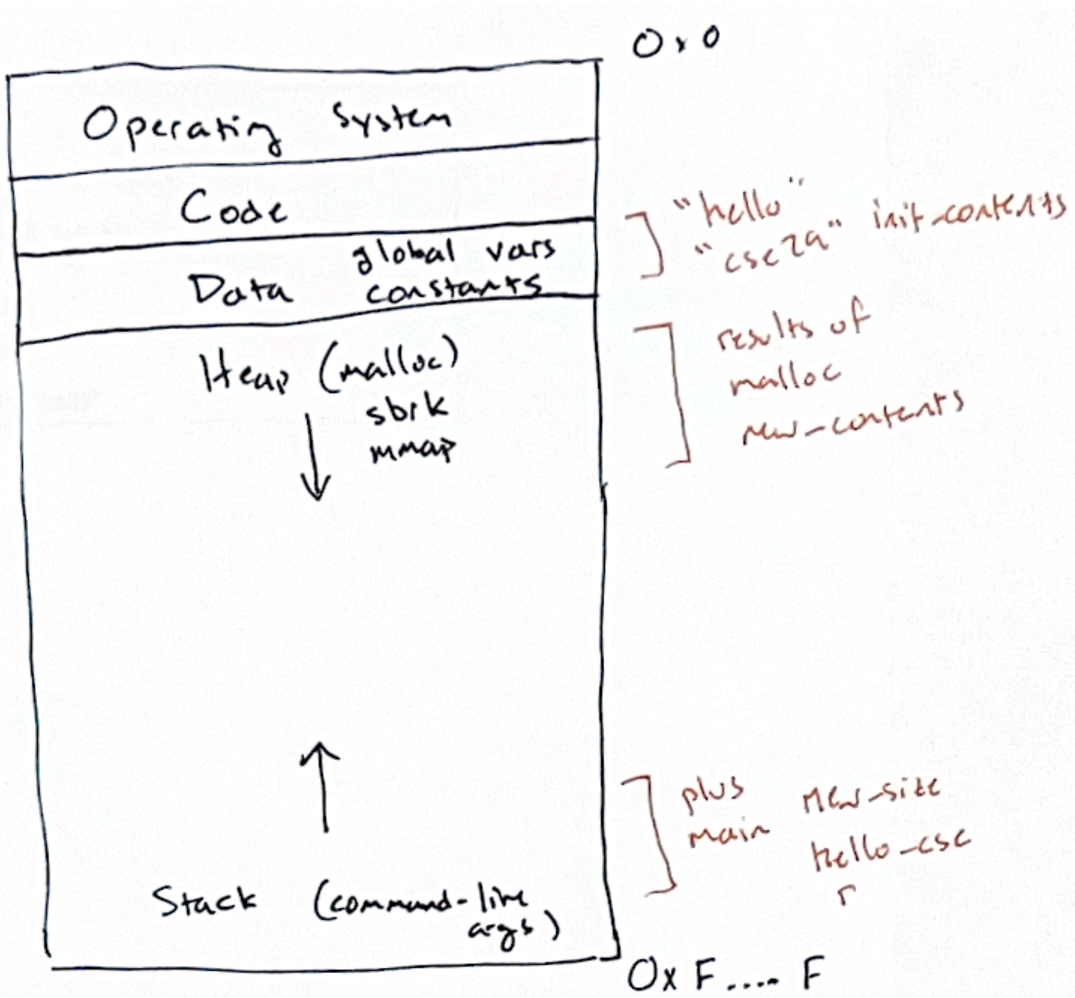
In contrast to C, languages like Python and Java (and others) have *immutable* strings that support operations like concatenation with +. How does that work in the machine?

```
$ python3
>>> a = "hello "
>>> b = "cse29"
>>> c = a + b
>>> c
'hello cse29'
>>> a
'hello '
>>> b
'cse29'
```

```c
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

struct String {
    uint64_t length; // should always equal strlen(contents)
    char* contents;  // should always have allocated space of length + 1
};

typedef struct String String;

String new_String(char* init_contents) {
    uint64_t size = strlen(init_contents);
    String r = { size, init_contents };
    return r;
}

String plus(String s1, String s2) {
    uint64_t new_size = s1.length + s2.length + 1;
    char new_contents[new_size];    malloc(new_size)
    strncpy(new_contents, s1.contents, s1.length);
    strncpy(new_contents + s1.length, s2.contents, s2.length);
    new_contents[new_size - 1] = 0;
    String r = { new_size - 1, new_contents };
    return r;
}

int main() {
    String s = new_String("hello");
    printf("%s\n", s.contents);

    String s2 = new_String("cse29");

    String hello_cse = plus(s, s2);
    String hello_bang = plus(s, new_String("!!!!"));

    printf("%s\n", hello_cse.contents);
    printf("%s\n", hello_bang.contents);
}
```



Obs: our stack addresses have been 0x7FF...

```c
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

struct String {
    uint64_t length;
    char* contents;
};

typedef struct String String;

String new_String(char* init_contents) {
    uint64_t size = strlen(init_contents);
    char* contents = malloc(size + 1);
    strcpy(contents, init_contents);
    String r = { size, contents };
    return r;
}

// This plus is *just* the heap-allocating version now
String plus(String s1, String s2) {
    uint64_t new_size = s1.length + s2.length + 1;
    char* new_contents = calloc(new_size, sizeof(char));
    strncpy(new_contents, s1.contents, s1.length);
    strncpy(new_contents + s1.length, s2.contents, s2.length);
    new_contents[new_size - 1] = 0;
    String r = { new_size - 1, new_contents };
    return r;
}

String join(String strs[], int count, String delimiter) {
    String s = new_String("");
    for(int i = 0; i < count; i += 1) {
        s = plus(s, strs[i]);
        if(i < count - 1) {
            s = plus(s, delimiter);
        }
    }
    return s;
}

int main() {
    String apple = new_String("apple");
    String banana = new_String("banana");
    String strawberry = new_String("strawberry");
    String fruit[] = { apple, banana, strawberry };

    String comma = new_String(", ");
    String fruitlist = join(fruit, 3, comma);
    printf("%s\n", fruitlist.contents);
}
```

how many allocations? what is in each of them for character data

focusing on these plus calls

Memory leak
nothing refers to this anymore!

6 bytes: apple\0

8 bytes: apple, ␣\0

14 byte: apple, ␣banana\0

16 byte: apple, ␣banana, ␣\0

26 byte: apple, ␣banana, ␣strawberry\0

↑

fruitlist.contents

free(ptr)

ptr must have been malloc'd

free tells C runtime that

malloc can use this space again