Syntax Warnings
Language-Integrated Nitpicking
#lang racket/base

(require "util.rkt"
  racket/match
    (for-syntax "macro-util.rkt"
      "macros.rkt"
      racket/vector
        (for-syntax "macro-util-more.rkt"
          (for-template "macro-output.rkt")))

Does this look wrong?
#lang racket/base

(require (for-syntax "macro-util.rkt"
               "macro-util-more.rkt")
         (for-template "macro-output.rkt")
         racket/match
         racket/vector
         "macros.rkt"
         "util.rkt")

What about this?
foo.rkt

#lang racket/base

(require (for-syntax "macro-util.rkt"
                     "macro-util-more.rkt")
        (for-template "macro-output.rkt"
                     racket/match
                     racket/vector
                     "macros.rkt"
                     "util.rkt")

Should this be enforced?
Compile errors are pass/fail
But humans use a continuum
#lang racket/base

(require "util.rkt"
  racket/match
    (for-syntax "macro-util.rkt"
            "macros.rkt"
      racket/vector
        (for-syntax "macro-util-more.rkt"
                        (for-template "macro-output.rkt")))

Plain Racket
WARNING!

Racket with syntax warnings
Racket with syntax warnings

Require clauses annotated with warnings
(define warning
  (syntax-warning
    #:message "require clauses not in phase order"
    ...)))
(define warning
 (syntax-warning
  #:message "require clauses not in phase order"
  #:stx require-stx ...
 ))
(define warning
  (syntax-warning
    #:message "require clauses not in phase order"
    #:stx require-stx
    #:kind require:phase-order
    ...)))
(define warning
  (syntax-warning
   #:message "require clauses not in phase order"
   #:stx require-stx
   #:kind require:phase-order
   ...
   ))

(define-warning-kind require:phase-order)
(define warning
  (syntax-warning
    #:message "require clauses not in phase order"
    #:stx require-stx
    #:kind require:phase-order
    ...)))
(define warning
 (syntax-warning
   #:message "require clauses not in phase order"
   #:stx require-stx
   #:kind require:phase-order
   #:fix fixed-stx))
(define warning
  (syntax-warning
    #:message "require clauses not in phase order"
    #:stx require-stx
    #:kind require:phase-order
    #:fix fixed-stx))

(define stx/warning (syntax-warn stx warning))
(define warning
  (syntax-warning
   #:message "require clauses not in phase order"
   #:stx require-stx
   #:kind require:phase-order
   #:fix fixed-stx))

(define stx/warning (syntax-warn stx warning))

(syntax-warnings stx/warnings)
(define warning
  (syntax-warning
   #:message "require clauses not in phase order"
   #:stx require-stx
   #:kind require:phase-order
   #:fix fixed-stx))

(define stx/warning (syntax-warn stx warning))

(syntax-warnings stx/warnings)

What now?
> raco warn foo.rkt
Checking 1 module
raco warn: /path/to/foo.rkt

[require:phase-order]
require clauses not in phase order

2 (require foo
3       (for-syntax bar)
4       baz)
Warning kinds provide a documentation hook
> raco warn foo.rkt
Checking 1 module
raco warn: /path/to/foo.rkt

[require:phase-order]
require clauses not in phase order
...
suggested fix:

2 (require (for-syntax bar)
3         baz
4         foo)
> raco fix foo.rkt
Checking 1 module
raco fix: /path/to/foo.rkt: 1 warning, fixing

raco fix applies suggestions automatically
> raco fix --dry foo.rkt
Checking 1 module
raco fix: /path/to/foo.rkt: 1 warning, would fix

--dry option makes no changes
install:
- raco pkg install --auto
  $TRAVIS_BUILD_DIR/mock
  $TRAVIS_BUILD_DIR/mock-rackunit

script:
- raco warn --package mock
- raco test --package mock

Options for checking entire packages
Use in CI to enforce warnings
What if a warning is wrong?
Or unhelpful?
Or just not your style?
What if a warning is wrong?
Or unhelpful?
Or just not your style?
We can configure warnings with submodules
What if I don't want to use a new language?
We don't need a language to add warnings
awesome.rkt

#lang racket/base

(require awesome-library)

;; deprecated! use (thing ...) instead
(make-thing ...)

awesome-library can attach syntax warnings to uses of make-thing
> raco warn awesome.rkt

[make-thing:deprecated]
make-thing is deprecated, use thing instead

6 (make-thing ...)

suggested fix:

6 (thing ...)

raco warn and raco fix just work!
With syntax warnings, packages can gradually improve client code over time
Future Work
#lang racket/base

(provide (deprecate-out make-thing) thing)

(define thing ...)
(define make-thing ...)
> raco warn --watch --package my-package
raco warn: watching for changes...

More complex tooling
Better support for custom readers
Built-in warnings for everything!
Built-in warnings for everything!

- Whitespace formatting
- Unused imports
- Unused variables
- Let where define would do
Built-in warnings for everything!

- Whitespace formatting
- Unused imports
- Unused variables
- Let where define would do
- Whatever you want!
Package system features
Package system features

• Package-wide configuration
• Catalog build system integration
• Automatically fixing client packages
• Using warnings for streamlined upgrades
Open problems
Suggested-fix construction is hard

Source locations need to be consistent
Macros can hide syntax warnings

If a macro produces bad code, who’s to blame?
Configurable warnings?

People have different preferences for max line length

How should this information be specified?
GUI integration

What’s a good way to show warnings in DrRacket? What about emacs?
Questions?

raco pkg install syntax-warn