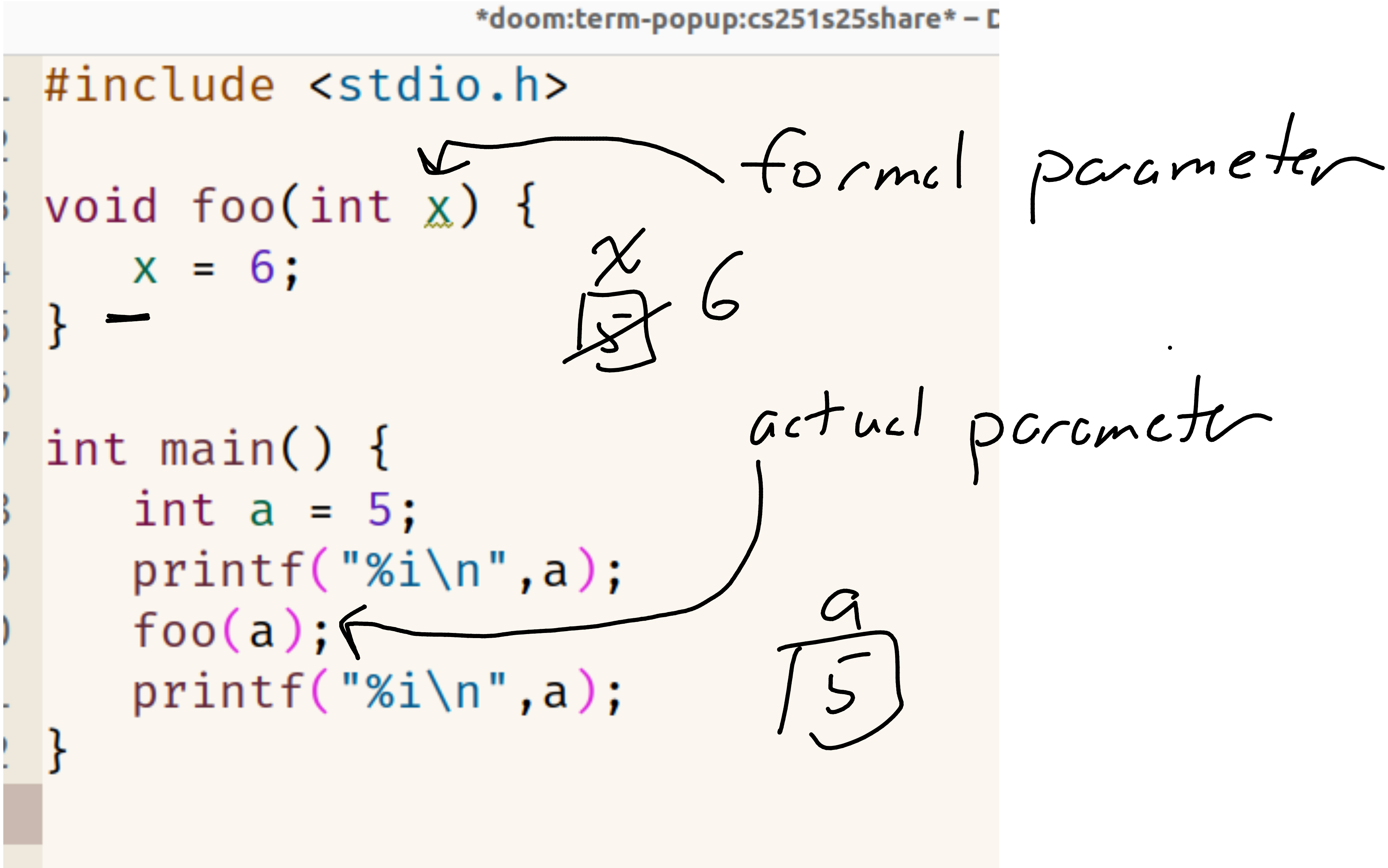
Parameter passing

What does C actually do?



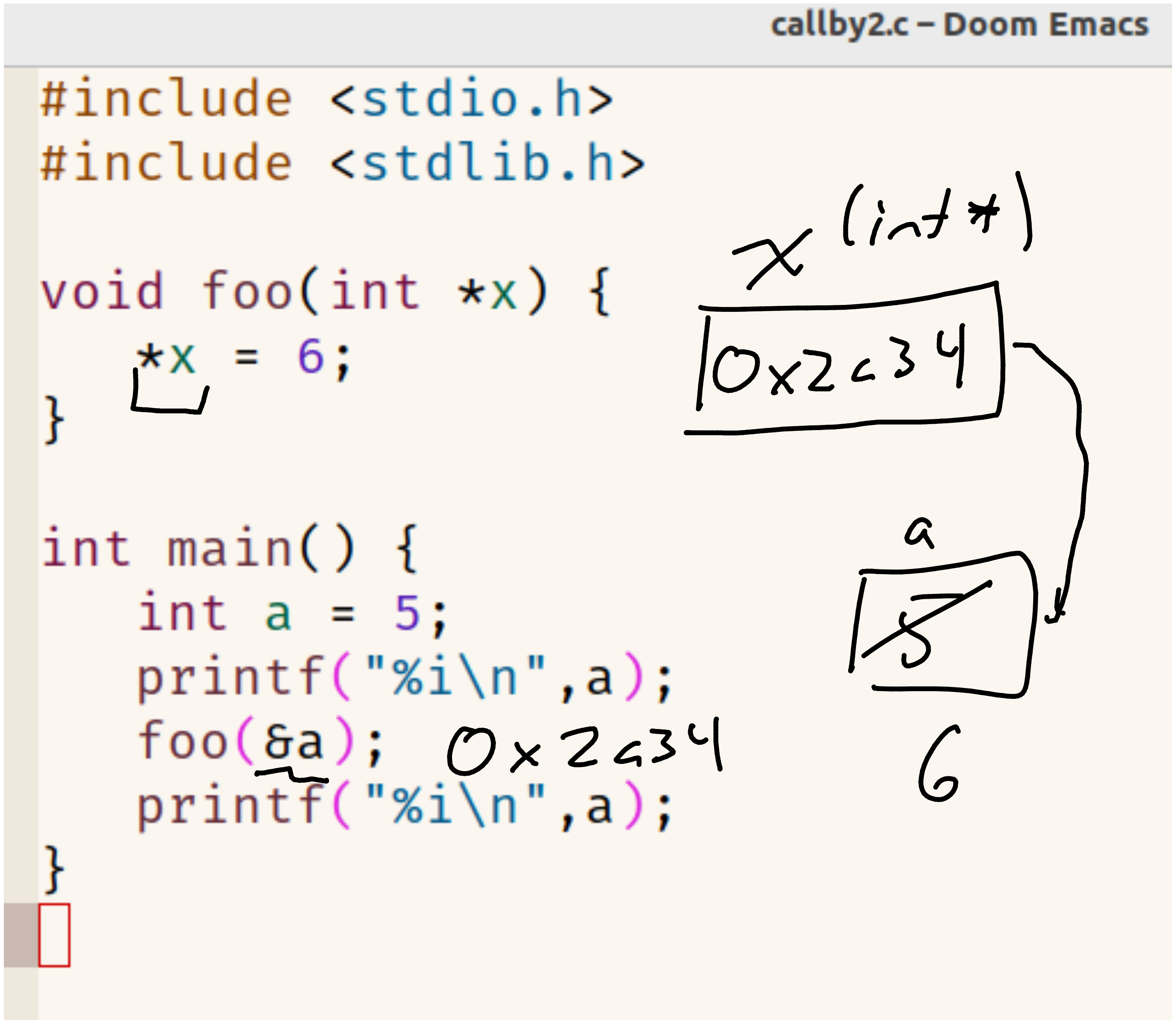
```c
#include <stdio.h>

void foo(int x) {
    x = 6;
}

int main() {
    int a = 5;
    printf("%i\n",a);
    foo(a);
    printf("%i\n",a);
}
```

??

When calling a function, it evaluates the actual parameter, then it copies that value to the formal parameter



```
#include <stdio.h>
#include <stdlib.h>

void foo(int *x) {
    *x = 6;
}

int main() {
    int a = 5;
    printf("%i\n",a);
    foo(&a);
    printf("%i\n",a);
}
```

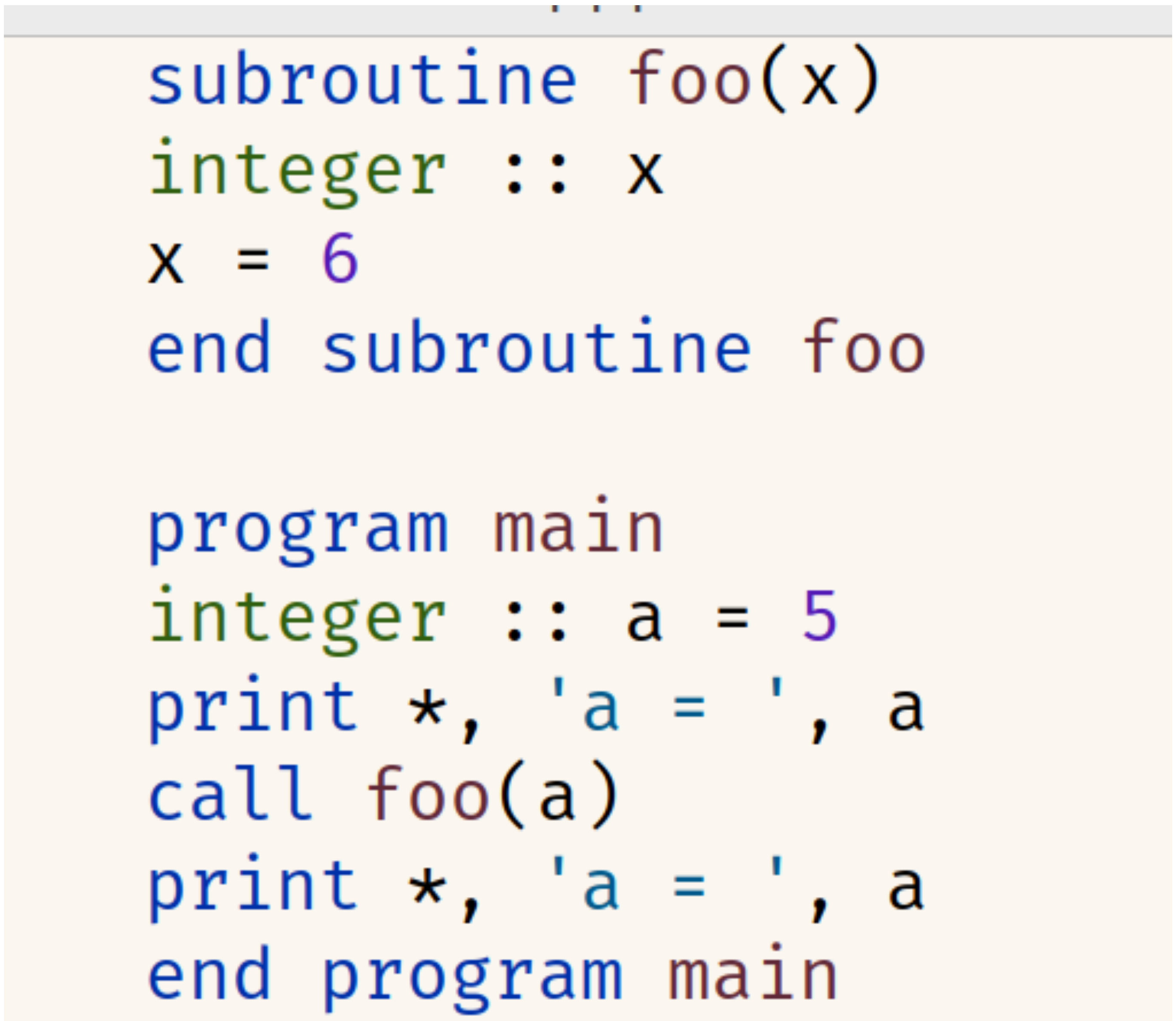
C does the same thing.

It evals the actual parameter, then it copies it

Now let's look at a language that does something different

(C is a "value model" language "copying")

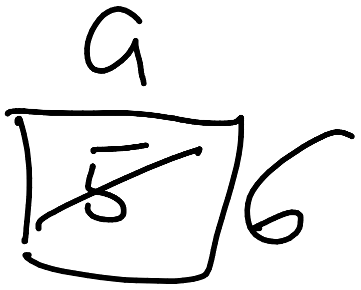
- Let's pick another lang like that



```
subroutine foo(x)
integer :: x
x = 6
end subroutine foo

program main
integer :: a = 5
print *, 'a = ', a
call foo(a)
print *, 'a = ', a
end program main
```

x is a new name for a



Fortran implements parameters entirely differently.

In Fortran, it doesn't eval the actual parameter.

Instead, the formal parameter becomes an alias (alternative name) for the actual parameter.

It does *not* *copy* anything.

Terminology (sigh).

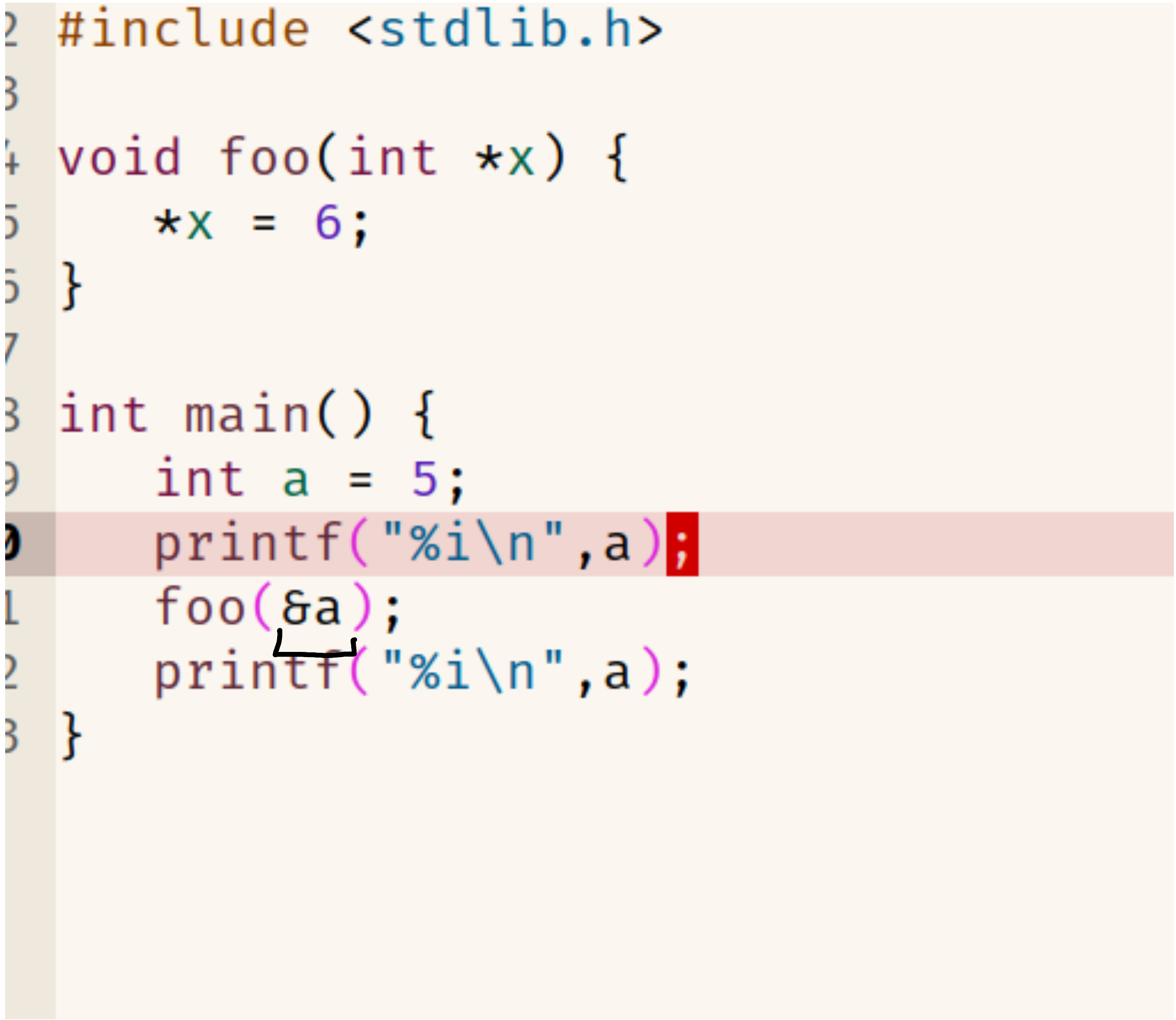
C version (eval, then copy)

- pass by value.
  (call by)

Fortran vesion (aliases)

- pass by reference
  (call by)

<u>Pain.</u>



```c
#include <stdlib.h>

void foo(int *x) {
    *x = 6;
}

int main() {
    int a = 5;
    printf("%i\n",a);
    foo(&a);
    printf("%i\n",a);
}
```

This is C.

It eval'ed the actual parameter (&a), then it copied it.

This behavior is called, historically, pass by value.

And yet, the confused masses who don't know things, try to call this pass by reference.

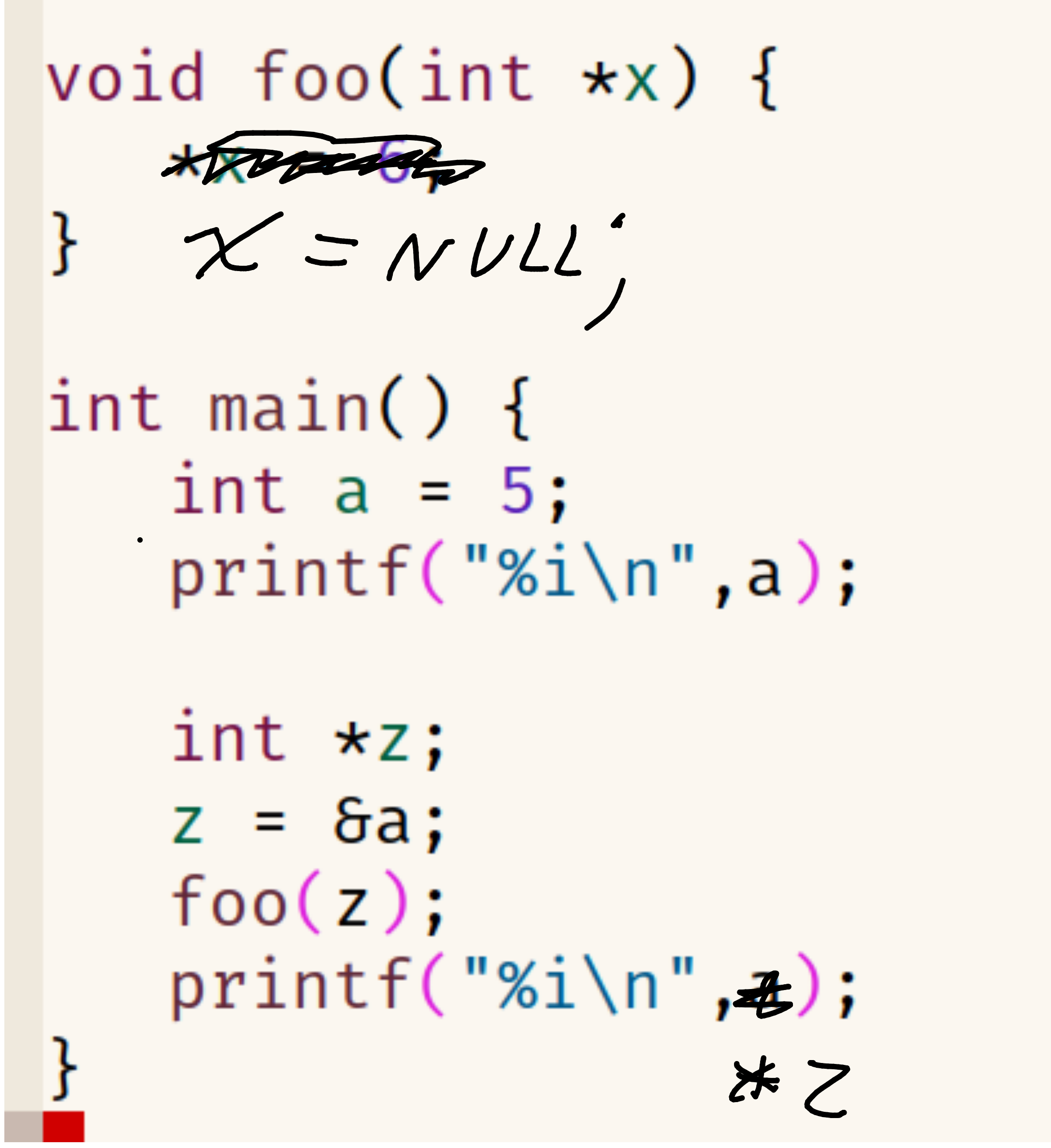
① Why do people make this mistake?

- we're using the word reference w/ two different meanings
  - ① reference = pointer
  - ② pass by reference = pass by aliasing

② Why does it matter?

These are implemented entirely differently inside the implementation. Do we really not care if we don't have accurate terminology?

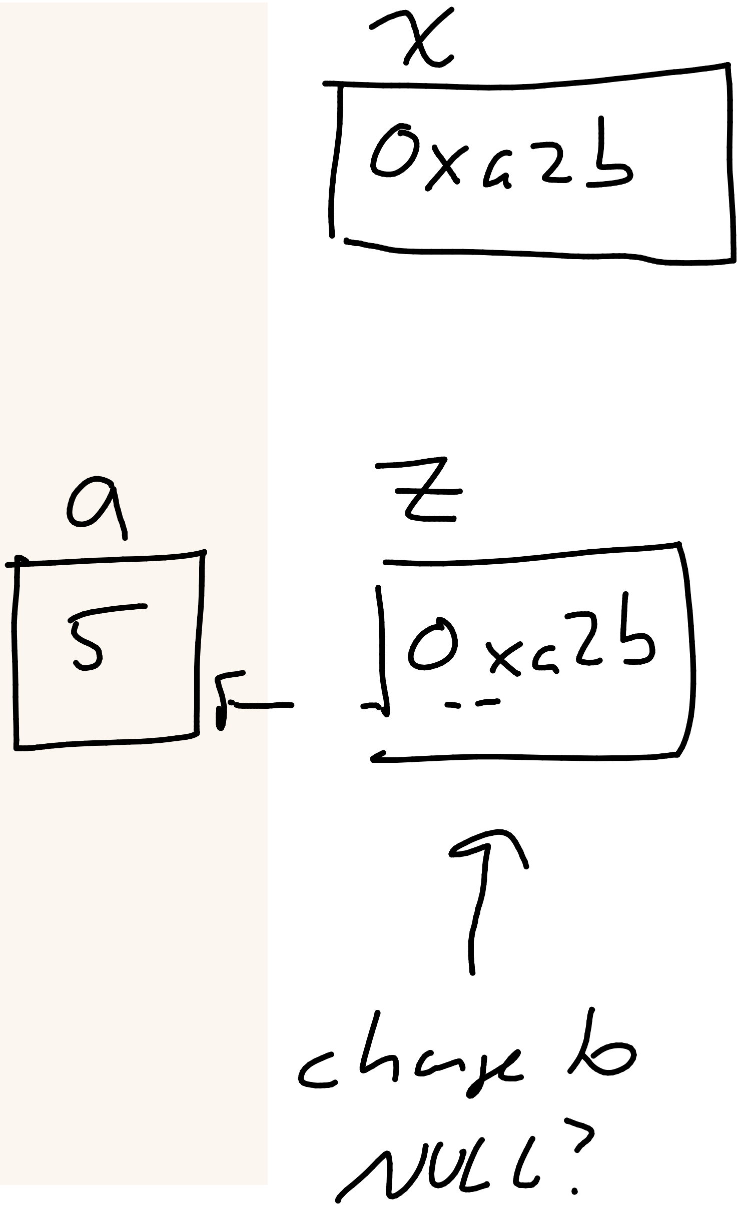
- Important to understand how language works



```
void foo(int *x) {
    ~~*x = 6;~~
}   x = NULL;

int main() {
    int a = 5;
    printf("%i\n",a);

    int *z;
    z = &a;
    foo(z);
    printf("%i\n",~~z~~);
}                 *z
```



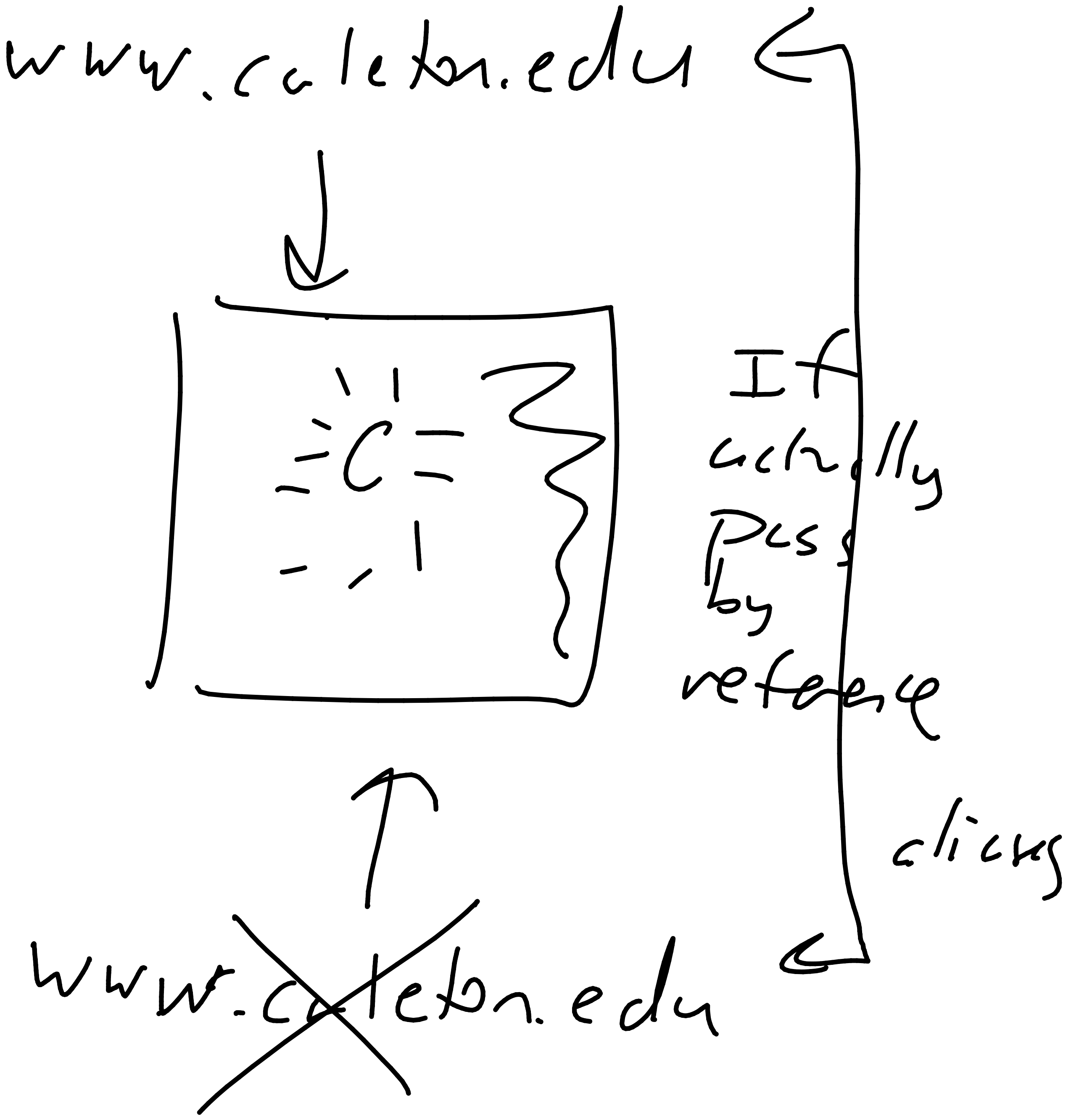
Does x=NULL; also change z?

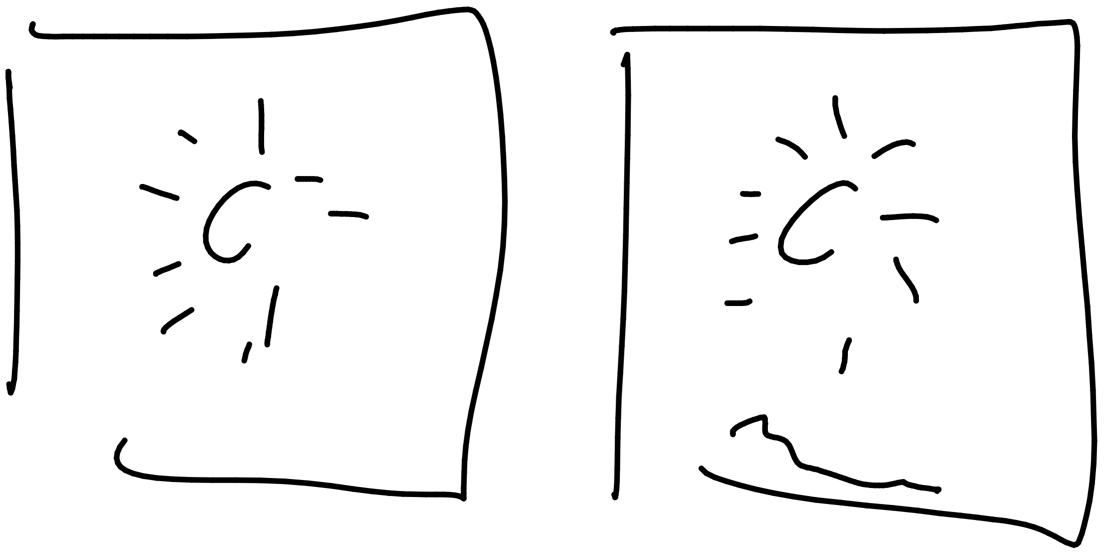
① yes

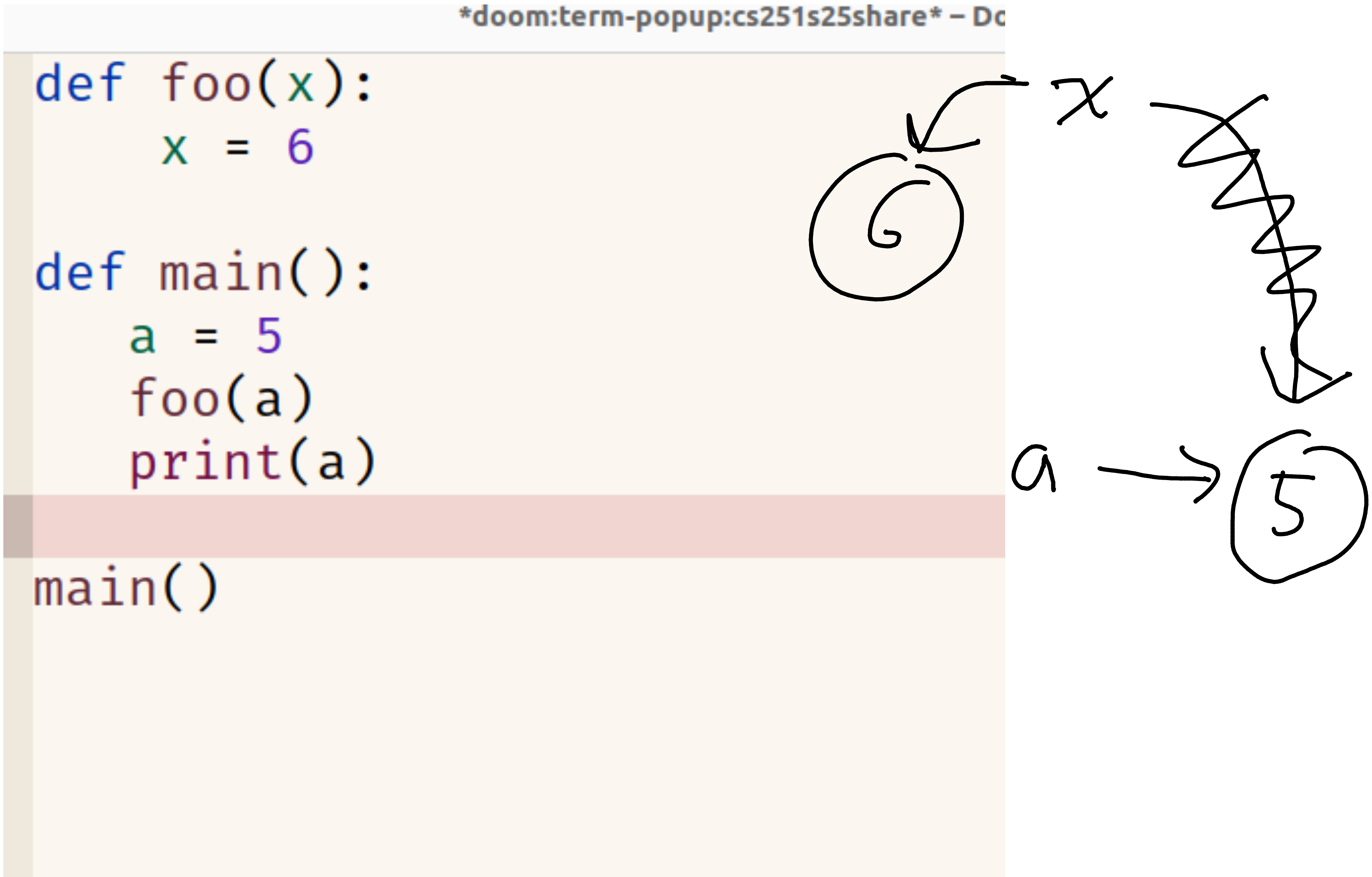
[✓ ② no]

But if C did actual honest pass by reference (which it doesn't), the answer would have been yes

Old SO


www.caleton.edu
If actually pass by reference
clicks
www.caleton.edu





```python
def foo(x):
    x = 6

def main():
   a = 5
   foo(a)
   print(a)

main()
```

Python is a "sharing" language

- variables are always implicitly associated with objects.

What does Python do?

- it evals actual parameter to set association
- it copies the association to the formal parameter

Is this more like

① pass by value?

② --- reference?

③ entirely different?

One thing this isn't is pass by reference. If it was, I would have printed a 6.

What is it?

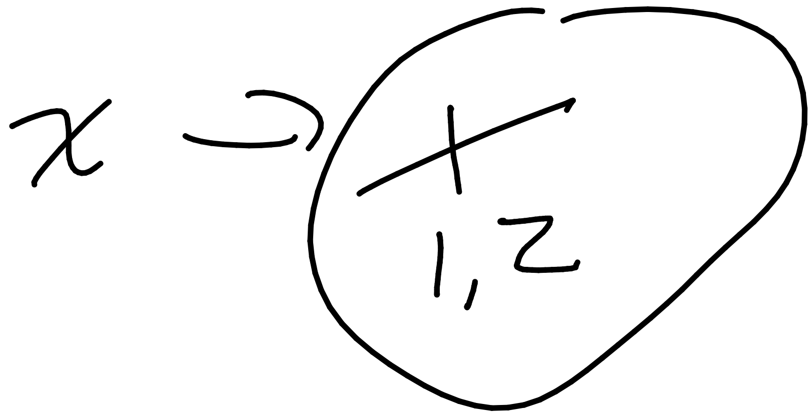
Some say this is pass by value, and what I copied was the association (pointer)

Some say this is something else - most commonly called pass by sharing (a few others)

What is a mutable vs immutable object?

In Python, a list is a mutable object.

```
x = [1]
x.append(2)
```



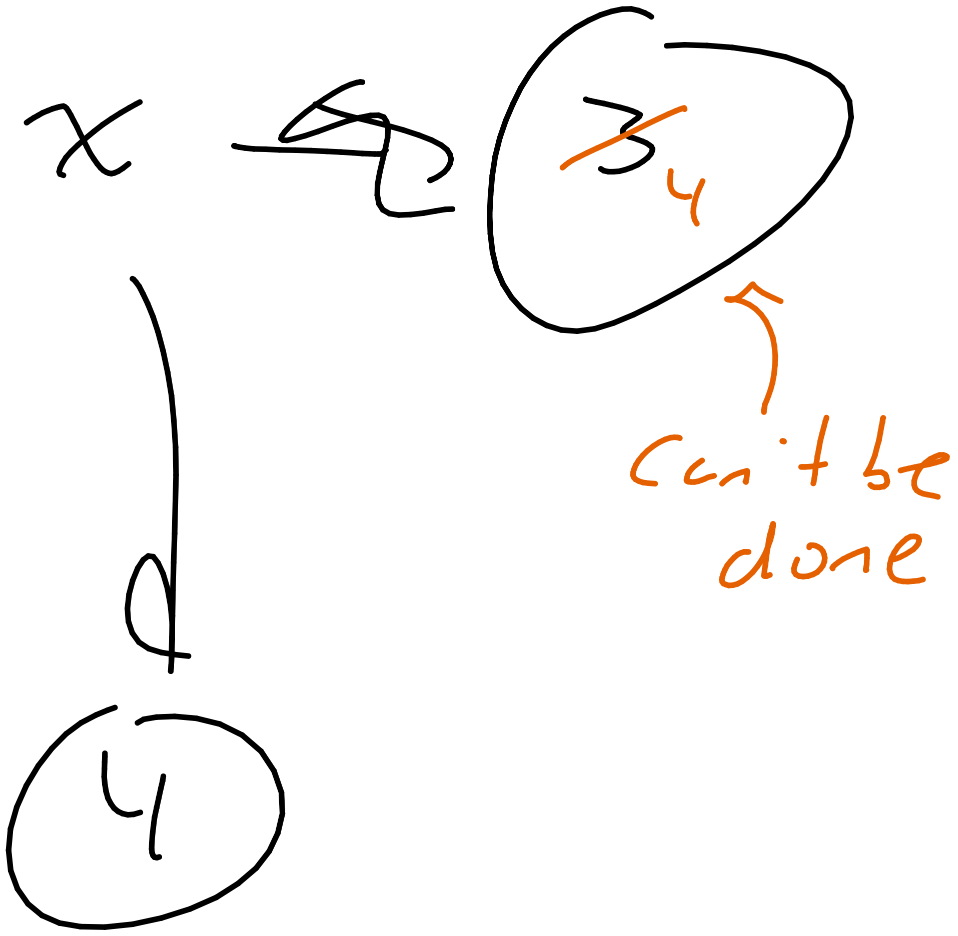
integers in Python are immutable

```
x = 3
```

There is no

```
x.add(1)
```

```
x = 4
```



Massive conception you'll find

"Passing immutable objects works as pass by value, and passing mutable objects works as pass by reference."