# Function Usage

## +

Add, Concatenate

```
2 4 +                -> 6

(1 2 3)(4 5 6) +   -> (1 2 3 4 5 6)
{+} (1 2 “3”) +    -> ({+} 1 2 “3”)
(1 2 “3”) 8 +      -> (1 2 “3” 8)
```

## -

Subtract, Array difference

```
8.6 2.3 -                 -> 6.3

(1 2 3 4 5) (3 5 2) -    -> (1 4)
(1 1 1 1 1) (1) -        -> ()
```

## *

Multiply, Duplicate, Repetitively Execute

```
5 3 *                 -> 15

(1 2 3 4) 3 *        -> (1 2 3 4 1 2 3 4 1 2 3 4)
3 (1 2 3 4) *        -> ((1 2 3 4)(1 2 3 4)(1 2 3 4))

{2} 5 *               -> 2 2 2 2 2
```

## /

Divide, Map

```
9 2 /                -> 4
9. 2 /               -> 4.5

(1 2 3 4) {2+} /    -> (3 4 5 6)
```

## =

Equality

```
2 4 =            -> 0
1 1 =            -> 1
(1 2)(1 2) =    -> 1
(3) {3} =        -> 0
```

## <

Less than

```
4 5 <               -> 1
(1 2) (3 4) <       -> 1
“b” “a” <           -> 0
(1) 1 <             -> 0
```

## >

More than

```
(1) 1 >             -> 1
“b” “a” >           -> 1
(1 2) (3 4) >       -> 0
4 5 >               -> 0
```

## ^

Power

```
2 10 ^     -> 1024
5.5 3 ^    -> 166.375
```

## %

Modulus, Filter, Zip

```
4 2 %                -> 0
1337 6 %             -> 5

(1 2 3 4 5) {2%}  -> (1 3 5)
(1 2 3 4 5) {2%n} -> (2 4)

(1 2 3) (4 5 6) % -> ((1 4) (2 5) (3 6))
```

## &

Boolean AND

```
1234723 0 &          -> 0
43 (0) &             -> 1
{2} 1 &              -> 1
```

## |

Boolean OR

```
1234723 0 |          -> 1
() () |              -> 0
(0) 0 |              -> 1
```

## @

Rotate 3

```
1 2 3 @        -> 3 1 2
(1) {2} 8. @   -> 8. (1) {2}
```

## ~

Not, Evaluate, Execute

```
53 ~           -> -54

"2 4+" ~       -> 6

{2 4+} ~       -> 6
```

## :

Index (access *n*th element), Index range

```
(1 2 3 4) 2 :       -> 3

(1 2 3 4 5) 2 4 :   -> (3 4)
```

## ;

Search (returns index, or -1 if not found)

```
(1 2 3 4 5) 2 ;     -> 1
(1 2 3 4 5) 7 ;     -> -1
```

## [

Increment, Rotate Array Clockwise

```
583 [          -> 584

(1 2 3 4) [    -> (4 1 2 3)
```

## ]

Decrement, Rotate Array Counterclockwise

```
583 ]          -> 582

(1 2 3 4) ]    -> (2 3 4 1)
```

## $

Thread/Weave

```
(1 2 3 4 5) 6 $    -> (1 6 2 6 3 6 4 6 5)
(1 2 3) (4) $      -> (1 (4) 2 (4) 3)
"Hello" "." $      -> "H.e.l.l.o"
```

## a

Absolute, Boolean ANY

```
-4 a       -> 4
4 a        -> 4
```

```
(1 0 {} ()) a     -> 1
(0 {} ()) a       -> 0
(-1) a            -> 1
```

## A

Boolean ALL

```
(1 2 3 4 5) A  -> 1
(1 2 3 0 4) A  -> 0
```

## b

Sum

```
(1 2 3 4 5) b   -> 15
((1)(2)(3)) b   -> (1 2 3)
(1 (2) 3) b     -> (1 2 3)
```

## B

Product

```
(1 2 3 4 5) B   -> 120
({2} 3 4) B     -> 2 2 2 2 2 2 2
```

## c

Cycle stack clockwise

```
1 2 3c          -> 3 1 2
```

## C

Cycle stack, Reverse Array

```
1 (2) {3} 2 C   -> (2) {3} 1
1 (2) {3} -1 C  -> (2) {3} 1
```

```
(1 2 3 4 5) C   -> (5 4 3 2 1)
```

## d

Duplicate

```
1 2 3 d      -> 1 2 3 3
```

## D

Drop

```
1 2 3 D      -> 1 2
```

## e

Contains/Is in Array

```
(1 2 3 4) 4 e    -> 1
(1 2 3 4) 5 e    -> 0
(1 (2) 3) (2) e -> 1
```

## E

Flatten, Convert to Float

```
5 E         -> 5.0
5.0 E       -> 5.0

((1 2) (3 4) (5 6)) E           -> (1 2 3 4 5 6)
(1 (2 (3) (4 (5) (6)))) E       -> (1 2 3 4 5 6)
```

## f

If, Conditional Drop/Drop If

```
5 1 {3+} f     -> 8
5 0 {3+} f     -> 5

1 (2) f        -> (2)
0 {3+} f       ->
```

## F

If Else, Conditional Square-Off/Drop If Else

```
1 1 {2+} {3+} F       -> 4
1 0 {2+} {3+} F       -> 3

1 (2) (3) F           -> (3)
0 (2) (3) F           -> (2)
```

## g

Number to String

```
123 g   -> “123”
```

## G

Encase, Encase up to *n*

```
(1) (2) {3} 2 G     -> (1) ((2) {3})
(1) (2) {3} 8 G     -> ((1) (2) {3})
(1) (2) {3} G       -> ((1) (2) {3})
```

## h

Forcibly halt program

## i

Char to Number, String to Number

```
“0.5” i        -> .5
49 i           -> 1
```

## I

IsNumeric

```
40 I      -> 1
3 I       -> 0

“34” M    -> 1
“823d” M  -> 0
```

## j

Remove duplicates, Convert to Integer

```
(1 2 2 3 3 3 4 5) j  -> (1 2 3 4 5)

4.1 j                -> 4
4.7 j                -> 4
```

## J

Sort Array

```
(1 4 5 2 5 6 7) J -> (1 2 4 5 5 6 7)
```

## k

Push Space/Push 32

```
k   -> 32
```

## K

Push Linefeed/Push Newline/Push 10

```
K   -> 10
```

## l

Len, Log/Log *e*

```
(1 3 {2}) l     -> 3

7 l             -> 1.945910149…
```

## L

Log *n*

```
8 10 L          -> 2.30258509…
```

## m

Prime/isPrime, Minimum

```
18 m            -> 0
17 m            -> 1
937 m           -> 1

(1 2 3 4 5) m  -> 1
(1 (1) {2}) m  -> 1
```

## M

Maximum

```
(1 2 3 4 5) M  -> 5
(1 (2) {3}) M  -> {3}
```

## n

Boolean NOT

```
1 n         -> 0
0 n         -> 1
(1 2 3) n  -> ()
```

## N

isAlphaNumeric

```
97 N        -> 1
18 N        -> 0
54 N        -> 1
```

```
"abC4" N   -> 1
"asdf$" N  -> 0
```

## o

Fold left

```
(1 2 3 4 5)+o      -> 15
(1 2 3 4 5){2**}o  -> 1920
```

## O

isWhitespace

```
k O        -> 1
9 O        -> 1
83 O       -> 0

"    " O    -> 1
'\n\t' O     -> 1
```

## p

Convert and output

```
48             >OUTPUTS> '0'
(47 48 49)     >OUTPUTS> '123'
"Hello World" >OUTPUTS> 'Hello World'
```

## P

Output representation

```
48              >OUTPUTS> '48'
(47 48 49)      >OUTPUTS> '(47 48 49)'
"Hello World"  >OUTPUTS> '(72 101 108 108 111 32 87 111 114 108 100)'
```

## r

Range, Fold right

```
5 r                 -> 0 1 2 3 4
3 5 r               -> 3 4

(1 2 3 4 5) + r   -> 15
```

## R

Random, Random Element, Randrange

```
(1 2 3 4 5 6) R      -> 5
(1 2 3 4 5 6) R      -> 1
```

```
1 5 R                -> 3
1 5 R                -> 2

(1 2 3 4 5 6) 3 R    -> (4 2 5)
```

## s

To String

```
5 s      -> ‘5’
(1 2) s  -> ‘(1 2)’
```

## S

Swap

```
5 (2) S    -> (2) 5
```

## t

Get character/getchar

## T

Get line/getline

## u

Do while

```
10 ]u    -> 0
```

## v

Set intersection

```
5r 3 8r v    -> (0 1 2 3 4 5 6 7)
```

## V

Set difference

```
5r 3 8r V    -> (0 1 2)
```

## w

While

```
10 ]w    -> 0
```

## W

Set Symmetric Difference

```
5r 3 8r W    -> (0 1 2 5 6 7)
```

## x

Stack length

```
1 2 3 x    -> 3
```

## X

Clear stack

```
1 2 3 (4) {5} X   ->
```

## y

Split at

```
“Hello World” “ “ y    -> (“Hello” “World”)
```

## Y

Find all

```
“A b c” k Y    -> (1 3)
```

## z

Zip with

```
5r 5r + z      -> (0 2 4 6 8)
5r 5r {2G}z    -> ((0 0) (1 1) (2 2) (3 3) (4 4))
5r 5r 3z       -> ((0 0 3) (1 1 3) (2 2 3) (4 4 3))
```

## #

Print stack for debugging

```
1 2 (3) {4+}    >OUTPUTS> “1 2 (3) {4+}”
```