`<if> := if <expr> <block> {else <block>}}?`

## 2.10 Loop

`<loop> := loop <block>`

## 2.11 While

`<while> := while <expr> <block>`