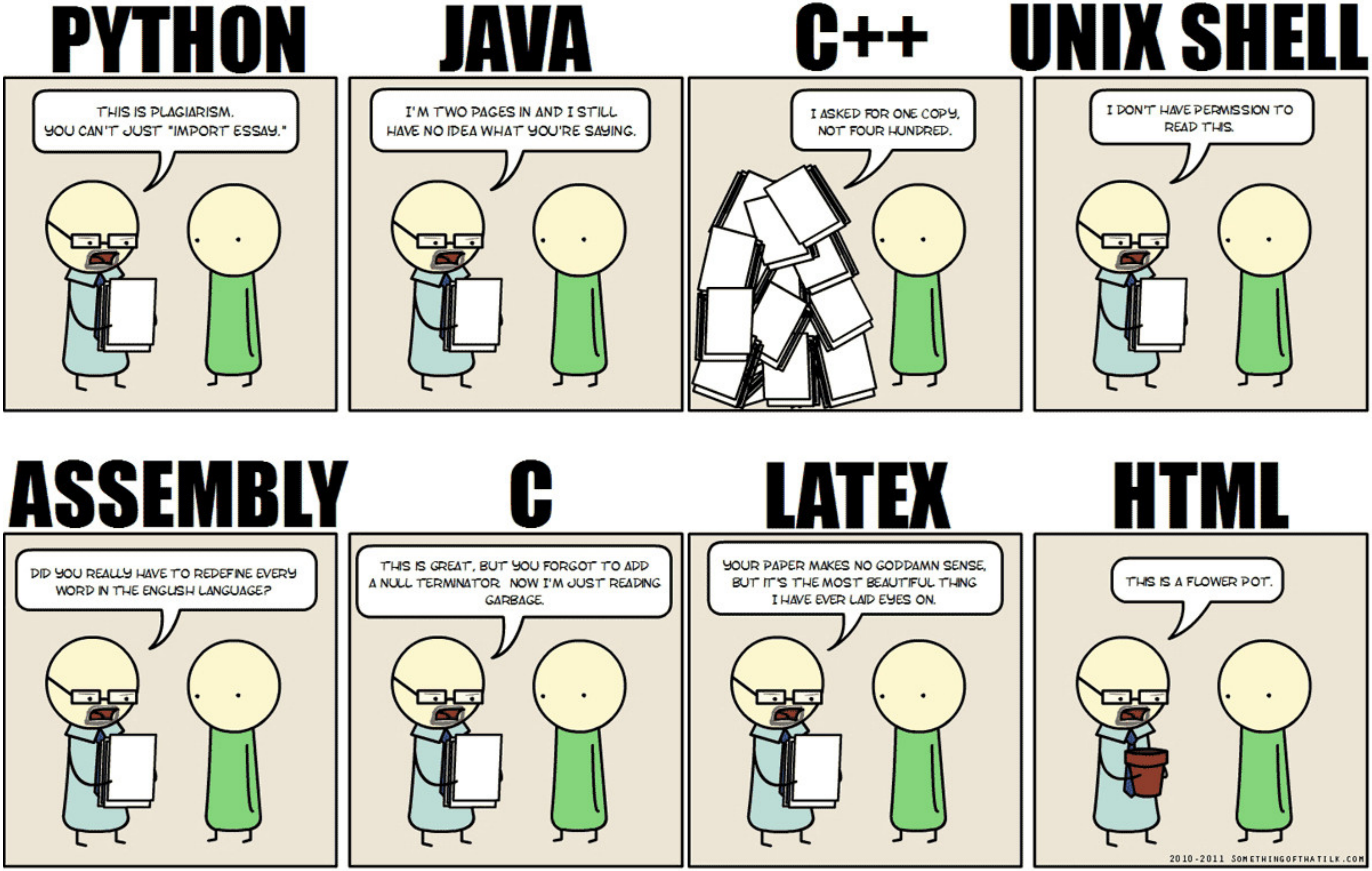

PYTHON
THIS IS PLAGIARISM. YOU CAN'T JUST "IMPORT ESSAY."
JAVA
I'M TWO PAGES IN AND I STILL HAVE NO IDEA WHAT YOU'RE SAYING.
C++
I ASKED FOR ONE COPY, NOT FOUR HUNDRED.
UNIX SHELL
I DON'T HAVE PERMISSION TO READ THIS.
ASSEMBLY
DID YOU REALLY HAVE TO REDEFINE EVERY WORD IN THE ENGLISH LANGUAGE?
C
THIS IS GREAT, BUT YOU FORGOT TO ADD A NULL TERMINATOR. NOW I'M JUST READING GARBAGE.
LATEX
YOUR PAPER MAKES NO GODDAMN SENSE, BUT IT'S THE MOST BEAUTIFUL THING I HAVE EVER LAID EYES ON.
HTML
THIS IS A FLOWER POT.
2010-2011 SOMETHINGOFTHATILK.COM

CS 152: *Programming Language Paradigms*

Higher Order Functions

Prof. Tom Austin
San José State University

Functional languages treat programs as *mathematical functions*.

Definition: *A function is a rule that associates to each x from some set X of values a unique y from a set of Y values.*

$$y = f(x)$$

f is the name of the function

Definition: *A function is a rule that associates to each x from some set X of values a unique y from a set of Y values.*

x is a variable in the set X

$$y = f(x)$$

X is the *domain* of f.
x∈X is the *independent variable*.

Definition: *A function is a rule that associates to each x from some set X of values a unique y from a set of Y values.*

$$y = f(x)$$

y is a variable in the set Y

Y is the *range* of f.
$y \in Y$ is the *dependent variable*.

Qualities of Functional Programing

1. Functions clearly distinguish **inputs** from **outputs**
2. No assignment (pure)
3. No loops (pure)
4. Result depends **only** on input values
   - Evaluation order does not matter
5. **Functions are first class values**

Referential transparency

In **purely functional** programs you can

- replace an expression with its value
- write code free of side-effects

Functions are **first-class** data values.

We can do anything with them that we can do with other values.

Higher-order function

A function that

- takes functions as arguments; or
- returns a function as its result; or
- dynamically constructs new functions

Higher-order functions example

(in class)

Lab 3: `map` and `filter`

See Canvas for a more detailed explanation.

1. Using `map`, implement `strings-to-nums`.
   ```
       (strings-to-nums '("1" "2"))
   -> '(1 2)
   ```
2. Using `map`, create a `make-names` function:
   1. input: list of first names, list of last names
   2. output: list of full names
3. Using the filter function, write a function that takes a list of employees and returns a list containing only managers.

Fold variants

- `foldr`
  - Traverses from the right
  - `(foldr * 1 '(2 4 8))` is the same as `(* 2 (* 4 (* 8 1)))`
- `foldl`
  - Traverses from the left
  - `(foldl * 1 '(2 4 8))` is the same as `(* 8 (* 4 (* 2 1)))`
- WARNING: Different languages define fold slightly differently.

`foldr` evaluation

```
    (foldr cons '() '(1 2 3))
->  (cons 1 (foldr cons '() '(2 3)))
->  (cons 1 (cons 2 (foldr cons '() '(3))))
->  (cons 1 (cons 2 (cons 3 (foldr cons '()
                                        '())))))
->  (cons 1 (cons 2 (cons 3 '())))
->  (cons 1 (cons 2 '(3)))
->  (cons 1 '(2 3)))
->  '(1 2 3)
```

`foldl` evaluation
(slightly inaccurate – emphasizing order of operations)

accumulator

```
   (foldl cons '() '(1 2 3))
-> (foldl cons (cons 1 '()) '(2 3))
-> (foldl cons (cons 2 (cons 1 '())) '(3))
-> (foldl cons (cons 3 (cons 2 (cons 1 '())))
                                          '())
-> (cons 3 (cons 2 (cons 1 '())))
-> (cons 3 (cons 2 '(1)))
-> (cons 3 '(2 1))
-> '(3 2 1)
```

`foldl` evaluation

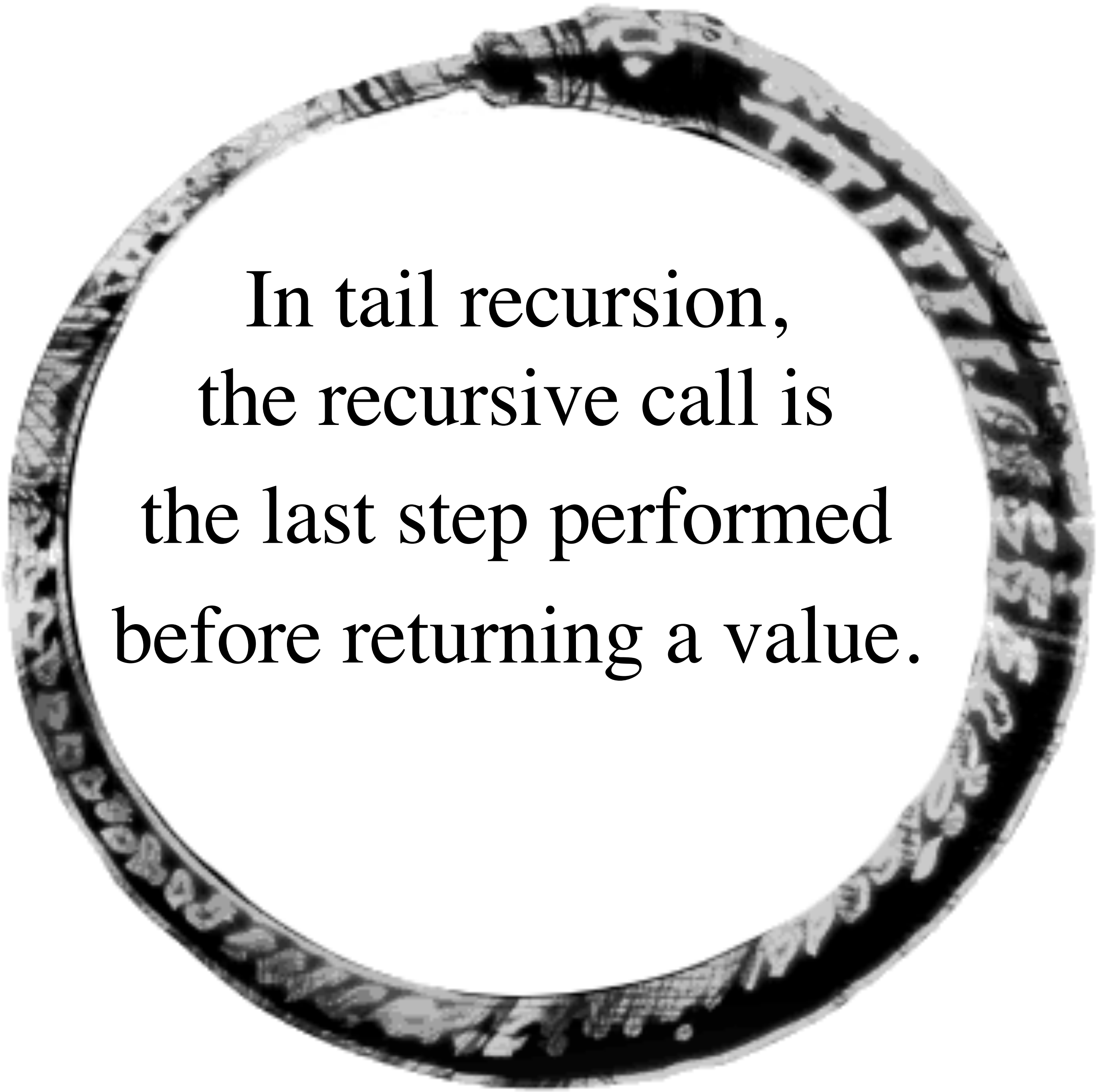
(accurate version)

```
   (foldl cons '() '(1 2 3))
-> (foldl cons (cons 1 '()) '(2 3))
-> (foldl cons '(1) '(2 3))
-> (foldl cons (cons 2 '(1)) '(3))
-> (foldl cons '(2 1) '(3))
-> (foldl cons (cons 3 '(2 1)) '())
-> (foldl cons '(3 2 1) '())
-> '(3 2 1)
```

Tail Recursion

Iterative solutions tend to be more efficient than recursive solutions.

However, compilers are very good at optimizing a *tail recursive* functions.



In tail recursion,
the recursive call is
the last step performed
before returning a value.

Is this function tail-recursive?

```
public int factorial(int n) {
    if (n==1) return 1;
    else {
        return n * factorial(n-1);
    }
}
```

No: the last step is multiplication

Is this function tail-recursive?

```
public int factorialAcc(int n, int acc){
    if (n==1) return acc;
    else {
        return factorialAcc(n-1, n*acc);
    }
}
```

Yes: the recursive step is the last thing we do

Which version is tail-recursive?

```
(define (fact n)
 (if (= n 1)
   1
   (* n
      (fact
        (- n 1)))))
```

```
(define (fact n a)
 (if (= n 1)
   a
   (fact
     (- n 1)
     (* n a))))
```

Lab 4: `foldl` and `foldr`

See Canvas for details.