"Static" memory requirements

- Local variables in a fn definition
    - int, float
    - arrays (fixed size, uniform type)
    - parameters

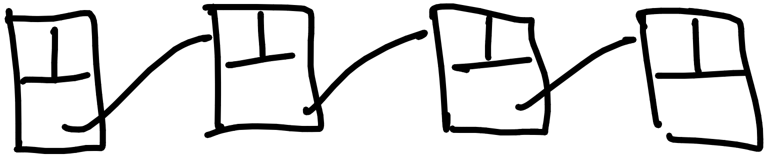
Fixed storage required for each invocation of function

Each function call creates a "frame"

- Frames are stacked
- Point to calling frame – restore on exit

"Dynamic" requirements

List made up of nodes



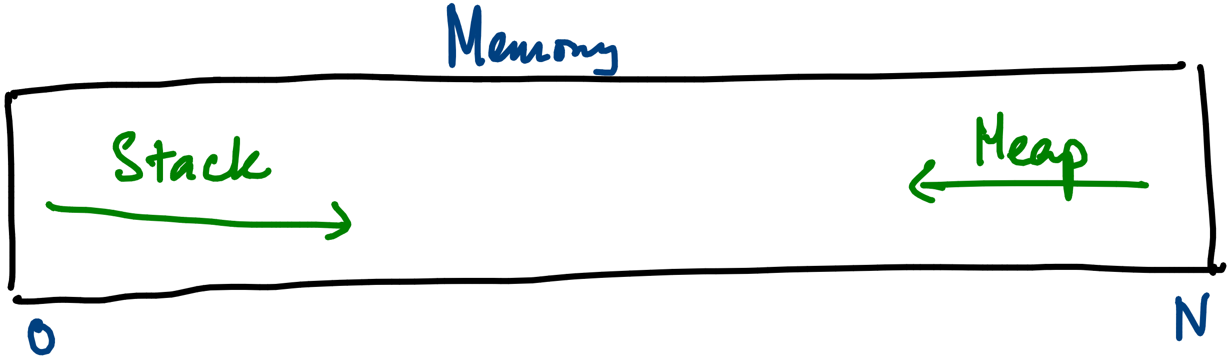
insert(v)

- create a new node
- node should persist after fn returns
  - cannot be in the "frame" for insert() on stack

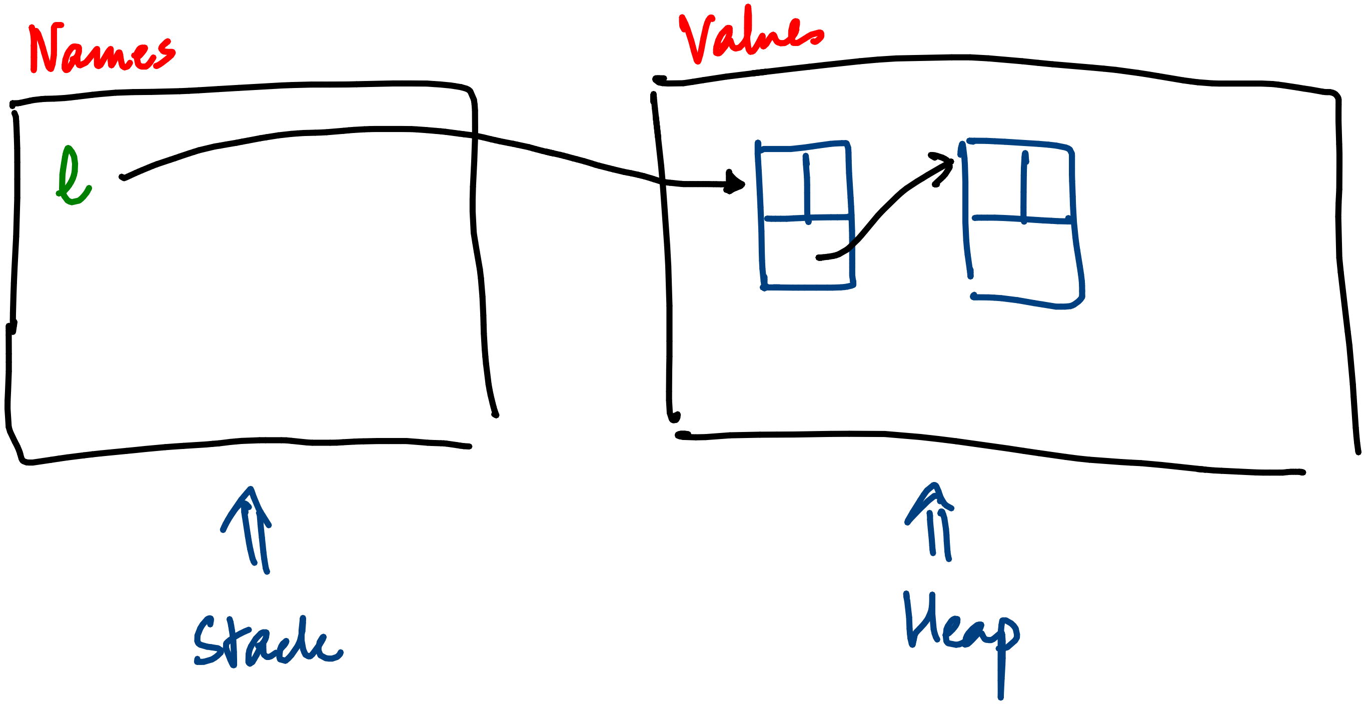
Need an alternate storage space for persistent data (whose extent is not known in advance)

"Heap" storage

Unstructured



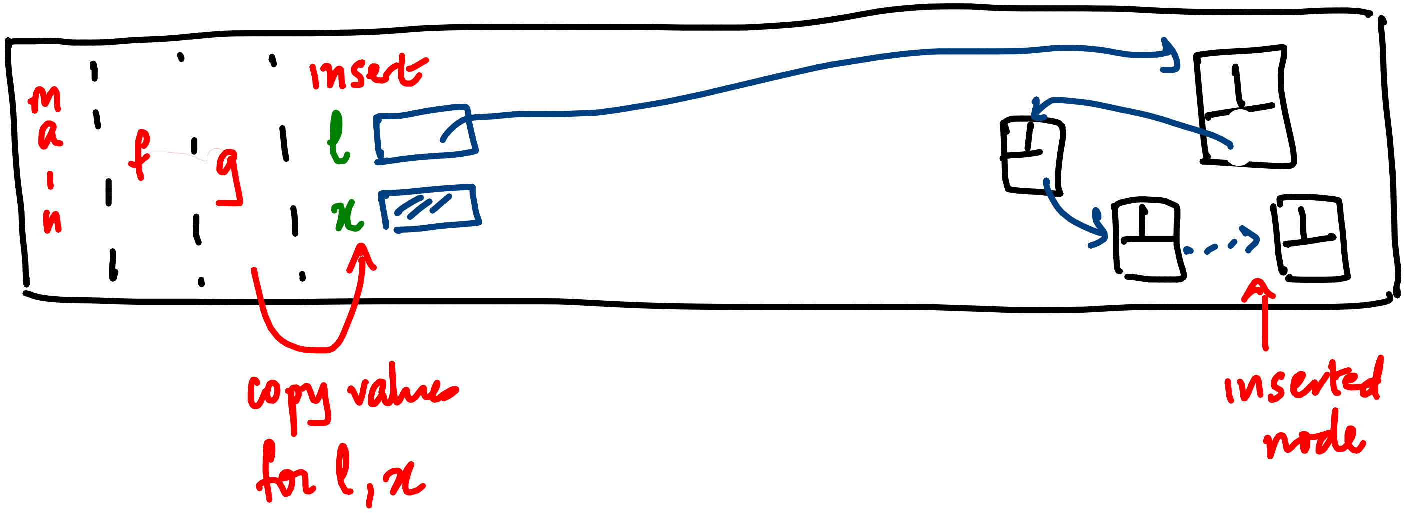
# Python memory model



int i, j; } Directly store values on stack
float g;

node l; – stores an "address" on stack

insert (node l, int x) {. . .}



After insert( ) ends

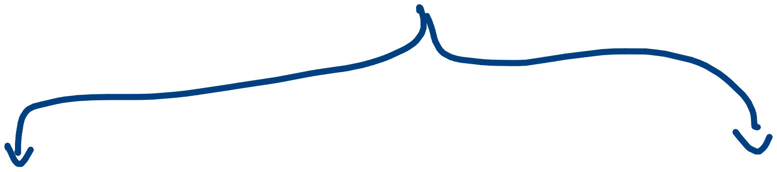
g still has a link to first list item on heap, and hence to newly added node

If insert( ) created a new node using a local name

```
insert(node l, int x) {
    node n;        ← on stack
      :
    n = new(node(..))
                 ↖ on heap
      :
      :
← exits, n is lost
```

Unreachable heap storage = "garbage"

e.g. delete a node in a list/tree



Python

Java

⋮

automatic garbage collection

C

explicitly "free" unwanted memory

Inverse operation of new()

otherwise "memory leak"

- space is blocked

In Java etc

new( — )

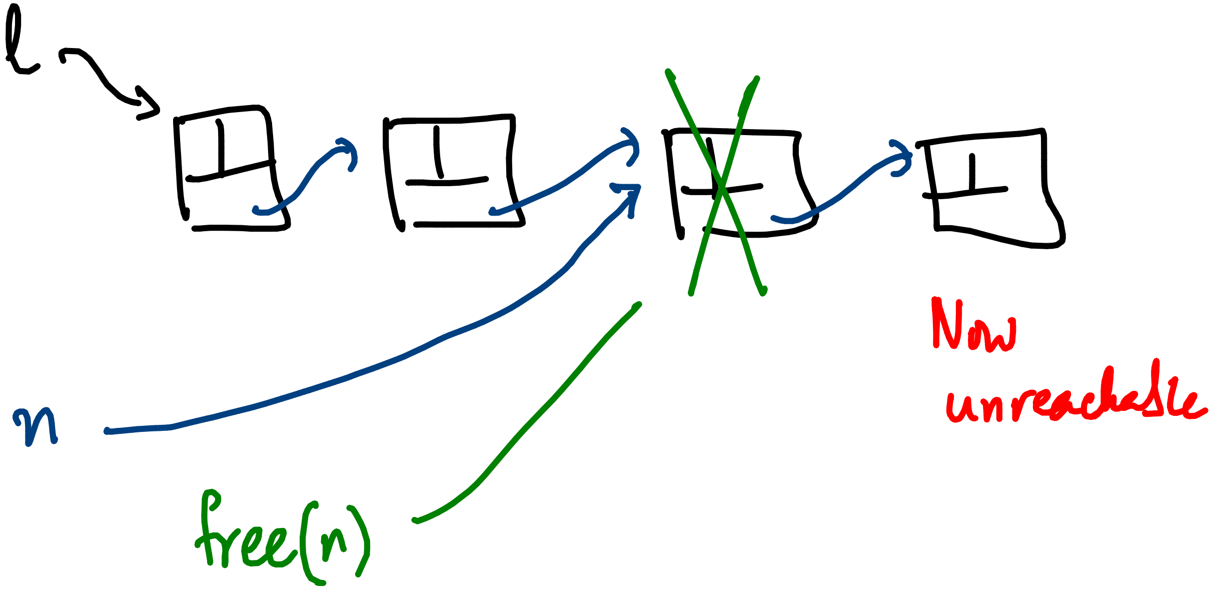
class defined earlier

allocates space as required by class definition

l = new(ListNode(..)) constructor argument

⋮

free(l) [if explicit release of unused storage is needed]



Need to be very careful when releasing space explicitly, not to accidentally create more garbage

In C, the call to new(..) does not convey type

"malloc" — ask for a block of memory of a fixed size, but not "typed"

explicitly convert type to what you want

(Like Python s = str(x))

# Passing parameters to functions

By value - copy value - e.g. int

By reference - copy an address

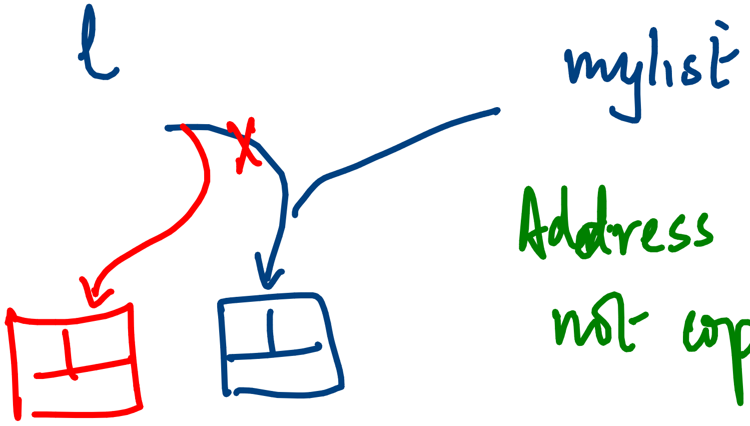
## By value

```
fact(int n) {          x = 7;
    :
    :                  y = fact(x);
   n = n-1;                      :
    :                            :
    :
}
```

will not affect x

```
f(list l, int x){
   :
   l = new(list(..))
   :
```

```
:
f(mylist, myx);
:
```



Address in l is not copied back!

How to write a function to read an int?

```
read (int x) {

    x = int (input(..))

}
```

read(z)

z is copied to x

but not vice versa

C has operators &x - address of x

*p - value pointed to by p