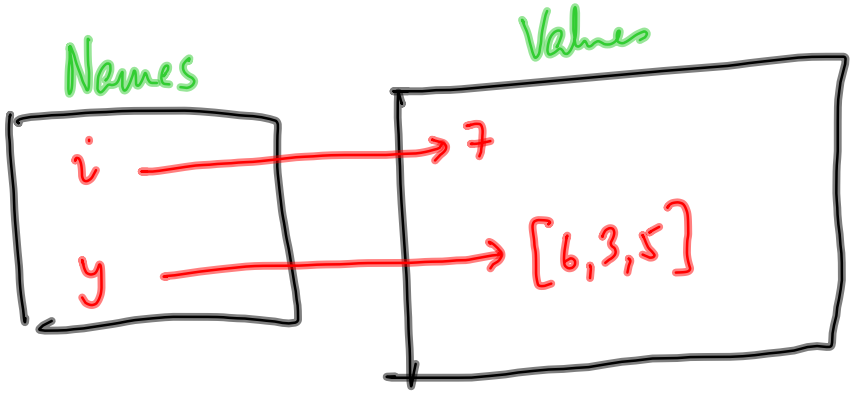
# Introduction to C

| Haskell → | Python → | C |
|---|---|---|
| Declarative | Imperative | Imperative |
| "What to compute" | "How to compute" | Explicit storage management |
| | Storage management is implicit | |

# Python vs C

## Storage Model




Python: names don't have any type associated
type is derived from value

Dynamic typing  i = y  i is now a list

In C:

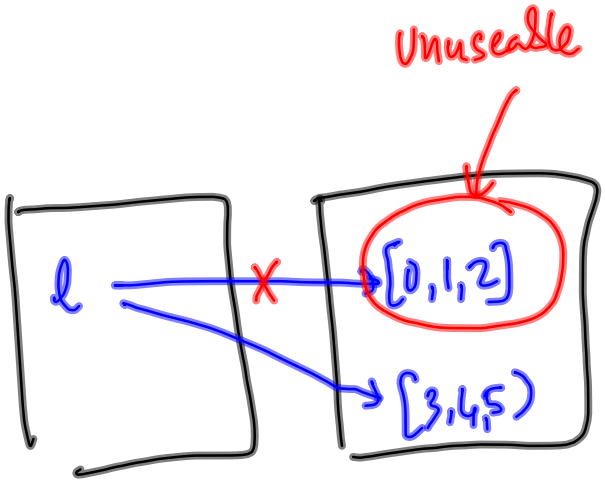
static typing – every name (variable) has a fixed type throughout the program

"Declare" all names/variables in advance, with type information

# Garbage collection

## In Python:

```
l = [0,1,2]
l = [3,4,5]
```




## In C

No automatic reclamation of free space

C is a compiled language

Compilers vs Interpreters

Source program

⇓

Object program (in "machine language")

Haskell ghc

Another program that "understands" the source PL

Python

Haskell – hugs

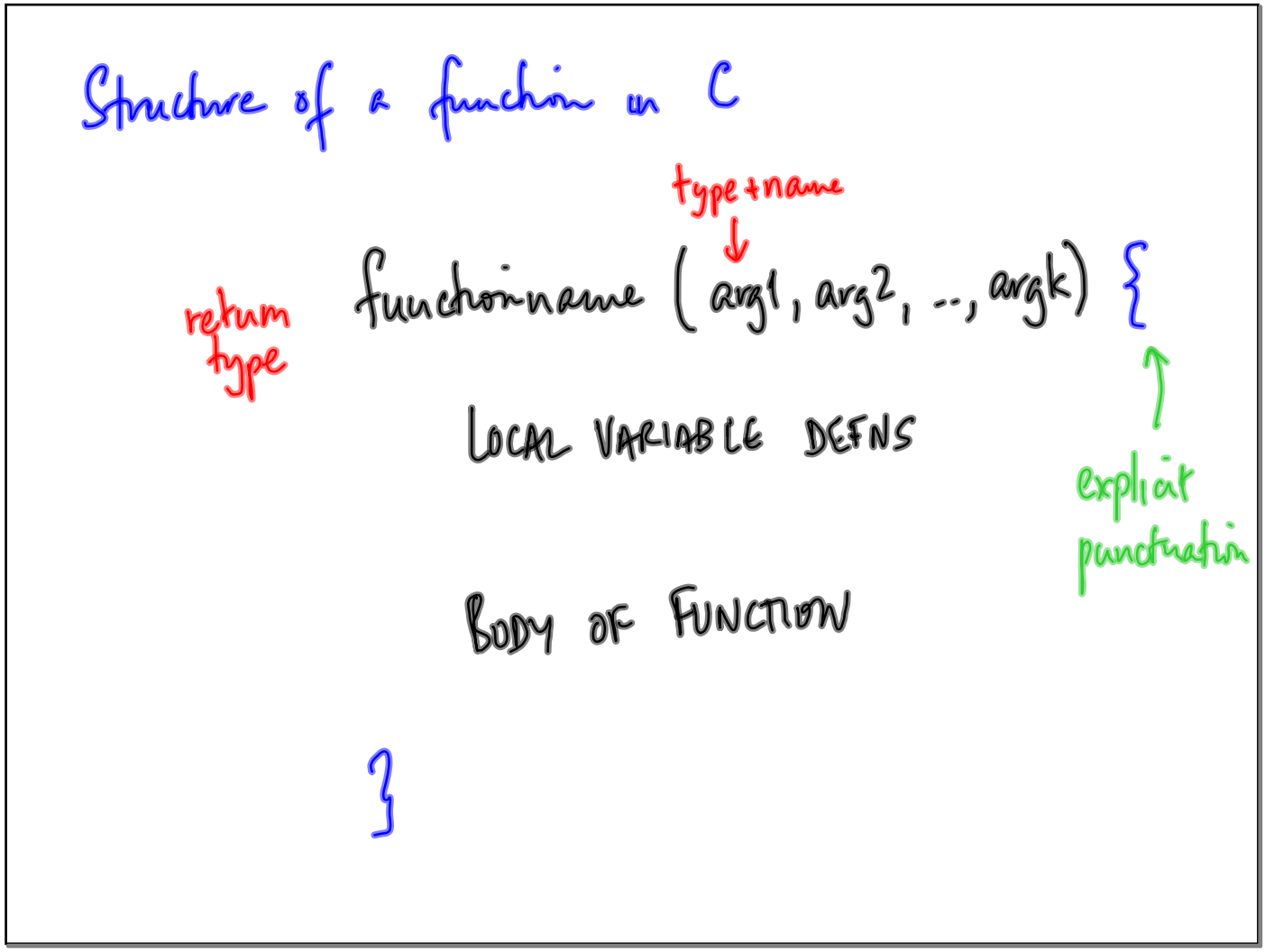
ghci

C program

Collection of functions like Haskell

```
f()
g()
:
```

main() ← execution starts here


Structure of a function in C
type+name
return type
functionname ( arg1, arg2, .., argk) {
LOCAL VARIABLE DEFNS
explicit punctuation
BODY OF FUNCTION
}

Example

```
int plus(int x, int y) {
    int temp;
    temp = x+y;
    return(temp);
}
```

Built in types:

- int ← range of values is system dependent
- float ← range of values is system dependent
- char

No explicit bool type

0 is false

≠0 is true

## Declaration

type variablename;

```
int temp;
float x;
char c;
```

extend to

type var1, var2 ..., varN;

```
int i,j,k,l;
```

↓

but not when describing fn parameters

Special type void - no values - use as return type for fns with no useful return val

# Statements

## Assignment

variable = expression ;

Updates the value stored in the location allocated to variable

```
x = 7;
y = (x = 7);
y = x = 7;
```

→ returns a value! like

An assignment stmt is an expn. x=7 returns 7 which in turn is assigned to y

# Conditional statements

```
if (conditional expression)
    statement;
else
    statement;
```

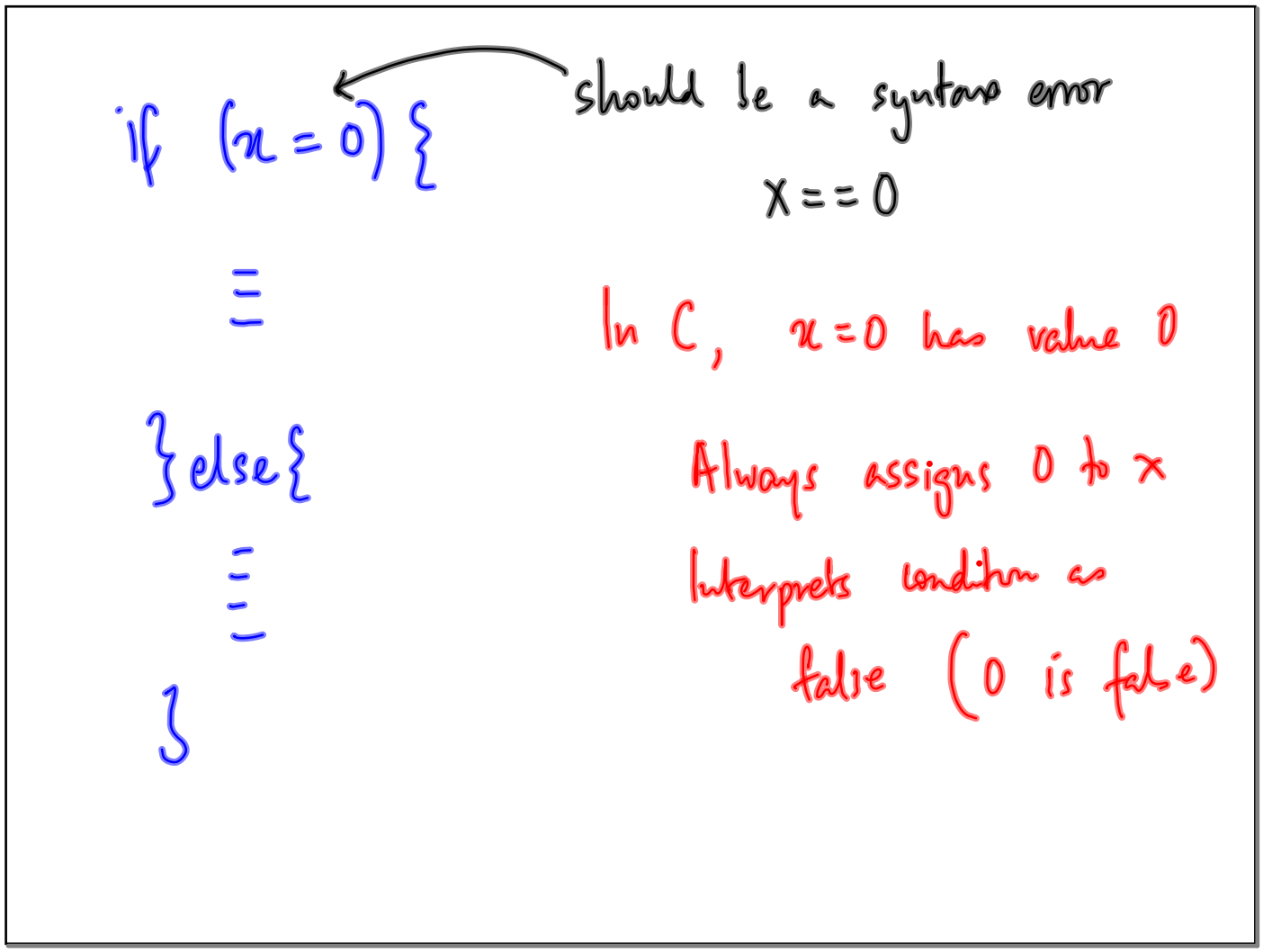
optional

```
if (x < 0) {
    x = -x;
}
```

```
if (x < 0) {
    x = -x;
    sign = sign+1;
}
```

ALWAYS USE { } outside if →

← add to group statements


if (x = 0) {
should be a syntax error
x == 0
=
In C, x = 0 has value 0
}else{
Always assigns 0 to x
=
Interprets condition as
false (0 is false)
}

## Loops

```
while (condition) {
    statement;
    =
    =
    =
}
```

**break** - exits loop

**continue** - returns to top of loop

**Be careful**

```
while (x = 2) {
    =
    =
    =
}
```

Infinite loop

x = 2 always true

Iteration

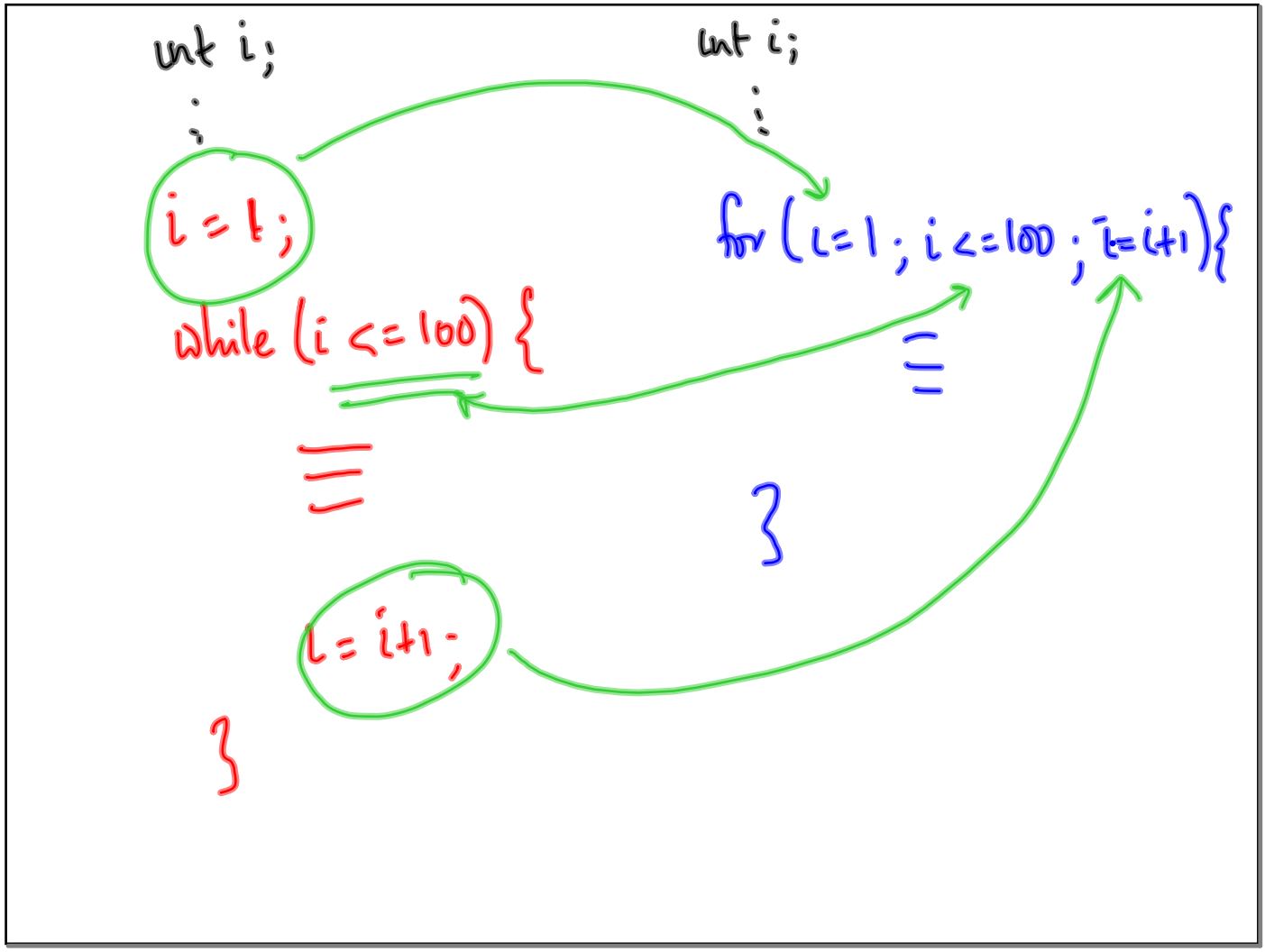
Do something starting with 1 upto 100 in increment of 1

```
for ( i = 1 ; i <= 100 ; i = i+1 ) {
}
```

initialization

termination condition

increment


int i;
int i;
i = 1;
for (i=1; i<=100; i=i+1){
while (i <= 100) {
}
i = i+1;
}

For loop is quite general

```
for ( i=0, j=0 ; i<m && j<n ; i=i+1, j=j+1){
```

≡ multiple initializations

multiple increments

```
}
```

x = x/2 is a valid update for increment

Implicitly // or div if both args are int

Type casting

(float) x

Convert & treat x as float in this expression

```
float z;
int x,y;

z = ((float)x)/y;
```