

---

# O Documentation

*Release 2.0*

**Phase**

November 16, 2015



<b>1</b>	<b>1. Getting Started</b>	<b>3</b>
1.1	1.1. Getting the interpreter . . . . .	3
1.2	1.2. Let's test it out! . . . . .	3
<b>2</b>	<b>2. The Basics</b>	<b>5</b>
2.1	2.1. I/O . . . . .	5
2.2	2.2. Number Literals are pushes individually . . . . .	5
2.3	2.3. Strings are enclosed in quotes . . . . .	6



The O Language is an esoteric programming language used for [code-golf](#). You can try it online at <https://o-lang.herokuapp.com>.

Docs:



---

## 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
gcc o.c -o o
./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.



---

## 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:

```
>>> i o
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.**

### 2.2 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

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
```