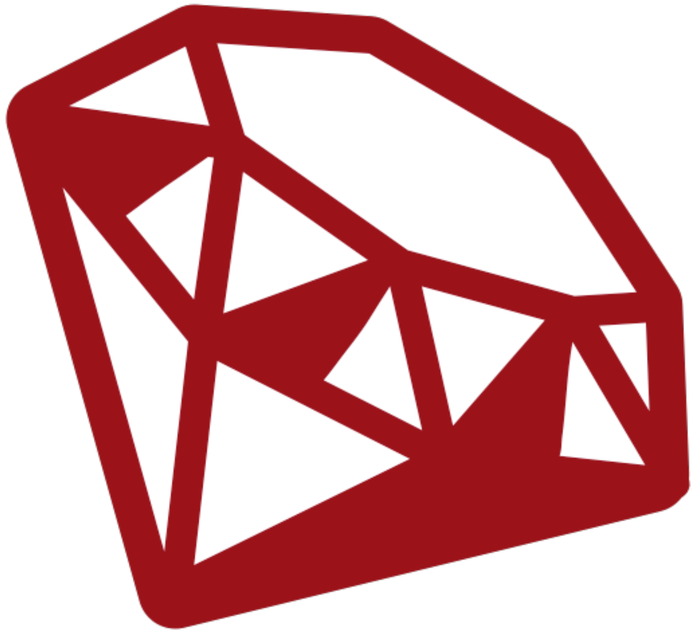


katie@glasnt.com

Chrome 63 Inspector

JavaScript

```
> 4 + 2
< 6
> 4 - 2
< 2
> 4 - "2"
< 2
> 4 + "2"
< "42"
```

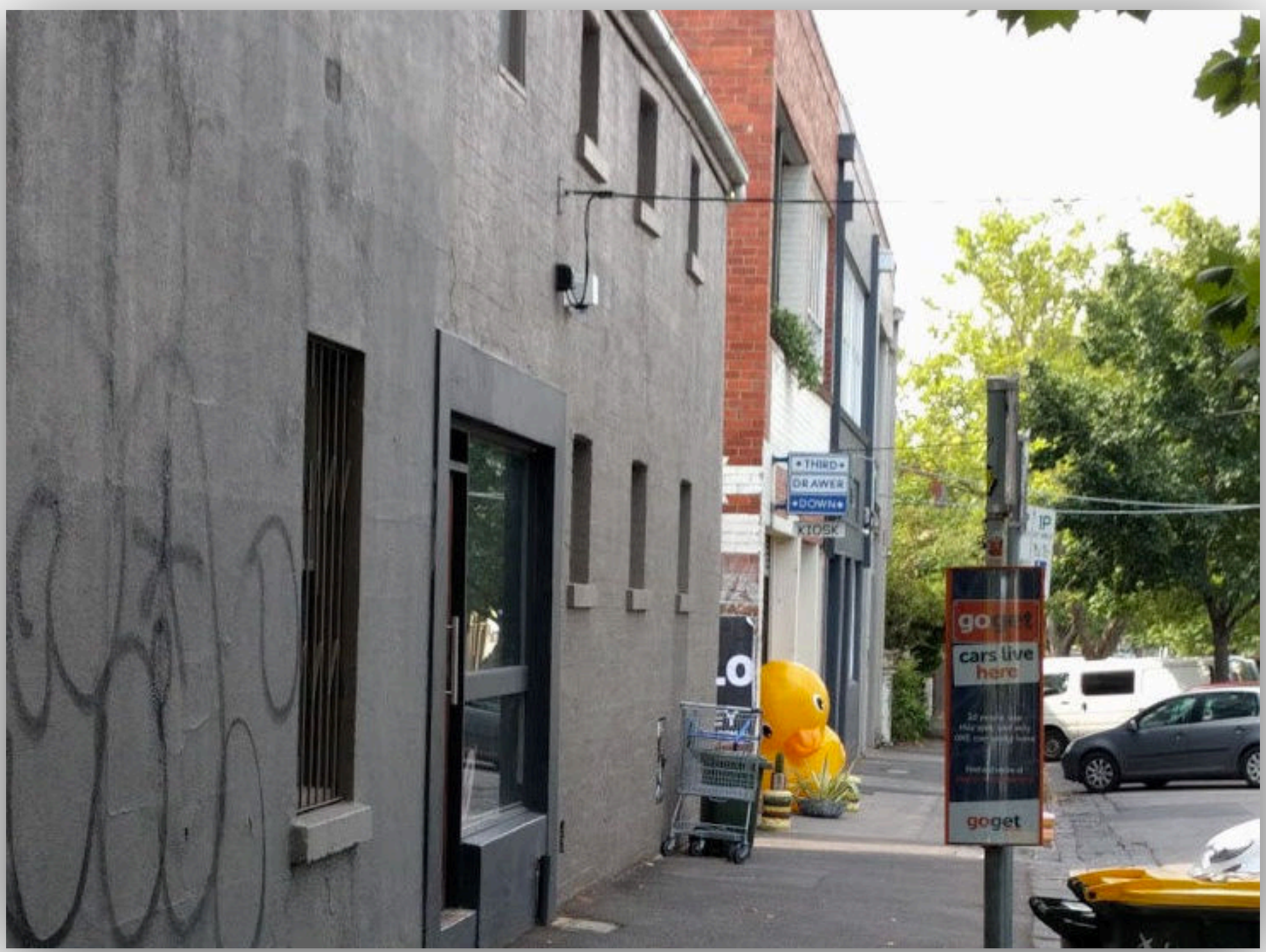
JavaScript

```
> 1 == "1"
< true

> 1 === "1"
< false
```

JavaScript

```
> [] + []
< ""
> [] + {}
< "[object Object]"
> {} + []
< 0
> {} + {}
< NaN
```


wat


THIRD
DRAWER
DOWN
KIOSK
cars live
here
goget

Chrome 63 Inspector

JavaScript

```
> 4 + 2
< 6
> 4 - 2
< 2
> 4 - "2"
< 2    // implicit type coercion
> 4 + "2"
< "42" // overloaded operand
```

All resources

↓

glasnt.com/wat

JavaScript

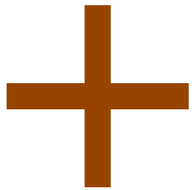
```
> 1 == "1"  // equality, with type coercion
< true

> 1 === "1" // equality without type coercion
< false
```

JavaScript

```
> [] + []
< ""  // ???
> [] + {}
< "[object Object]" // ???
> {} + []
< 0   // ???
> {} + {}
< NaN // ???
```

glasnt.com/wat

JavaScript



glasnt.com/wat

https://www.ecma-international.org/ecma-262/#sec-addition-operator-plus

12.8.3 The Addition Operator (+)

NOTE The addition operator either performs string concatenation or numeric addition.

Convert both to a primitive

undefined, null, Bool, Number, String

`valueOf()`

`toString()`

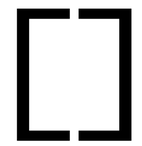
String?

`String()`, concatenate

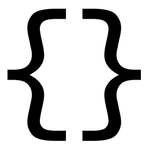
otherwise

`Number()`, sum

JavaScript



```
> a = []
> a.valueOf()
< [] // not primitive ✗
> a.toString()
< "" // primitive ✔
```



```
> o = {}
> o.valueOf()
< {} // not primitive ✗
> o.toString()
< "[object Object]" // primitive ✔
```

glasnt.com/wat

JavaScript

```
> [] + []   // "" + ""
< ""
> [] + {}   // "" + "[object Object]"
< "[object Object]"
> {} + []   // code block + "" unary addition
< 0                          // > Number("")
                             // < 0
> {} + {}   // code block + {}
< NaN                        // > Number({})
                             // < NaN
```

glasnt.com/wat

Chrome 64 Inspector

JavaScript

Chrome 64

```
> [] + []   // "" + ""
< ""
> [] + {}   // "" + "[object Object]"
< "[object Object]"
> {} + []   // "[object Object]" + ""
< "[object Object]"
```

no longer a code block

```
> {} + {}   // two strings
< "[object Object][object Object]"
```

glasnt.com/wat

JavaScript
Chrome 64

```
> [] + {}
< "[object Object]"
> {} + []
< "[object Object]"
```

Idempotent!

JavaScript isn't awful

JavaScript is awe-ful

So, don't use it.

List of languages that compile to JavaScript

https://github.com/jashkenas/coffeescript/wiki/list-of-languages-that-compile-to-js

340+ entries

Using another language won't save you.

```
$ irb
```

```
irb> not true && false
true

irb> not true and false
false
```

Ruby

```
irb> not true && false
true

irb> not true and false
false
```

..., &&, ..., not, and, ...

glasnt.com/wat

Ruby

```
irb> not (true && false)
true

irb> (not true) and false
false
```

..., &&, ..., not, and, ...

glasnt.com/wat

Ruby

```
irb> a && b or c && d
```

glasnt.com/wat

```
$ python
```

Python

```
>>> a = 256
>>> b = 256
>>> a is b
True
```

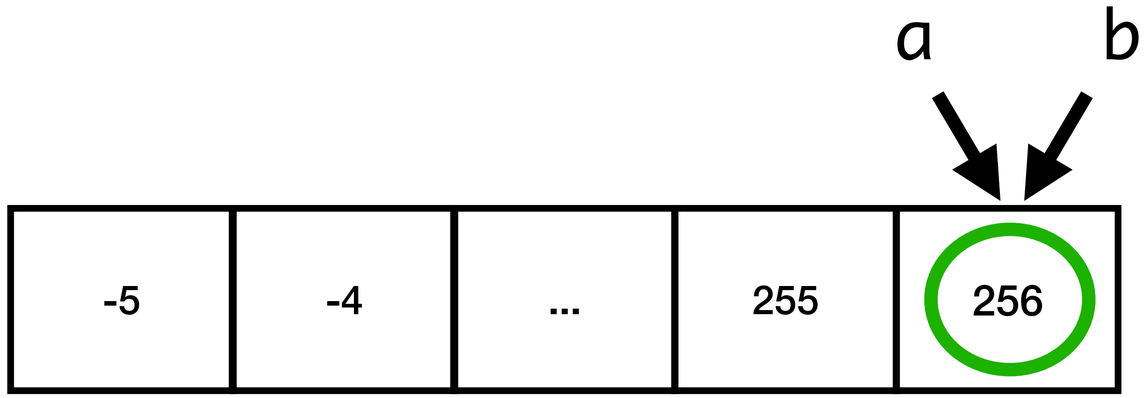
glasnt.com/wat

Python

```
>>> a = 257
>>> b = 257
>>> a is b
False

>>> a = 257; b = 257
>>> a is b
True
```

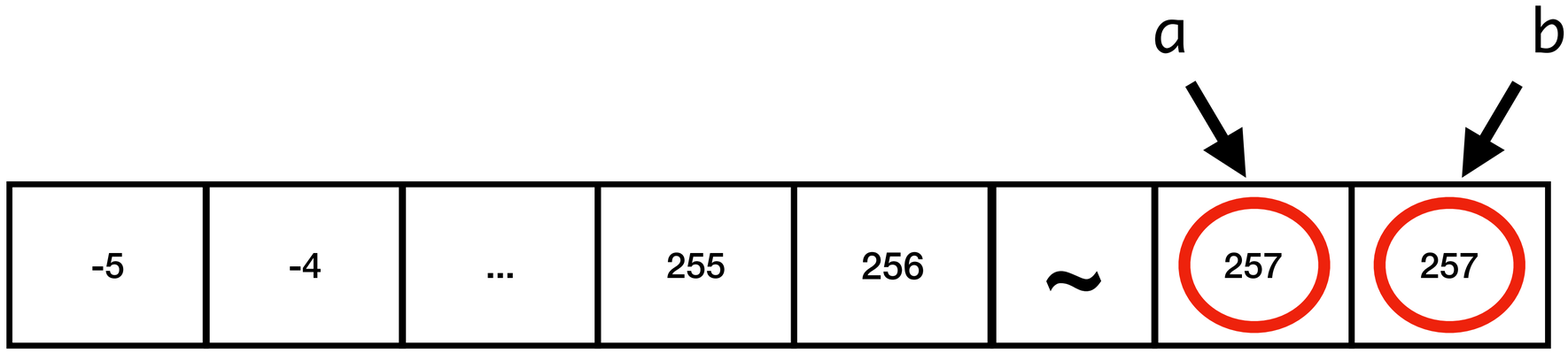
glasnt.com/wat

```
$ python
>>> a = 256
>>> b = 256
>>> a is b
True
```



Python

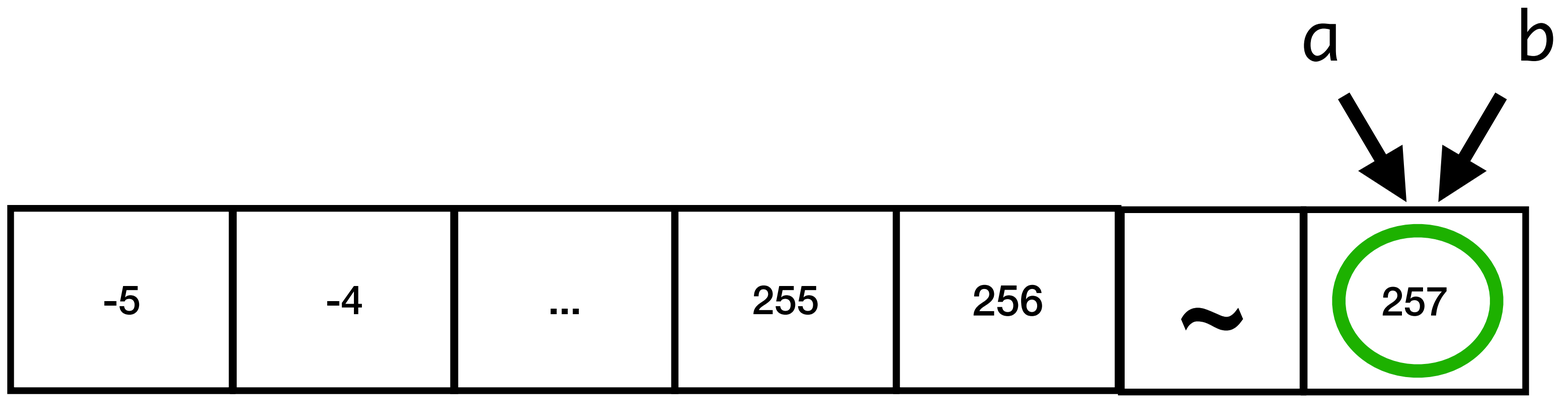
```
>>> a = 257
>>> b = 257
>>> a is b
False
```



glasnt.com/wat

Python

```
>>> a = 257; b = 257
>>> a is b
True
```



glasnt.com/wat

```
$ ghci
```

Haskell

```
λ> let a = 2 + 2

λ> a

4


λ> let a = 2 + 2
       where 2 + 2 = 5

λ> a

5
```

glasnt.com/wat

Haskell

```
fib :: Int -> Int
fib 0 = 0
fib 1 = 1
fib n = fib (n - 1) + fib (n - 2)
```

Haskell

```haskell
fib :: Int -> Int
fib 0 = 0
fib 1 = 1
fib n = fib (n - 1) + fib (n - 2)
```

Ruby

```ruby
def fib(n)
    return n if (0..1).include? n
    (fib(n - 1) + fib(n - 2))
end
```

```
$ bash
```

Bash

```
$ 4 + 2
bash: 4: command not found

$ $((4 + 2))
bash: 6: command not found

$ echo $((4 + 2))
6
```

```
$ iex
```

Elixir

```
iex> Enum.map(1..5, fn(x) -> x*x end)
[1, 4, 9, 16, 25]

iex> Enum.map(6..10, fn(x) -> x*x end)
'$1@Qd'
```

glasnt.com/wat

Elixir

```
iex> a = Enum.map(6..10, fn(x) -> x*x end)
iex> Enum.map(a, fn(x) -> IO.puts x)

36
49
64
81
100
```

glasnt.com/wat

```
iex> Enum.map(65..90, fn(x) -> x end)

'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

```
iex> Enum.map(65..90, fn(x) -> x end)

'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

$ go

Go

```
package main

import ("fmt")

func main() {
  var a int8 = -128
  fmt.Println(a/-1)
}


-128
```

glasnt.com/wat

Go

```
package main

import ("fmt")
// int8 range: -128 to 127
func main() {
  var a int8 = -128
  fmt.Println(a/-1)
}


-128 // integer overflow
```

```
$ gcc
```

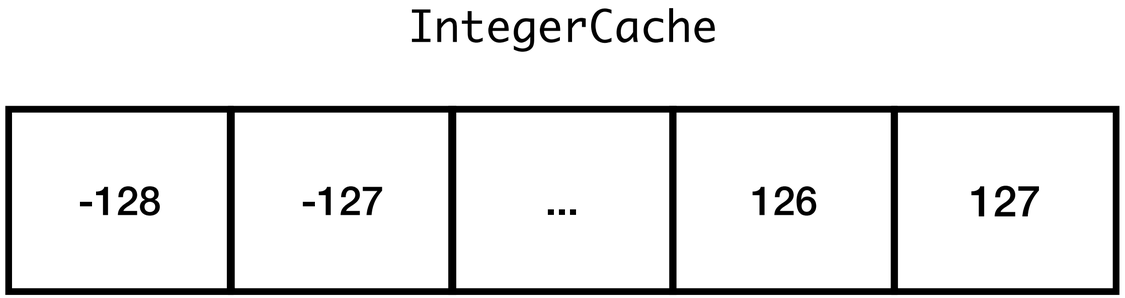
C, C++

```
> printf("wat??!")
wat|


// Trigraphs   ISO 646

// ??! ➡ |
// ??< ➡ {
// ??> ➡ }
// ??= ➡ #
```

```
$ java
```

Java

```
java> Integer a = 128;
java> Integer b = 128;
java> a <= b
true

java> a >= b
true

java> a == b
false
```



glasnt.com/wat

Java

```
java> Integer a = 128;
java> Integer b = 128;
java> a <= b
true

java> a >= b
true
```

```
java> a.equals(b)
true
```

glasnt.com/wat

Java

```
java> int a = 128;
java> int b = 128;
java> a <= b
true

java> a >= b
true

java> a == b
true
```

glasnt.com/wat

```
$ perl
```

Perl

```
> if ("a" == "b") {
    print "true"
  } else {
    print "false"
  }


true
```

==  eq

glasnt.com/wat

Perl

```
> if ("a" eq "b") {
    print "true"
  } else {
    print "false"
  }


false
```

glasnt.com/wat

```
$ php
```

PHP

```
php> echo (TRUE ? "True" : "False");
true

php> echo (FALSE ? "True" : "False");
false
```

glasnt.com/wat

```
php> echo (FALSE ? "one" :
           FALSE ? "two" :
                   "three");
three
```

```
php> echo (FALSE ? "one" :
           TRUE ? "two" :
                  "three");
two
```

```
php> echo ( TRUE ? "one" :
            TRUE ? "two" :
                   "three");

two
```

PHP

```
php> echo ((TRUE ? "one" :
            TRUE)? "two" :
                   "three");

two
```

```
php> echo ((TRUE ? "one" : TRUE)
           ? "two"
           : "three");
```

avoid "stacking" ternary expressions

- php.net

glasnt.com/wat

```
> powershell
```

PowerShell

```
PS> if (2 > 1) { "true"  }
          else   { "false" }

true

PS> if (2 < 1) { "true"  }
          else   { "false" }

The '<' operator is reserved for future use.
```

glasnt.com/wat

PowerShell

```
PS> if (2 -gt 1) { "true"  }
        else   { "false" }

true

PS> if (2 -lt 1) { "true"  }
        else   { "false" }

false
```

glasnt.com/wat

Every programming language is **different**

Every programming language solves a problem

Being different
is **good**

katie@glasnt.com