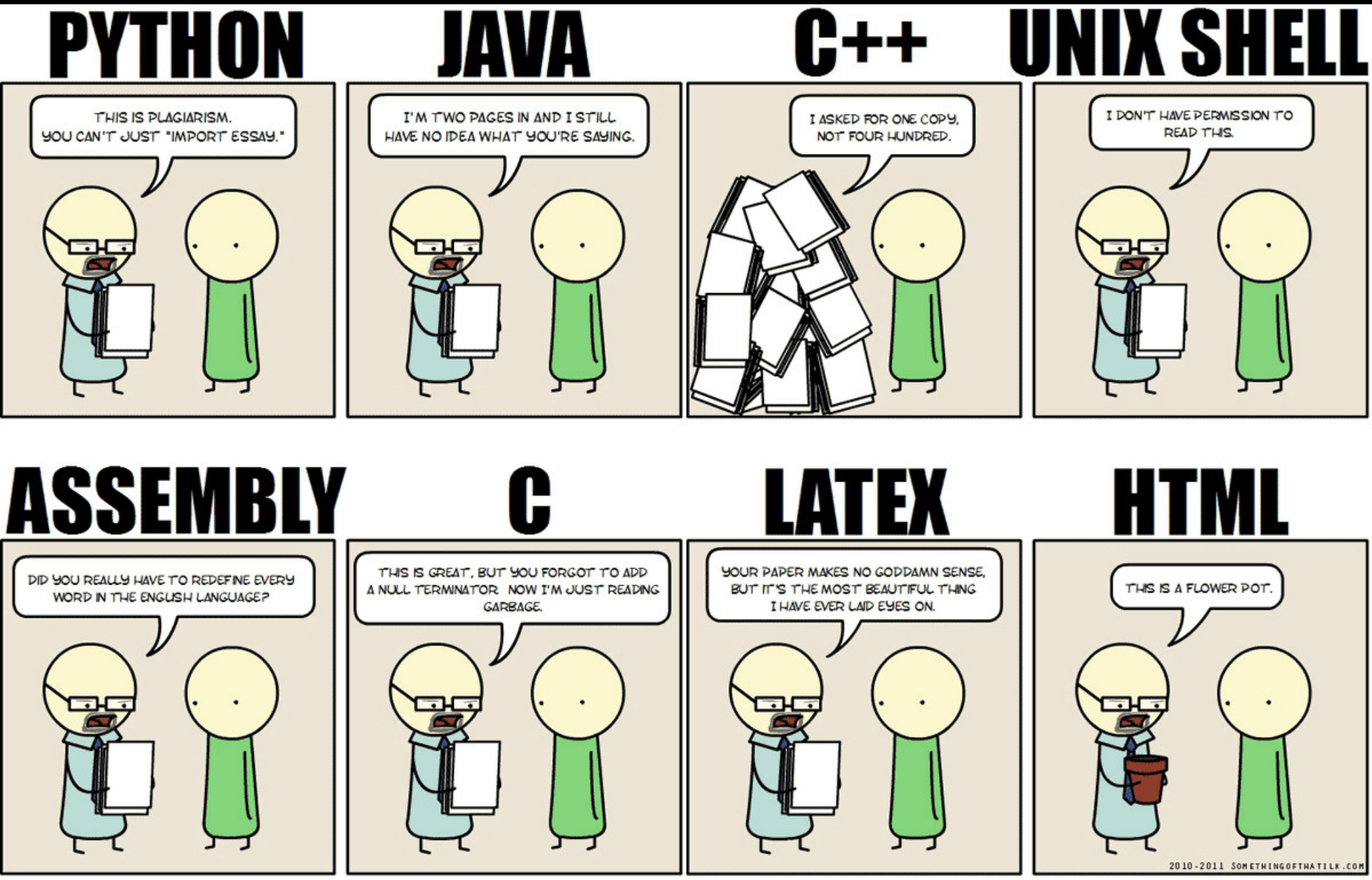
PYTHON
THIS IS PLAGIARISM. YOU CAN'T JUST "IMPORT ESSAY."
JAVA
I'M TWO PAGES IN AND I STILL HAVE NO IDEA WHAT YOU'RE SAYING.
C++
I ASKED FOR ONE COPY, NOT FOUR HUNDRED.
UNIX SHELL
I DON'T HAVE PERMISSION TO READ THIS.
ASSEMBLY
DID YOU REALLY HAVE TO REDEFINE EVERY WORD IN THE ENGLISH LANGUAGE?
C
THIS 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

Lab review

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.
$x \in X$ 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)

Map example

```
(define (add-one x)
  (+ x 1))

(add-one 1)

(map add-one '(1 2 3))
```

Returns 2

Returns '(2 3 4)

More map examples

```
(define lst '(1 2 3 4 5))

(map number->string lst)

(map (lambda(x)(* x x)) lst)

(map number->string
    (map (λ(x) (expt 2 x))
        lst))
```

Filter example

```
(positive? 3)
```

true

```
(positive? -2)
```

false

```
(filter positive?
    '(1 2 -3 4 0 5))
```

'(1 2 4 5)

Combining higher-order functions

```
(map (λ(x)(* x x))
     (filter even?
          '(1 2 3 4 5 6)))
```

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.