Arrow

Sanjay Sankaran

Contents

Arrow

Arrow is a general purpose high level scripting language, with dynamic typing. The arrow interpreter is made in Java.

Usage

Invoking the Arrow interpreter through the command line is easy.

- Compile java files (only required once) : `javac arrow/*.java`
- Execute main() : `java arrow.main path/to/your/file.ar`

Hello World

a quick hello world would be:

```
[print] <- "Hello World"
```

Features and Syntax

In arrow, everything is an expression, including the entire program itself. Thus, each line (except the last one) is ended with a comma (`,`).

Here is an example to echo an input by a user:

```
[print] <- "Enter something",
value <- [input] <- null,
[print] <- "You said " + value
```

As you can see, the commas make the program seem like an expression.

Variables and Types

Arrow currently supports the following types:

- `null` or `NULL`
- Numeric (int/float)
- Boolean
- String
- Function (Callable)

Since Arrow is dynamically typed, variables do not have to have a fixed type at declaration.

Operators

The operators in Arrow are as follows:

Numeric Operations:

- `a + b` (Numeric Addition)
- `a - b` (Numeric Subtraction)
- `a * b` (Numeric Multiplication)
- `a / b` (Numeric Division)

Logical Operations:

- `a > b` (Numeric comparison)
- `a < b` (Numeric comparison)
- `a == b` (Generic equality)
- `a && b` (Boolean and)
- `a || b` (Boolean or)

String Operations:

- `a + b` (String Concatenation)

Call/Assignment Operator (`<-`):

- `a <- value` (Variable Initialization/Assignment)
- `return_vals <- (func) <- args` (Function call)

Get/Pull Operator (`->`):

- `funcname <- funcarg -> funcreturn` (Function defenition)
- `? -> condition <- expression` (Conditional statement)

Literals

Arrow currently supports the following literals:

- String: `"Value"`
- Numeric: `123`, `456.789`, `.23`
- Null: `NULL` or `null`

Functions

Arrow currently supports functions with zero or one arguments. for example:

```
myfunc <- null -> {
    [print] <- "Myfunc Has been called"
}

say_hello <- name -> {
    "Hello " + name
}
```

Also, the return value of a function is the last expression inside it. For example:

```
somefunc <- null -> {
    a <- 10,
    b <- 20,
    a + b
}
```

will return `a + b`, which is `30`.

The following built-in functions have been implemented:

- `null <- [print] <- value` (prints `value` to the screen, and returns `null`)
- `null <- [print_raw] <- value` (prints `value` to the screen (without a trailing newline), and returns `null`)
- `<String> <- [input] <- query` (prints `query` to the screen, and returns the user's input as a string)
- `<String> <- [input_num] <- query` (prints `query` to the screen, and returns the user's input as a number)

Conditionals

Conditional Statements in arrow return a value if a condition has been met. For example:

```
value <- 1,

? -> (value == 1) <- {
    [print] <- "Hello World"
}
```

Arrow Doesn't have any equivalent of `else if` and `else` statements.

Examples

Hello World

```
[print] <- "Hello World"
```

output:

```
Hello World
```

Hello Person

```
name <- [input] <- "What's your name? ",
[print] <- "Hello " + name
```

output:

```
What's your name? JaSON
Hello JaSON
```

Add Two Numbers

```
a <- [input_num] <- "Enter a number : ",
b <- [input_num] <- "Enter a number : ",
sum <- a + b,
[print] <- "Their sum is " + sum
```

Pass/Fail

```
score <- [input_num] <- "Enter your percentage : ",

? -> (score > 80) <- {
    [print] <- "You passed!"
}
? -> (score < 80) <- {
    [print] <- "You Failed."
}
```

output:

```
Enter your score : 96
You passed!
```

```
Enter your score : 15
You Failed.
```

Print Stars

```
n <- ([input_num] <- "Enter size : ") + 1,
i <- 1,

while <- (i < n) -> {

    j <- 0,
    while <- (j < i) -> {

        [print_raw] <- "*",
        j <- j + 1

    },

    [print] <- null,

    i <- i + 1
}
```

output:

```
Enter size : 5
*
**
***
****
*****
```

Hello Person (with functions)

```
sayhello <- name -> {
    "Hello " + name
},
[print] <- [sayhello] <- [input] <- "What's your name? "
```

output:

```
What's your name? JaSON
Hello JaSON
```

Note: the function definition can be shortened as `sayhello <- name -> "Hello " + name`

Compromises and Scope for further development

The following features haven't been implemented:

- Advanced Control Flow (`else if` and `else` statements)
- List/Tuple type
- Multi-argument Functions