Dilbert, by Scott Adams

CS 152: *Programming Language Paradigms*

Contracts

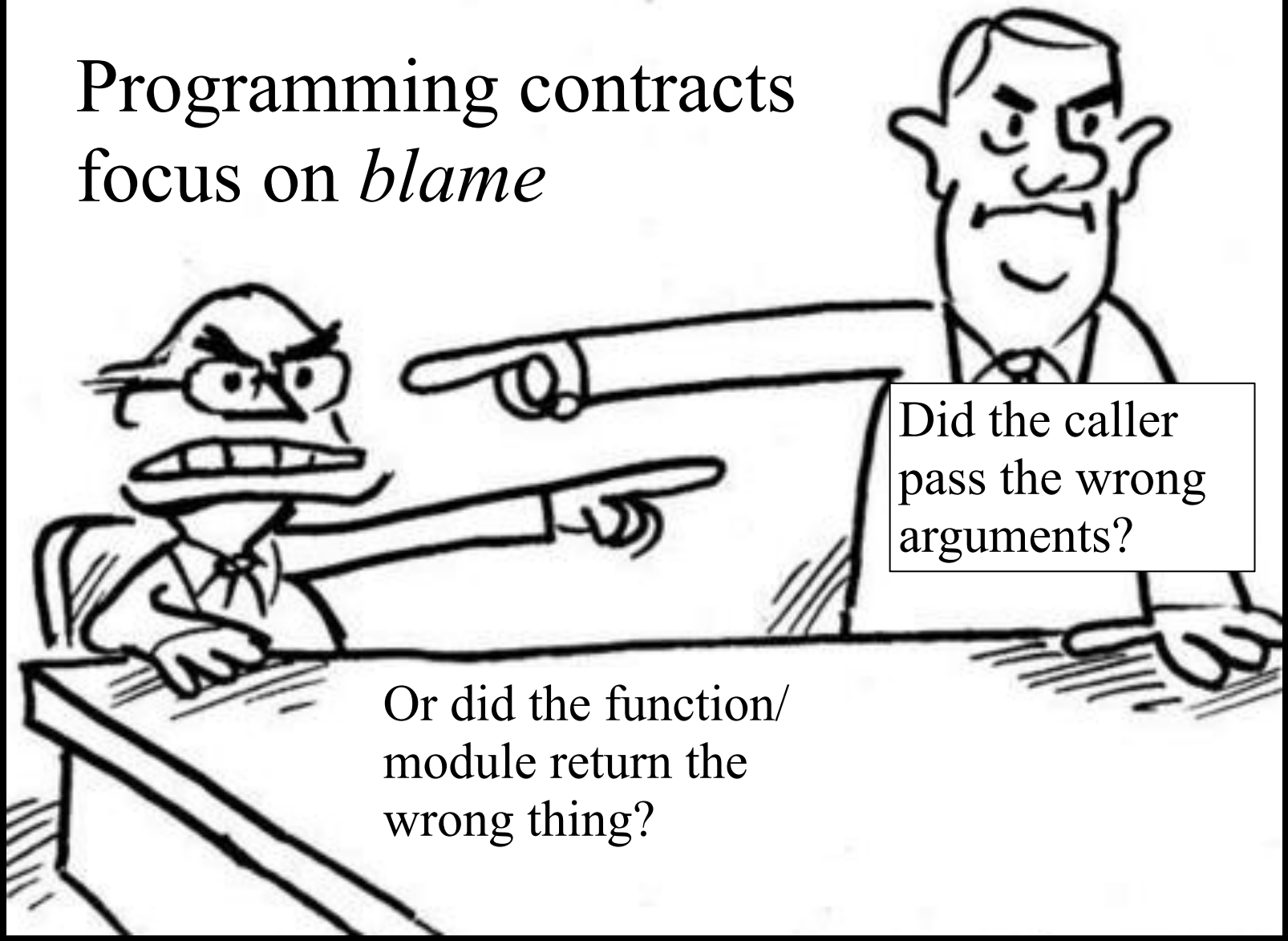
Prof. Tom Austin
San José State University

What is a contract?

Contracts:

- clearly delineate obligations and **responsibilities** for each party
- let each party know what **guarantees** to expect from others
- *establish who is to **blame** when something goes wrong*

Programming contracts focus on *blame*



A programming contract consists of:

- **Preconditions**: requirements for the input
  - if they do not hold, we blame the caller
- **Postconditions**: promises for the output
  - if they do not hold, we blame the library

Comparison to Type Checking

Type checkers:

+ Enforcement at *compile time*
- Less flexible
+ Less runtime overhead

Contracts:

- Enforcement at *run time*
+ More flexible
- More runtime overhead

Contracts in Racket

Review: modules

```
(provide add-one fav-num)

(define fav-num 42)

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

(define (dec-one x)
  (- x 1))
```

`add-one` and `fav-num` are public.

Using a module

```
(require "math-module.rkt")

(add-one fav-num)
; evaluates to 43

(dec-one 10)
```

Import module from specified file

```
dec-one: unbound identifier in module in: dec-one
```

Contracts & Boundaries

- Modules provide natural boundaries between code.
- `contract-out` specifies the requirements.

But what *is* a contract in Racket?

A function (predicate) that:

- Takes exactly one argument
- Returns `#t` or `#f`

Any predicate is a valid contract.

```
(provide (contract-out
   [fav-num number?])
   )

(define fav-num "four")

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

(define (dec-one x)
  (- x 1))
```

The module guarantees that `fav-num` is a number

But…

```
(require "math-module.rkt")
(+ fav-num 1)
```

```
fav-num: broke its contract
promised: number?
produced: "four"
in: number?
contract from:
    …/math-module.rkt
blaming: …/math-module.rkt
at: …/math-module.rkt:6.11
```

```
(provide (contract-out
   [fav-num number?]
   [add-one (-> number? number?)]))

(define fav-num 42)

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

(define (dec-one x)
  (- x 1))
```

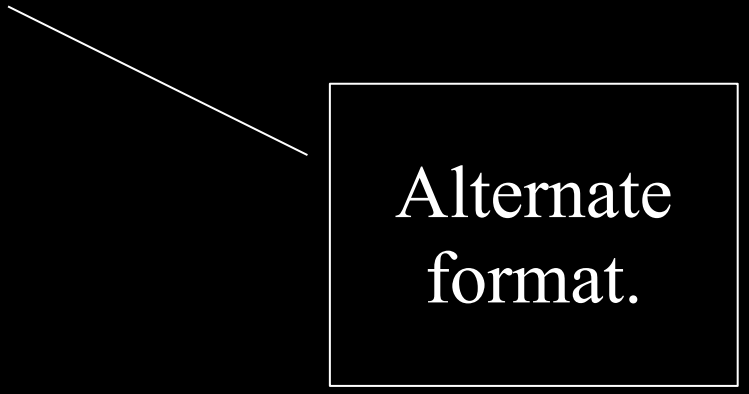
-> is a *contract combinator.* It combines a contract on the input with a contract on the output

```
(provide (contract-out
   [fav-num number?]
   [add-one
        (number? . -> . number?)]))

(define fav-num 42)

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

(define (dec-one x)
  (- x 1))
```



```
(require "math-module.rkt")
(add-one "forty-nine")
```

```
add-one: contract violation
expected: number?
given: "forty-nine"
in: the 1st argument of
    (-> number? number?)
contract from:
    …/math-module.rkt
blaming: …/using-contract.rkt
at: …/math-module.rkt:7.11
```

Contracts on Functions

- **The good**: able to catch internal errors.
- **The bad**: repeats checking of contracts
  –degrades performance.

Contract on a function

```
(define/contract
    (make-num-pair x y)
    (-> number? number? pair?)
  (cons x y))
```

Quicksort
(in-class)

Contracts & structs lab

- Implement contracts for a banking account module.
- See Canvas for details.
- For more information on contracts, see http://docs.racket-lang.org/guide/contracts.html