O Documentation

Release 2.0

Phase

Nov 22, 2017

Contents

The O Language is an esoteric programming language used for code-golf. You can try it online at https://o-lang.herokuapp.com.

Docs:

CHAPTER 1

1. Getting Started

Getting started in O is easy! Just follow the steps below and you'll be ready to go!

1.1 1.1. Getting the interpreter

You can download latest release from http://github.com/phase/o/releases or clone the repository and compile it manually:

```
git clone https://github.com/phase/o && cd o
make all
./o
```

Running the executable without any arguments will open the REPL, which you can type lines of O code into to have them interpreted.

You can also go to http://o-lang.herokuapp.com and use the always-updated-interpreter so you won't have to recompile every update.

1.2 1.2. Let's test it out!

Whether you're on the online IDE or using the REPL, this *Hello World* program will run on both:

```
"Hello, World!"o
```

This is the simplest *Hello World* program in O. It pushes the string `Hello, World!` to the stack and outputs it with `o`. You will learn about why the `o` isn't needed in the next section.

CHAPTER 2

2. The Basics

O revolves around one stack in which you can push and pop values to and from. Here are some basic rules for writing O code.

2.1 2.1. I/O

There are a couple ways to get a user's input. `i` will push the input to the stack as a string, while `j` will push the input to the stack as an integer. `o` pops the top value off the stack and outputs it. `p` outputs with a new line. Here's an example:

```
>>> io
test
test
```

This is the smallest cat program possible. `i` gets the input and pushes it, `o` pops it and outputs it to stdout.

Another cool feature of O is that the stack contents will be outputted when the code finishes execution. Meaning `1234` will output `1234`, while `1234oooo` will output `4321`. This feature does not work in the REPL.

2.1.1 2.1.1. Input as a number

`j` will parse the input as a number, and throw an error if otherwise.

```
>>> j5+o
5
10
```

`q` will push the input as a number if it is one, and push it as a string if otherwise.

```
>>> q5+o
5
10
```

```
>>> q5+o
hi
hi5
```

`Q` will do the same except assign it to a variable called *Q*. You will learn about variables more later.

2.2 2.2. Number Literals are pushes individually

`1234` doesn't push the number *one thousand twenty-four*, it pushes `1`, then `2`, `3`, and finally `4`. Each individual digit is pushed to the stack:

```
>>> 1234oooo
4321
```

Each `o` pops one number off and outputs it, meaning the number that was pushed last will pop off first.

2.2.1 2.2.1. Hexadecimal works too

Hexadecimal numbers work for capital notation. Lowercase notation is saved for other functions.

```
>>1Aoo
101
```

`1` was pushed to the stack, then `A` pushed `10` to the stack, and then they were outputted.

2.3 2.3. Strings are enclosed in quotes

To make a string, you just need to enclose it within quotes.

```
>>> "Hello, World!"o
Hello, World!
```

What about printing something different?

```
>>> "Hello'World!"oo
World!Hello
```

2.3.1 2.3.1 Strings don't need quotes at all

Remember when I said you need to put them in quotes? I lied. `'` has some interesting properties with strings. When used by itself, it will push the next character to the stack as a string.

```
>>> 'ao
a
```

If you are in the middle of making a string, it will push the current string buffer to the stack and start making a new string (like a macro for `""`).

```
>>> "a'a"oo
aa
>>> "hello'madam"oo
madamhello
```

The next section will cover basic arithmetic.

2.3.2 2.3.2 Strings don't need quotes all the time

If a string is at the end of a file, you don't need to put a quote at the end.

```
"Hello, World!
```

Put that in a file and it will print:

```
Hello, World!
```

CHAPTER 3

3. Arithmetic

Arithmetic is in postfix notation, along with every other operator in O.

3.1 3.1. Simple Arithmetic

Numbers are the easiest objects to modify. Doing basic arithmetic is as easy as putting the sign you want.

3.1.1 3.1.1. Addition

Addition uses the + operator.

```
>>> 12+o
3
```

3.1.2 3.1.2. Subtraction

Subtraction uses the − operator.

```
>>> 75-o
2
```

3.1.3 3.1.3. Multiplication

Multiplication uses the ⋆ operator.

```
>>> 85*o
40
```

3.1.4 3.1.4. Division

Division uses the / operator.

```
>>> 42/o
2
```

3.1.4.1 3.1.4.1. Remainders

Remainders will go out to 6 decimal places.

```
>>> 54/o
1.25
>>> 5/3
1.666667
```