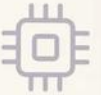


which values can be changed in place, and what every operator actually does to them



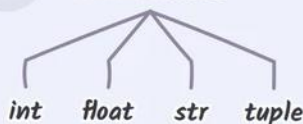
Mutable vs immutable, the core idea

- **mutable** means a value's contents can change in place, without creating a new object
- **immutable** means once created, that value can never be changed, only replaced by a fresh one
- `id()` reports an object's identity, its address in memory, which is the real test of mutation vs a new object

Immutable types vs Mutable types

Immutable

Immutable



four of the immutable types branching from one root, each one replaced rather than changed

- int, float, bool, str, complex, tuple
- any change to these makes a brand new object, the old one is untouched

Mutable

Mutable



the three mutable types, all changeable in place without a new `id()`

- list, dict, set
- these can be changed in place, same object, same id, new contents

`id()` and rebinding vs mutation

`x = [1, 2, 3]` → `x` has some id, say 5000

`x.append(4)` → `x` is now `[1, 2, 3, 4]`, id is still 5000, the same list was mutated

`x = x + [5]` → `id()` changes here, a new list was built and `x` was rebound to it

`s = 'cat'` → `s.upper()` returns 'CAT' as a new object, `s` itself is still 'cat', strings can't mutate at all

same id after the change means it was mutated, a new id means it was rebound

Arithmetic operators, exactly as Python evaluates them

<code>7 / 2</code> , true division	3.5
<code>7 // 2</code> , floor division	3
<code>-7 // 2</code> , not -3	-4
<code>7 % 2</code> , the remainder	1
<code>-7 % 2</code>	1
2 to the power 10	$2^{10} = 1024$

true division / always returns a float, even for `4 / 2`; floor division and modulus both floor toward negative infinity, and the modulus result's sign follows the divisor

Why the distinction actually matters *

- passing a mutable object into a function lets that function change it, and the caller sees the change
- an immutable object passed the same way can never be altered by the function, only a new value can be returned
- among the built-in types, only the immutable ones can be dictionary keys or set members, and a tuple qualifies only if everything inside it is immutable too

