```
> print("Screentest: max length = 44. Ish.")
This is a large text screentest
> abs(JS.maxint)  // This is a comment.
# Under..Prog.. Pecul..?... UPPIES!

9,007,199,254,740,991
> for (i; i > j; i++) {
    print('hi')
  }
// E_OUT_OF_SPACE
```

Understanding programming peculiarities

Katie McLaughlin (they/them)
DevRel @ Google Cloud

```
console.log('Hello world!');
```

JavaScript

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

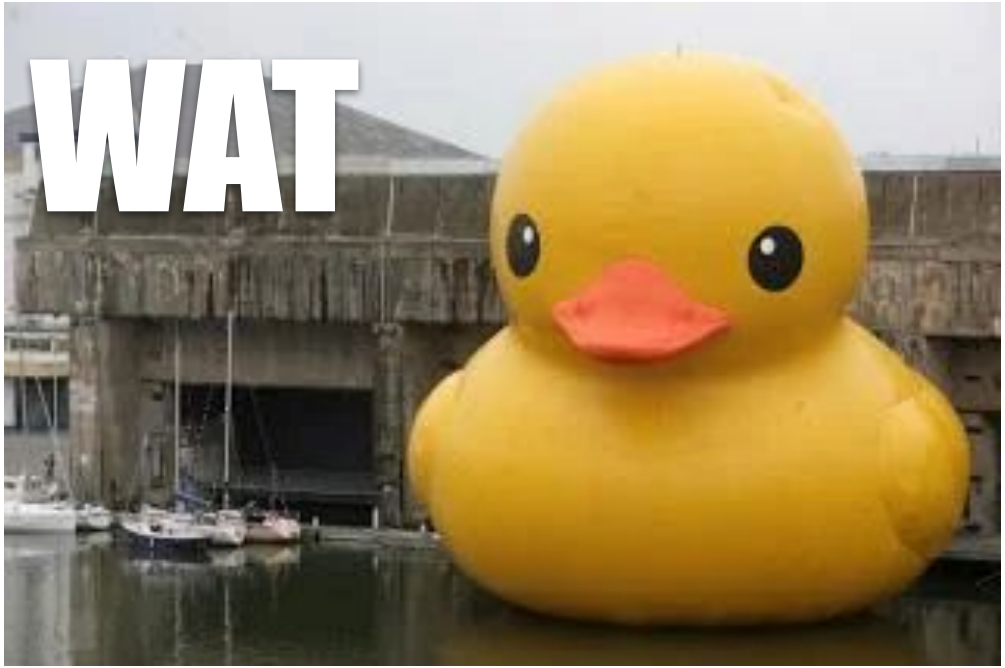
```
> 1 == "1"
  true
> 1 === "1"
  false
```

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

wat !

Wat

A lightning talk by Gary Bernhardt from CodeMash 2012



destroyallsoftware.com/talks/wat

~~wat!~~

why?

Hofman, Florentijn. *Rubber Duck*. PVC. Photo: 564dude (CC BY-SA 3.0)

~~wat!~~

why?

how?

Wild Republic Toys, *Rubber Duck - Cobra* Product photograph.

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

```
> 4 + 2
  6
> 4 - 2
  2
> 4 - "2"   // implicit coercion
  2
> 4 + "2"   // overloaded operator
  "42"
```

```
> 1 == "1"    // with type coercion
  true
> 1 === "1"   // without type coercion
  false
```

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

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

ECMA-262

ECMAScript® 2024 Language Specification

Sources available at glasnt.com/wat

ECMA-262

ECMAScript® 2024 Language Specification

13.8.1 The Addition Operator (+)

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

13.8.1.1 Runtime Semantics: Evaluation

AdditiveExpression **:** *AdditiveExpression* **+** *MultiplicativeExpression*

1. Return ? EvaluateStringOrNumericBinaryExpression(*AdditiveExpression*, **+**, *MultiplicativeExpression*).

13.15.4 EvaluateStringOrNumericBinaryExpression

The abstract operation EvaluateStringOrNumericBinaryExpression takes arguments [...]

1. [...]
1. Return ? ApplyStringOrNumericBinaryOperator(*lval*, *opText*, *rval*).

Sources available at glasnt.com/wat

13.15.3 ApplyStringOrNumericBinaryOperator (*lval*, *opText*, *rval*)

1. If *opText* is **+**, then
   a. Let *lprim* be ? ToPrimitive(*lval*).
   b. Let *rprim* be ? ToPrimitive(*rval*).
   c. If *lprim* is a String or *rprim* is a String, then
      i. Let *lstr* be ? ToString(*lprim*).
      ii. Let *rstr* be ? ToString(*rprim*).
      iii. Return the string-concatenation of *lstr* and *rstr*.
      iv. Set *lval* to *lprim*.
   d. Set *rval* to *rprim*.
2. NOTE: At this point, it must be a numeric operation.
3. Let *lnum* be ? ToNumeric(*lval*).
4. Let *rnum* be ? ToNumeric(*rval*).
5. If Type(*lnum*) is not Type(*rnum*), throw a **TypeError** exception.
6. If *lnum* is a BigInt, then [...]
7. Let *operation* be the abstract operation associated with *opText* and Type(*lnum*) in the following table: [...]
8. Return *operation*(*lnum*, *rnum*).

13.15.3
ApplyStringOrNumericBinaryOperator

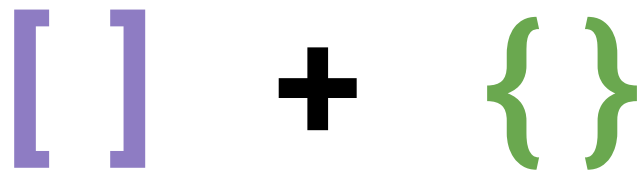
1. Convert *a* and *b* to primitives[1].
2. If *a* or *b* are a string:
    a. Return the string concatenation of *a* and *b*.
3. Convert *a* and *b* to numerics.
4. Return operation(*a* , *b*)

> *a* + *b*

[1] 7.1.1
ToPrimitive

1. If *a* is an object:
   a. Let *result* be valueOf(*a*)
      i. Not an object? Return *result*.
   b. Let *result* be toString(*a*)
      i. Not an object? Return *result*.
2. Return *a*.

> *a*



13.15.3
ApplyStringOrNumericBinaryOperator

1. Convert ***a*** and ***b*** to primitives[1].
2. If ***a*** or ***b*** are a string:
    a. Return the string concatenation of ***a*** and ***b***.
3. Convert ***a*** and ***b*** to numerics.
4. Return operation(***a*** , ***b***).

```
> [] + {}
```

7.1.1
ToPrimitive

1. If *a* is an object:
   a. Let *result* be valueOf(*a*)
      i. Not an object? Return *result*.
   b. Let *result* be toString(*a*)
      i. Not an object? Return *result*.
2. Return *a*.

```
> []

> typeof([])
  "object"
> [].valueOf()
  []
> [].toString()
  ""
```

23.1.3.36
Array.prototype.toString ( )

1. Return Call(join , *a*)

```
> [1,2,3].join()
  "1,2,3"

> [].join()
  ""
```

7.1.1
ToPrimitive

1. If *a* is an object:
   a. Let *result* be valueOf(*a*)
      i. Not an object? Return *result*.
   b. Let *result* be toString(*a*)
      i. Return *result*.
2. Return *a*.

```
> {}

> typeof({})
  "object"
> ({}).valueOf()
  {}
> ({}).toString()
  "[object Object]"
```

20.1.3.6
Object.prototype.toString ( )

1. Use internal methods to determine ***tag***
    a. Otherwise, let ***tag*** be "Object".
2. Return "[object ***tag***]".

```
> ({"a": "b"}).toString()
  "[object Object]"

> JSON.stringify({"a": "b"})
  '{"a":"b"}'
```

13.15.3
ApplyStringOrNumericBinaryOperator

1. Convert *a* and *b* to primitives[1].
2. If *a* or *b* are a string:
   a. Return the string concatenation of *a* and *b*.
3. Convert *a* and *b* to numerics.
4. Return operation(*a* , *b*)

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

> [] + []
> "" + ""
  ""
```

13.15.3 ApplyStringOrNumericBinaryOperator

1. Convert *a* and *b* to primitives[1].
2. If *a* or *b* are a string:
   a. Return the string concatenation of *a* and *b*.
3. Convert *a* and *b* to numerics.
4. Return operation(*a* , *b*)

```
> {} + []
> ?? + ""
  0

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

14.2.2
Runtime Semantics: Evaluation

Block : `{ }`

1. Return empty.

```
> {}
  undefined

> {} + []
>    + []
```

13.5.4
Unary + Operator

1. Return toNumber(*a*)

7.1.4.1.2
StringNumericValue

StringNumericLiteral ::: *StrWhiteSpace*

1. Return 0.

```
> {}
  undefined

> {} + []
>    + []
>    + ""
  0
```

13.5.4
Unary + Operator

1. Return toNumber(*a*)

```
> {}
  undefined

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

7.1.1
ToPrimitive

1. If *a* is an object:
   a. Let *result* be valueOf(*a*)
      i. Not an object? Return *result*.
   b. Let *result* be toString(*a*)
      i. Return *result*.
2. Return *a*.

```
> {}

> typeof({})
  "object"
> ({}).valueOf()
  {}
> ({}).toString()
  "[object Object]"
```

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

```
> [] + []
  ""
> [] + {}
  "[object Object]"
> ({}) + []  # prevent interpreter misparsing
  "[object Object]"
> ({}) + {}
  "[object Object][object Object]"
```

Does Python have this JavaScript peculiarity?

```
>>> [] + []
[]
>>> [] + {}
TypeError: can only concatenate list (not "dict") to list
>>> {} + []
TypeError: unsupported operand type(s) for +: 'dict' and 'list'
>>> {} + {}
TypeError: unsupported operand type(s) for +: 'dict' and 'list'
```

404

Python Standard Library - Built-in Types

Sequence Types — list, tuple, range

Common Sequence Operations

`s + t`

the concatenation of *s* and *t*

```
>>> [] + []
[]

>>> [] + {}
TypeError: can only concatenate list (not "dict") to list
```

```
>>> [] + []
[]

>>> [] + {}
TypeError: can only concatenate list (not "dict") to list
```

CPython 3.13

Objects/listobject.c

```
static PyObject *
list_concat(PyObject *aa, PyObject *bb) {
  if (!PyList_Check(bb)) {
    PyErr_Format(PyExc_TypeError,
      "can only concatenate list (not
          \"%.200s\") to list",
      Py_TYPE(bb)->tp_name);
    return NULL;
  }
  ...
```

Python Language Reference
3.3. Special method names

A class can implement certain operations that are invoked by special syntax [..] by defining methods with special names.

3.3.8. Emulating numeric types

`object.__add__(self, other)`

These methods are called to implement the binary arithmetic operations (+, [..]

```
>>> [] + []
[]
>>> type([])
<class 'list'>
>>> list.__add__([], [])
[]
>>> list.__add__.__doc__
'Return self+value.'
```

Python Language Reference
3.3. Special method names

A class can implement certain operations that are invoked by special syntax [..] by defining methods with special names.

3.3.8. Emulating numeric types

`object.__add__(self, other)`

These methods are called to implement the binary arithmetic operations (`+`, [..]

```
>>> type({})
<class 'dict'>
>>> dict.__add__
AttributeError: type object 'dict' has no attribute '__add__'. Did you mean: '__hash__'?
```

```
HelloWorld {
public static void main(String[] args) {
    System.out.println("Hello world!");
}
```

Java

```
java> Integer a = 127
java> Integer b = 127
java> a == b
true
```

```
java> Integer a = 128
java> Integer b = 128
java> a == b
false
```

```
java> Integer a = 127
java> Integer b = 127
java> a == b
true
```

openjdk/jdk13
src/../java/lang/Integer.java

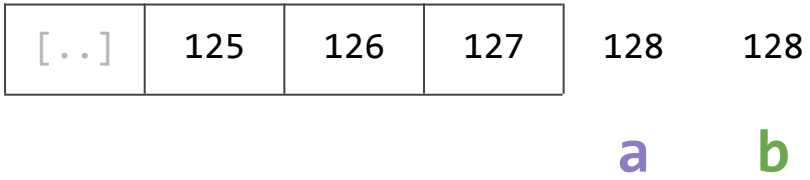
```
private static class IntegerCache {
  static final int low = -128;
  static final int high;
  static final Integer[] cache;
  static {
    // "java.lang.Integer.IntegerCache.high"
    int h = 127;
    [..]
    int size = (high - low) + 1;
    Integer[] c = new Integer[size];
  }
```

ab

```
java> Integer a = 128
java> Integer b = 128
java> a == b
false
```

openjdk/jdk13
src/../java/lang/Integer.java

```
private static class IntegerCache {
  static final int low = -128;
  static final int high;
  static final Integer[] cache;
  static {
    // "java.lang.Integer.IntegerCache.high"
    int h = 127;
    [..]
    int size = (high - low) + 1;
    Integer[] c = new Integer[size];
  }
```



```
java> Integer a = 128
java> Integer b = 128
java> a.equals(b) // Integer equality
true
```

```
java> int a = 128  // use simple type
java> int b = 128
java> a == b // equality
true
```

Does Python have this
Java peculiarity?

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

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

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

```
$ python

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

python/cpython internal/pycore_global_objects.h

```
#define _PY_NSMALLPOSINTS   257
#define _PY_NSMALLNEGINTS   5

struct _Py_static_objects {
  struct {

   /* Small integers are preallocated in
this array so that they can be shared. */

   PyLongObject
        small_ints[_PY_NSMALLNEGINTS
                + _PY_NSMALLPOSINTS];
```

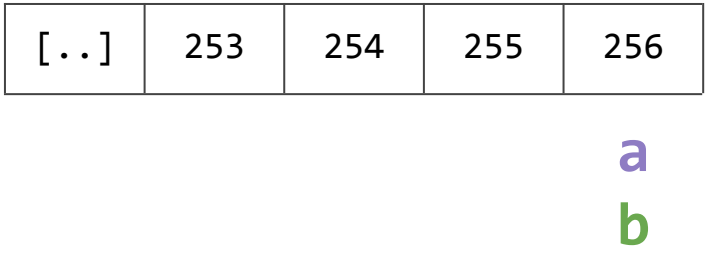
```
$ python

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

python/cpython internal/pycore_global_objects.h

```
#define _PY_NSMALLPOSINTS  257
#define _PY_NSMALLNEGINTS  5

PyLongObject small_ints[_PY_NSMALLNEGINTS
                        + _PY_NSMALLPOSINTS];
```



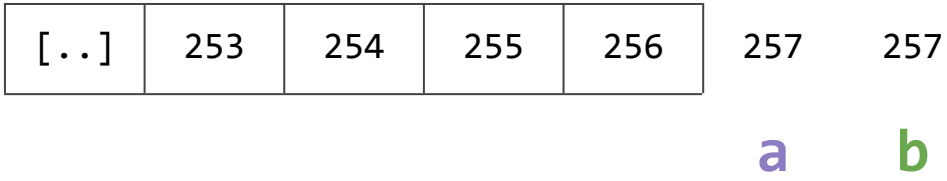
```
$ python

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

python/cpython internal/pycore_global_objects.h

```
#define _PY_NSMALLPOSINTS  257
#define _PY_NSMALLNEGINTS  5

PyLongObject small_ints[_PY_NSMALLNEGINTS
                        + _PY_NSMALLPOSINTS];
```



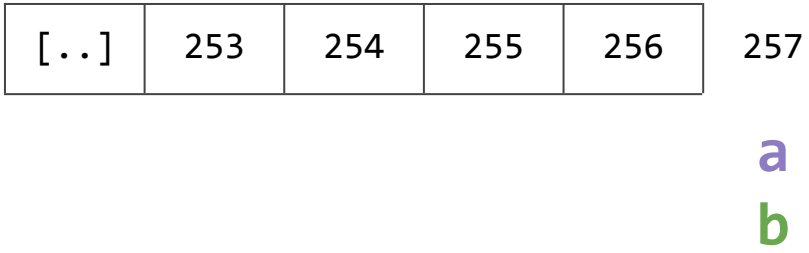
```
$ python

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

python/cpython
internal/pycore_global_objects.h

```
#define _PY_NSMALLPOSINTS  257
#define _PY_NSMALLNEGINTS  5

PyLongObject small_ints[_PY_NSMALLNEGINTS
                        + _PY_NSMALLPOSINTS];
```



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

```
>>> a = 257
>>> b = 257
>>> a == b // FIX: avoid using "is" in Python
True       //      when "==" works.
```

```
#!/usr/bin/perl
print("Hello world!\n");
```

Perl

```
> print "match" if "a" == "b"
  match
```

```
> print "match" if "a" == "b"
  match
```

Perldoc 5.38.2
Equality Operators

Binary "==" returns true if the left argument is **numerically** equal to the right argument.

```
> print "match" if "a" eq "b"
# Use string equality for strings in Perl
```

```
#!/bin/bash
echo Hello world\!
```

Bash

```
~ $ if [[ "a" -eq "b" ]];
     then echo "match"
   fi
  "match"
```

```
~ $ if [[ "a" -eq "b" ]]
      then echo "match"
    fi
  "match"
```

Bash Reference Manual

6.4 Bash Conditional Expressions

`-eq -nq`

arithmetic binary operators

```
~ $ if [[ "a" == "b" ]]
      then echo "match"
    fi      // non-numeric
```

Bash Reference Manual

6.4 Bash Conditional Expressions

`-eq -nq`

arithmetic binary operators

6.5 Shell Arithmetic

The operators and their precedence, associativity, and values are the same as in the C language.

`== !=`

equality and inequality

[..]

Does Python have this
Perl and Bash peculiarity?

==

__eq__

```
>>> object.__eq__.__doc__
'Return self==value.'
```

Python Language Reference

3.3. Special method names

By default, `object` implements `__eq__()` by using `is`.

6.10.3. Identity comparisons

The operators `is` and `is not` test for an object's identity: `x is y` is true if and only if *x* and *y* are the same object. An Object's identity is determined using the `id()` function.

```
#!/usr/bin/env python2
print "Hello world!"
```

Python 2

```
>>> 4 < 2
False
>>> 4 > 2
True
>>> 4 > '2'
False
>>> '4' > 2
True
```

```
>>> 4 < [4]
True
>>> 4 < [4] < '4'
True
>>> 4 < [4] < '4' < (4,)
True
```

```
>>> 4 < [4]
True
>>> 4 < [4] < '4'
True
>>> 4 < [4] < '4' < (4,)
True
```

Python 2.7 Standard Library

Built-in Types: Comparisons

Objects of different types, except different numeric types and different string types, never compare equal; such objects are ordered consistently but arbitrarily (so that sorting a heterogeneous array yields a consistent result).

CPython implementation detail:
Objects of different types except numbers are ordered by their type names

```
# avoid Python 2
>>> type(4)
'int'
>>> type([4])
'list'
>>> type('4')
'string'
>>> type((4,))
'tuple'
```

Python 2.7 Standard Library

Built-in Types: Comparisons

Objects of different types, except different numeric types and different string types, never compare equal; such objects are ordered consistently but arbitrarily (so that sorting a heterogeneous array yields a consistent result).

CPython implementation detail:
Objects of different types except numbers are ordered by their type names

```
>>> 4 < [4] < '4'
TypeError: '<' not supported between instances of 'int' and 'list'
```

Python 3.13 Standard Library

Built-in Types: Comparisons

Objects of different types, except different numeric types, never compare equal.

Python 3.13 Tutorial: Data Structures

Comparing Sequences and Other Types

[..] Rather than providing an arbitrary ordering, the interpreter will raise a `TypeError` exception.

```
#!/usr/bin/env elixir
IO.puts("Hello world")
```

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'
```

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

36
49
64
81
100
```

```
iex> Enum.map(65..90, fn(x) -> x end)
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
// ASCII is cool!
```

```
iex> Enum.map(6..10, fn(x)
-> x*x end)
'$1@Qd'
iex> a = Enum.map(6..10,
fn(x) -> x*x end)
iex> Enum.map(a, fn(x) ->
IO.puts x end)
36
49
64
81
100
```

Inspect.Opts

Defines the options used by the `Inspect` protocol.

The following fields are available:

`:charlists` - When the default `:infer`, **the list will be printed as a charlist if it is printable**, otherwise as list. See `List.ascii_printable?/1` to learn when a charlist is printable.

List.ascii_printable?

A printable charlist in Elixir contains only the printable characters in the standard seven-bit ASCII character encoding, which are characters ranging from **32 to 126 in decimal notation**

Does Python have this Elixir peculiarity?

```
>>> import datetime
>>> now = datetime.datetime.now()
>>> print(now)
2025-02-02 12:15:32.012345
>>> now
datetime.datetime(2025, 2, 2, 12, 15, 32, 12345)
```

```
>>> import datetime
>>> now = datetime.datetime.now()
>>> print(now.__str__)  // or str(now)
2025-02-02 12:15:32.012345
>>> repr(now)           // or repr(now)
datetime.datetime(2025, 2, 2, 12, 15, 32, 12345)
```

```
#!/usr/bin/env pwsh
'Hello world!' | Write-Host
```

PowerShell

```
PS> if (36 > 42) { "true" } else { "false" }
false

PS> if (36 < 42) { "true" } else { "false" }
The '<' operator is reserved for future use.

PS> cat 42
36
```

```
PS> if (36 > 42) { "true" }
else { "false" }

false
```

about_Redirection
Potential confusion with comparison operators

The `>` operator isn't to be confused with the Greater-than comparison operator (often denoted as `>` in other programming languages).

Depending on the objects being compared, the output using `>` can appear to be correct (because 36 isn't greater than 42).

```
PS> if (36 > 42) { "true" } else { "false" }
false
```

```
PS> if (2 -gt 4) { "true" } else { "false" }
false

PS> if (2 -lt 4) { "true" } else { "false" }
true      # use actual comparison operators
```

Does Python have this PowerShell peculiarity?

```
>>> print("True") if 36 > 42 else print("False")
False
>>> print("True") if 36 < 42 else print("False")
True
>>> import os; os.listdir()
[]
```

```
>>> a = {"a": 1}
>>> b = {"b": 2}
>>> a | b
{'a': 1, 'b': 2}
// PEP 584 - Python 3.9+
```

```
>>> from pathlib import Path
>>> p = Path("mydir")
>>> "/tmp" / p / "file.txt"
PosixPath('/tmp/mydir/file.txt')
>>> "data" > p
TypeError: '>' not supported between
instances of 'str' and 'PosixPath'
... yet.
```

Learn something ✨new✨

Thanks!

Sources and demos: **glasnt.com/wat**