Types

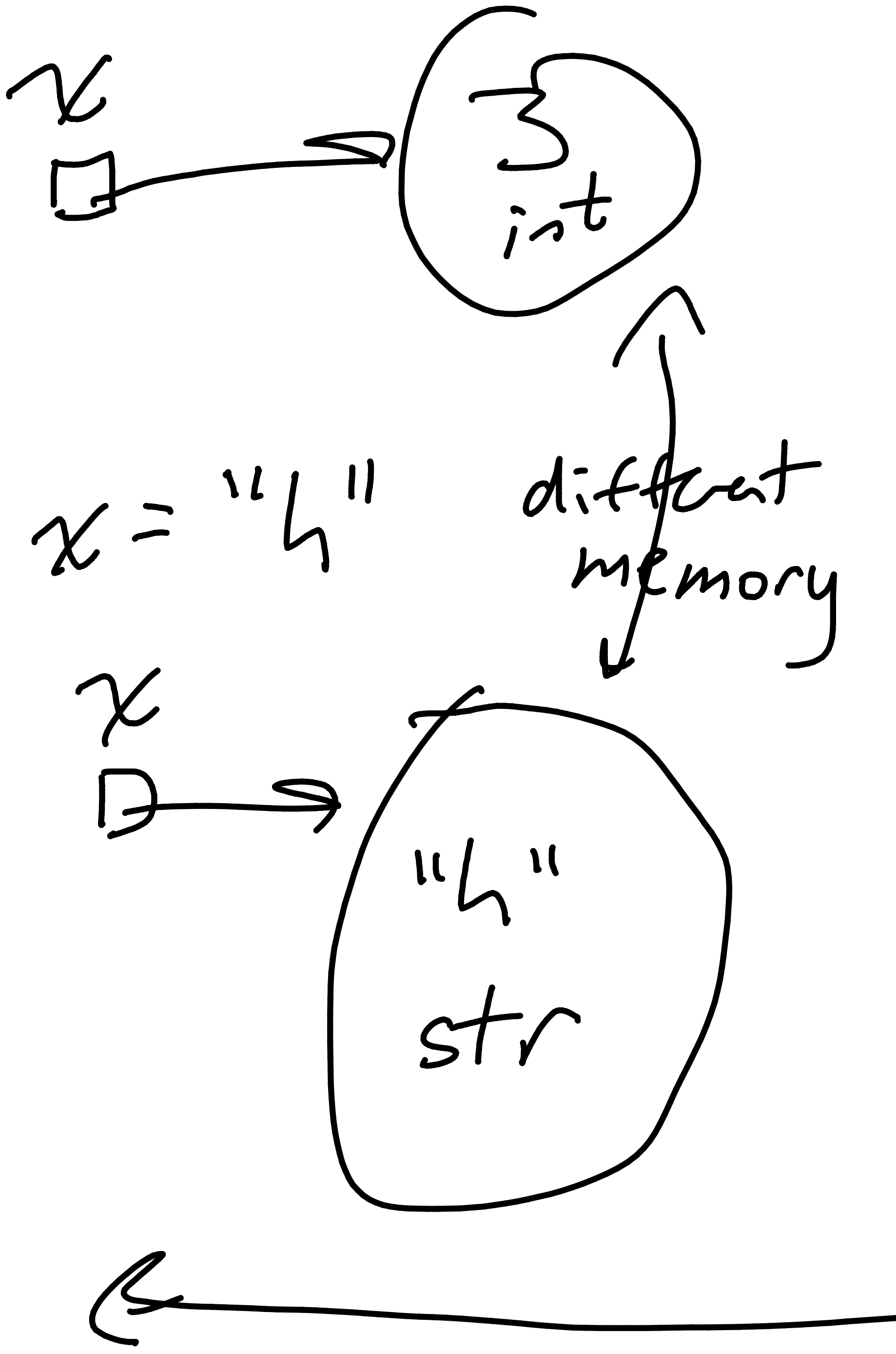
Tail Call optimization

Java

```
int add(int x, int y) {
      ...
}

String add(String x,
           String y) {
       ...
}

  ... add(3, 5)

   .. add("ab", "cd")
```

In Python

x = 3



x = "h"

1. Imagine a programming language without types (int, string, whatever). What challenges might emerge, either for the user of the language, or for the implementer?

special casing/redundant code
string /number mayhem

...

2. We know a type when we see it, but what is it? Try to construct a definition for "type”.

data; set of possible values that belong together
-operations (+? len?)
- storage represatation (#bytes, etc)
-attributes (#of dims, naming convetions)

3. Some language require types to be declared (like in C, `int x;`). There are many advantages of this. Think through why declarations are useful for each of the following, compared to a language like Python that does not require declarations of types.

   (a) Minimizing the amount of memory used for storing data

   Know in advance how much memory you need, so just allocate that

   (b) Function overloading (multiple functions with same names, but different parameter types)

   (c) Type checking (verifying that you are using types correctly, e.g. not subtracting a string from an integer)

Java X

x - y
int String
	int x;
	String y;
	x-y

4. Languages need to do type checking, which means making sure that every operation is correct according to the rules of the operation itself and the types that it uses. Consider these two examples (first Python, second Java), both of which should give an error:

```
x = 3
print('hi')
if True == True:
    print(len(x))
```

"dynamic type-checking" (looking for typing errors)

```
class Typething {
    public static void main(String[] args) {
        int x = 3;
        System.out.println("hi");
        if (true == true) {
            System.out.println(x.length());
        }
    }
}
```

"static type-checking"

(a) Both programs give an error at the len/length call, because integers don't have a length. Python prints "hi" before hitting the error, whereas the Java program won't compile. What's different about the two languages regarding the timing of these checks? What are the relative pros and cons?

Python: can run part of program before hitting error

-more flexible

→ but, if error is rare, might never know until some user finds it someday

Java: know about error immediately

static type-checking catches more errors before the program is run, doesn't depend on potatially rare conditions

dynamic - lets the program stat running, and will only error when it hits the error.

- more flexible

In general, static typechecking helps to produce more bug-free code

---

"Strongly typed" "Weakly typed"

poorly defined. but ....

What are they getting at? *["strongly typed" vs weakly

① Type safety - how much does the language stop you from ~~being dumb~~? making mistakes you wish you hadn't?

Examples:

C:
```
int *x = 3;
```
C will (w/ prodding?) let you do this nonsense

Python:
```
x = 3
y = 3.0
if x == y:
```
bad idea? if types are different

better? some would say
```
if x == round(y)
```
and very strict type safety would prevent comparing ints and floats

② Static typechecking or dynamic

③ Type inferencing

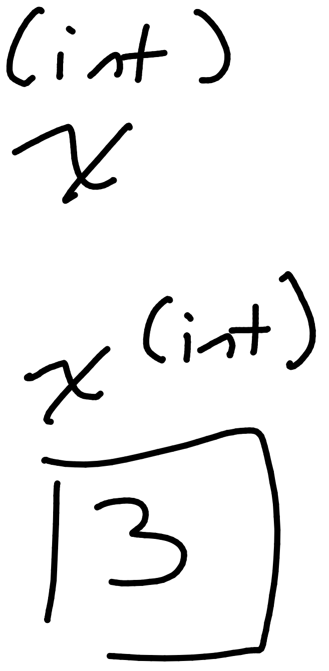
new Java allows

```
var x = 3;      int
var y = x;      int
var z = new Scanner(System.in);
```

Scanner

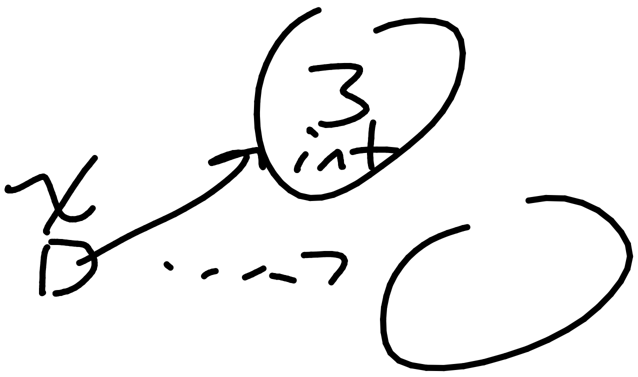
Java

```
int x;

var x = 3;
```



Python

```
if daytime:
    x = 3
else
    x = "hello"
```



line 7

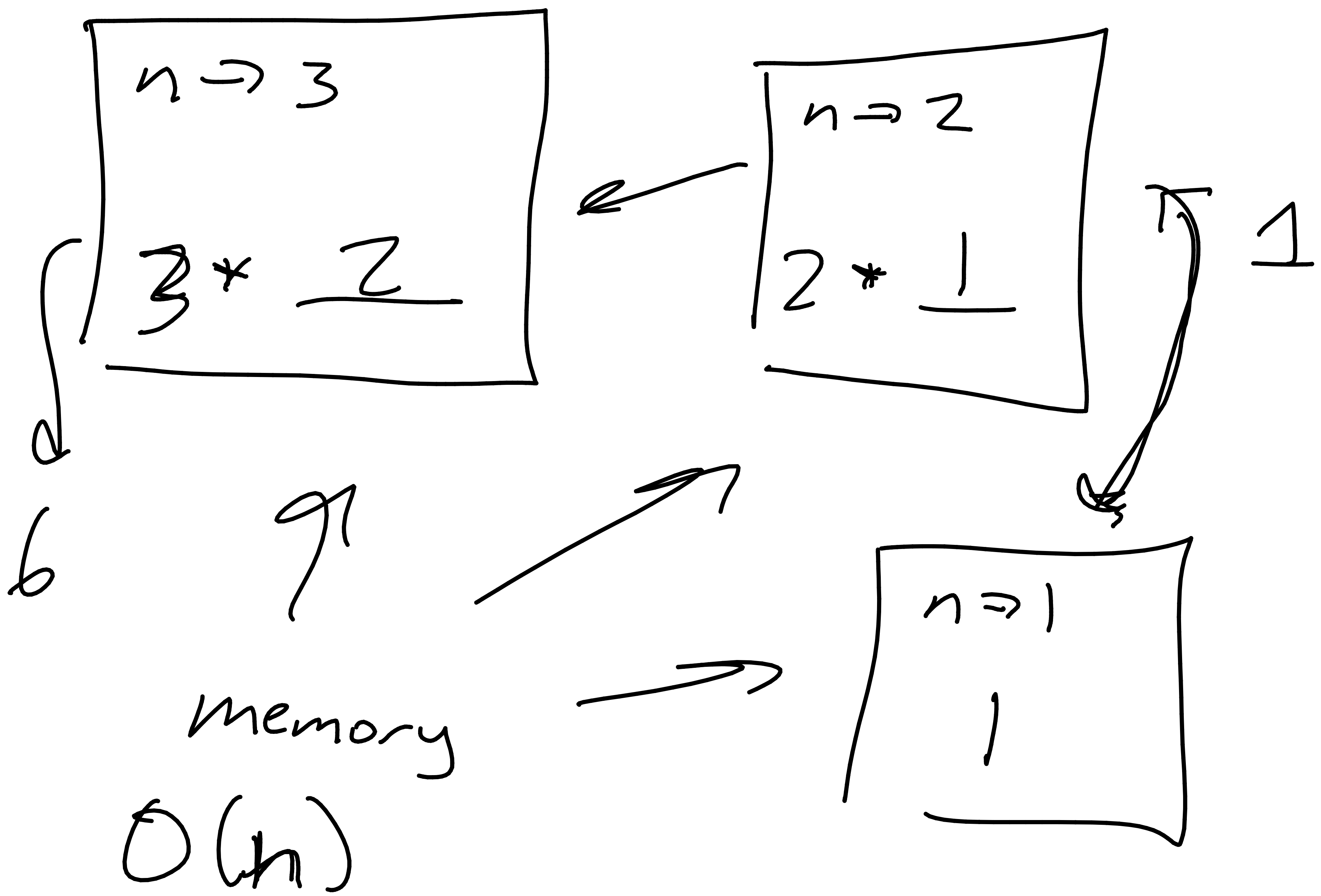
Recursion vs iteration.

What is bad about recursion from a perspective of memory/time, relative to loops?

- every recursive call makes another stack frame

```
(define fact
  (lambda (n)
     (if (equals? n 1)
          1
          (* 1  (fact (-  n  1)))))))))))
```

(fact 3)



Some examples - to do w/ a loop properly requires a stack and uses just as much memory anyway (e.g. depth first seach)

Can you optimize that memory, at least in some situations?