Io (programming language)

Io is a pure object-oriented programming language inspired by Smalltalk, Self, Lua, Lisp, Act1, and NewtonScript.[2] Io has a prototype-based object model similar to the ones in Self and NewtonScript, eliminating the distinction between instance and class. Like Smalltalk, everything is an object and it uses dynamic typing. Like Lisp, programs are just data trees. Io uses actors for concurrency.

Remarkable features of Io are its minimal size and openness to using external code resources. Io is executed by a small, portable virtual machine.

Io

Io	
Paradigm	object-oriented prototype-based
Designed by	Steve Dekorte
Developer	Steve Dekorte, Jonathan Wright, Jeremy Tregunna
First appeared	2002
Stable release	20170906 / August 11, 2017[1]
Typing discipline	dynamic, strong
Website	iolanguage.org (https://iolanguage.org/)
Major implementations	
Io (http://www.iolanguage.com/) Io.NET (https://web.archive.org/web/20110716175739/http://synrc.com/io/index.htm)	
Influenced by	
Smalltalk, NewtonScript, Self, Lua, Lisp, Python, Act1	
Influenced	
Ioke, Potion	

Contents

History

The language was created by Steve Dekorte in 2002, after trying to help a friend, Dru Nelson, with his language, Cel. He found out that he really didn't know much about how languages worked, and set out to write a tiny language to understand the problems better.[3]

Philosophy

Io's goal is to explore conceptual unification and dynamic languages, so the tradeoffs tend to favor simplicity and flexibility over performance.

Features

- Pure object-oriented based on prototypes

- Code-as-data / homoiconic
- Lazy evaluation of function parameters
- Higher-order functions
- Introspection, reflection and metaprogramming
- Actor-based concurrency
- Coroutines
- Exception handling
- Incremental garbage collecting supporting weak links
- Highly portable
- DLL/shared library dynamic loading on most platforms
- Small virtual machine

Syntax

In its simplest form, it is composed of a single identifier:

```
doStuff
```

Assuming the above doStuff is a method, it is being called with zero arguments and as a result, explicit parentheses are not required.

If doStuff had arguments, it would look like this:

```
doStuff(42)
```

Io is a message passing language, and since everything in Io is a message (excluding comments), each message is sent to a receiver. The above example demonstrates this well, but not fully. To describe this point better, let's look at the next example:

```
System version
```

The above example demonstrates message passing in Io; the "version" message is sent to the "System" object.

Operators are a special case where the syntax is not as cut-and-dried as the above examples. The Io parser intercepts a set of operators defined by the interpreter, and translates them to method calls. For example, the following:

```
1 + 5 * 8 + 1
```

translates to:

```
1 + (5 *(8)) +(1)
```

All operators in Io are methods; the fact that they do not require explicit parentheses is a convenience. As you can see, there is also a little bit of operator precedence happening here, and the precedence levels are the same as with the C precedence levels.

Methods and blocks

In Io there are two ways of creating anonymous functions: methods and blocks. Between them, they are almost identical except for scope. While blocks have lexical scope, methods have dynamic scope.

Both *method* and *block* are higher-order functions.

Examples

The ubiquitous Hello world program:

```
"Hello, world!" println
```

New objects are created by cloning objects. In Io specifically, a new, empty object is created and only the differences between it and its parent are stored within the new object; this behavior is known as differential inheritance. An example of this behavior is shown:

```
A := Object clone          // creates a new, empty object named "A"
```

A simple non-recursive factorial function, in Io:

```
factorial := method(n,
    if(n == 0, return 1)
    res := 1
    Range 1 to(n) foreach(i, res = res * i)
)
```

Because assignment of `res * i` to `res` is the last action taken, the function implicitly returns the result and so an explicit return expression is not needed. The above demonstrates the usage of ranges, and doesn't use a `for()` loop, which would be faster.

See also

- Ioke (programming language)

References

1. "Io Releases" (https://github.com/IoLanguage/io/releases). *GitHub*. Retrieved 2020-02-06.
2. Io Programming Guide (http://iolanguage.org/guide/guide.html#Introduction-Overview)
3. Tate, Bruce (2010). "Chapter 3: Io" (https://archive.org/details/sevenlanguagesin00tate/page/60). *Seven Languages in Seven Weeks: A Pragmatic Guide to Learning Programming Languages* (1st ed.). Raleigh, North Carolina: Pragmatic Bookshelf. p. 60, 72 (https://archive.org/details/sevenlanguagesin00tate/page/60). ISBN 978-1934356593.

External links

- Io (http://iolanguage.com/) home page

- Io at Synrc Research Center (https://web.archive.org/web/20121012010044/http://synrc.com/research/io/index.htm)
- Io (https://curlie.org/Computers/Programming/Languages/Io) at Curlie
- Jasmine.Io (https://github.com/bekkopen/jasmineio) BDD Testing Framework for Io

---