

Slideshow Tutorial



Press the spacebar to continue

About Slideshow

Slideshow is a library for creating slide presentations

- A Slideshow presentation is a Racket program
- Instead of a WYSIWYG interface, you get the power of Racket

How to Control this Viewer

Alt-q, Cmd-q, or Meta-q	: end show
Esc	: if confirmed, end show
→, ↓, Space, f, n, or click	: next slide
←, ↑, Backspace, Delete, or b	: previous slide
1 / g	: first/last slide
a / s	: previous/next slide name
Alt-g, Cmd-g, or Meta-g	: select a slide
Alt-p, Cmd-p, or Meta-p	: show/hide slide number
Alt-c, Cmd-c, or Meta-c	: show/hide commentary
Alt-d, Cmd-d, or Meta-d	: show/hide preview
Alt-m, Cmd-m, or Meta-m	: show/hide mouse cursor
Shift-→, etc.	: move window 1 pixel
Alt-→, Cmd-→, or Meta-→, etc.	: move window 10 pixels

Slideshow Programs

A Slideshow program has the form

```
#lang slideshow  
... code to generate slide content ...
```

To run a Slideshow program,

- Double-click the **Slideshow** executable or run **slideshow** on the command line
- Click the **Open File...** link and select the Slideshow program file, such as **mytalk.rkt**

Slideshow Programs

A Slideshow program has the form

```
#lang slideshow  
... code to generate slide content ...
```

Alternately, run a Slideshow program in DrRacket:

- Open `mytalk.rkt` in DrRacket

DrRacket's language should change automatically to
Module

- Click **Run** in DrRacket

Use DrRacket only if you trust the program

Slideshow Programs

A Slideshow program has the form

```
#lang slideshow  
... code to generate slide content ...
```

Important security information:

A slideshow program has access to the *full Racket language*

If you don't know the creator of a slide program (or if you don't trust them), run the slides through the **Slideshow** executable or **slideshow** command line

When run in **Slideshow** instead of DrRacket, a slide program cannot write files or make network connections

Slideshow Programs

A Slideshow program has the form

```
#lang slideshow  
... code to generate slide content ...
```

When using a command line, you can specify the program directly:

```
slideshow mytalk.rkt
```

To print the talk:

```
slideshow --print mytalk.rkt
```

*Run **slideshow --help** for more options*

Slides and Picts

The body of a Slideshow program

1. Makes and combines *picts*

For example,

```
(t "Hello")
```

creates a pict like this:

Hello

2. Registers certain picts as slides

For example,

```
(slide (t "Hello"))
```

registers a slide containing only Hello

The Rest of the Tutorial

The rest of this tutorial (starting with the next slide) is meant to be viewed while reading the program source

The source is

[/Applications/Racket v6.11/share/pkgs/slideshow-exe/slideshow/tutorial-show.rkt](#)

Part I: Basic Concepts

This slide shows how four pics
get vertically appended by the
slide

function to create and install a slide

See how the

`t`

function takes a string and

produces a pict with a normal sans-serif font, but

`tt`

produces a pict with a fixed-width font?

The

`#:layout 'top`

option for

slide

aligns the slide to the screen top

Titles

The

#:title

option for

slide

supplies a title string

Paragraphs

Breaking up text into lines is painful, so the `para` function takes a mixture of strings and `picts` and puts them into a paragraph

It doesn't matter how strings are broken into parts in the code

The `para` function puts space between separate strings, but not before punctuation!

Paragraph Alignment

The **slide** function centers body pics horizontally, but **para** makes a picture with left-aligned text

The **frame** function wraps a frame around a pict to create a new pict, so you can easily see this individual pict

More Paragraph Alignment

The `#:align 'center` option for `para` generates a paragraph with centered lines of text

This line uses the `#:fill? #f` option

The `#:fill? #f` option creates a paragraph that is wrapped to fit the slide, but it allows the resulting `pict` to be more narrow than the slide

More Alignment

Of course, there's also `#:align 'right`

This paragraph is right-aligned using `#:align 'right`,
and `#:fill?` is `#f` the first time and `#t` the second time

For comparison, the same text using the default `#:fill?`:

This paragraph is right-aligned using `#:align 'right`,
and `#:fill?` is `#f` the first time and `#t` the second time

Unless your font happens to make the first line exactly as
the allowed width, the last box will be slightly wider with
extra space to the left

Paragraph Width

The `para` function by default makes the paragraph take $2/3$ of the slide width

The `para` function also accepts an explicit `#:width` option, which is 300 for this paragraph

Spacing

The `slide` functions insert space between each body pict

The amount of space is 24, which is the value of `(current-gap-size)`, which defaults to `gap-size`

Controlling Space

If you want to control the space, simply append the `picts` yourself to create one body `pict`

The first argument to `vc-append` determines the space between pictures

If the first argument is a `pict` instead of a number, then 0 is used

For text in one paragraph, the `para` function uses `(current-line-sep)`, which returns 5

Appending Picts

This is

vl-append

This is

vc-append

This is

vr-append

Horizontal Appending

This is **hc-append**
obviously

This is **ht-append**
obviously

This is **hb-append**
obviously

Text Alignment

hb1-append aligns text baselines

It's especially useful for font mixtures

ht1-append is the same for single lines

The difference between **ht1-append** and **hb1-append** shows up with multiple lines:

bottom lines align when using **hb1-append**

top lines align when using **ht1-append**

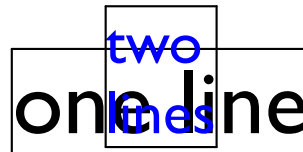
Superimposing



The `cc-superimpose` function puts pict's on top of each other, centered

Each of `l`, `r`, and `c` is matched with each of `t`, `b`, `c`, `bl`, and `tl` in all combinations with `-superimpose`

For example, `cb1-superimpose`:



By definition, the screen is 1024 x 768 units

If you have more or less pixels, the image is scaled

There's a margin, so the "client" area is 984 x 728

The font size is 32

Titled Client Area

If you use a title, then **titleless-page** is the same size as the area left for the body

It's useful

Text and Styles

Functions exist for **bold**, *italic*, and even ***bold-italic*** text

The **text** function gives you more direct control over the *font*, size, and even *angle*

Racket Code

For Racket code, the `slideshow/code` library provides a handy `code` macro for typesetting literal code

The `code` macro uses source-location information to indent code

```
(define (length l)
  (cond
    [(null? l) 0]
    [else (+ 1 (length (cdr l)))]))
```

Colors

Use the **colorize** function to color most things,
including text

A **colorize** applies only to sub-picts that do not
already have a **color**

➤ **Part I: Basic Concepts**

➤ **Part II: Practical Slides**

Using `make-outline` and more...

➤ **Part III: Fancy Picts**

➤ **Part IV: Advanced Slides**

➤ **Part V: Controlling the Background**

➤ **Part VI: Printing**

➤ **Conclusion**

Itemize

- Bulleted sequences are common in slides
- The `item` function makes a bulleted paragraph that is as wide as the slide
- + You can set the bullet, if you like, by using the `#:bullet` argument to `item`
 - Naturally, there is also `subitem`

Itemize

You could write `item` yourself:

```
(define (item . l)
  (let ([w (- client-w
                (pict-width bullet)
                (/ gap-size 2))])
    (html-append (/ gap-size 2)
                  bullet
                  (para #:width w l))))
```

where `bullet` is a constant pict: •

Grouping and Space

Sometimes you want to group items on a slide

- A bullet goes with a statement
- And another does, too

Creating a zero-sized pict with **(blank)** effectively doubles the gap, making a space that often looks right

Steps

- Suppose you want to show only one item at a time

Steps

- Suppose you want to show only one item at a time
- In addition to body picts, the **slide** functions recognize certain staging symbols
- Use '**next**' in a sequence of **slide** arguments to create multiple slides, one containing only the preceding content, and another with the remainder

Steps

- Suppose you want to show only one item at a time
- In addition to body picts, the **slide** functions recognize certain staging symbols
- Use '**next**' in a sequence of **slide** arguments to create multiple slides, one containing only the preceding content, and another with the remainder

'**next**' is not tied to **item**, though it's often used with **items**

Alternatives

Steps can break up a linear slide, but sometimes you need to replace one thing with something else

For example, replace this...

Alternatives

Steps can break up a linear slide, but sometimes you need to replace one thing with something else

... with something else

Alternatives

Steps can break up a linear slide, but sometimes you need to replace one thing with something else

... with something else

- An 'alts' in a sequence must be followed by a list of lists
- Each list is a sequence, a different conclusion for the slide's sequence
- Anything after the list of lists is folded into the last alternative

Alternatives

Steps can break up a linear slide, but sometimes you need to replace one thing with something else

... with something else

- An `'alts` in a sequence must be followed by a list of lists
- Each list is a sequence, a different conclusion for the slide's sequence
- Anything after the list of lists is folded into the last alternative

Of course, you can mix `'alts` and `'next` in interesting ways

- **Part I: Basic Concepts**
- **Part II: Practical Slides**
- **Part III: Fancy Picts**
 - Creating interesting graphics
- **Part IV: Advanced Slides**
- **Part V: Controlling the Background**
- **Part VI: Printing**
- **Conclusion**

Fancy Picts

In part I, we saw some basic pict constructors: `t`, `vl-append`, etc.

Slideshow provides many more...

Bitmaps

For example, the `bitmap` function loads a bitmap to display






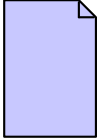
Clickbacks

The `clickback` function attaches an arbitrary thunk to a `pict` for interactive slides

Click Me

Tables

The `table` function makes rows and columns

First		<code>cons</code>
Second		<code>car</code>
Third		<code>cdr</code>
Fourth		<code>null?</code>

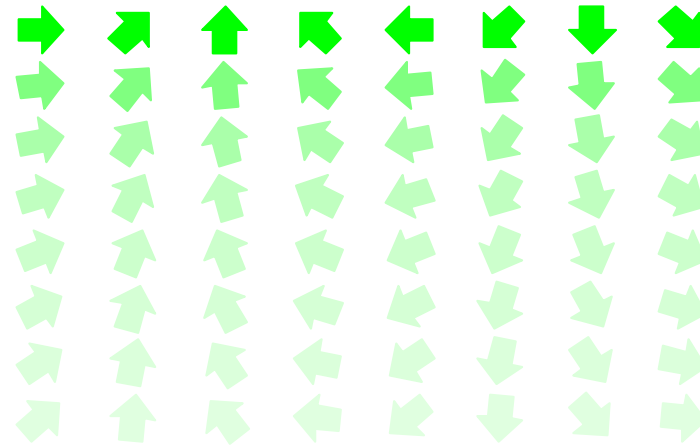
The above also uses `standard-fish`,
`jack-o-lantern`, `cloud`, and `file-icon`

Arrows

The `arrow` function creates an arrow of a given size and orientation (in radians)

Simple: 

Fun:



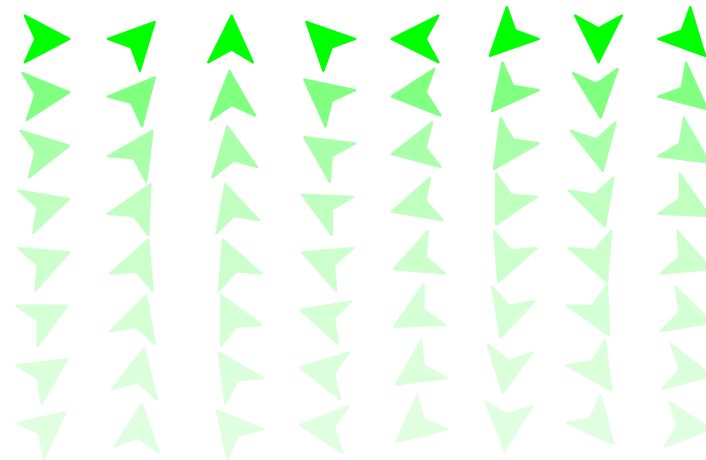
(That's 64 uses of `arrow`)

Arrows

The `arrowhead` function creates an arrowhead of a given size and orientation (in radians)

Simple: ◀

Fun:



(That's 64 uses of `arrowhead`)

Faces

The `pict/face` library makes faces

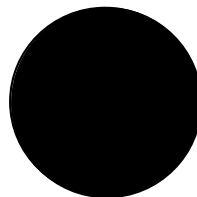
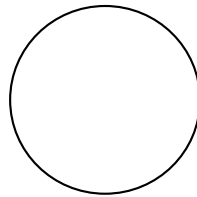


Arbitrary Drawing

The `dc` function provides an escape hatch to the underlying `racket/draw` library

For example, `(disk 100)` is the same as

```
(dc (lambda (dc dx dy)
      (send dc draw-ellipse dx dy 100 100)
      100 100 0 0))
```

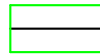


Frames

- As we've already seen, the **frame** function wraps a `frame` around a pict
- The **#:color** option wraps a `colored frame`; compare to **frame** followed by **colorize**, `like this`
- One way to increase the `line thickness` is to use **linewidth**
- It's often useful to `add space` around a pict with **inset** before framing it

Lines and Pict Dimensions

- The **hline** function creates a horizontal line, given a bounding width and height:



(The **hline** result is framed in green above)

- Naturally, there's also **vline**:



- To underline a pict, get its width using **pict-width**, then use **hline** and **vc-append**
- If the pict is text, you can restore the baseline using **refocus**

Placing Picts

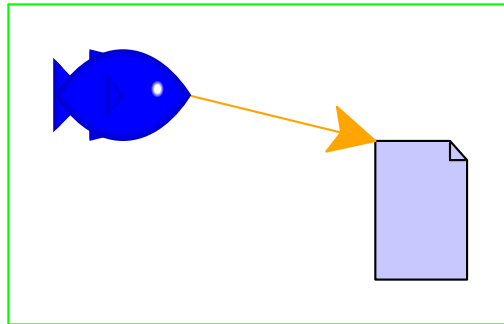
- Another underline strategy is to use **pin-over**, which places one pict on top of another to generate a new pict
- The new pict has the original pict's bounding box and baselines

(The green frame is the “bounding box” of the result)

- The **pin-over** function is useful with **pip-arrow-line** to draw an outgoing arrow without changing the layout

Finding Picts

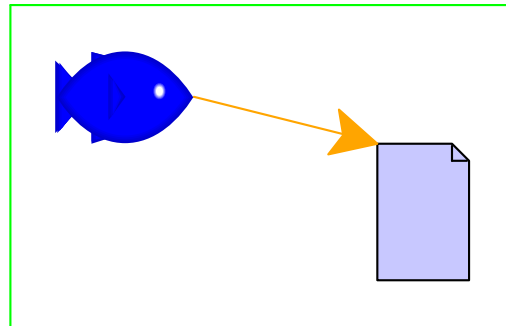
Typically, an arrow needs to go from one pict to another
Functions like **rc-find** locate a point of a pict (such as “right center”) inside a larger pict



There’s a **-find** function for every combination of **l**, **c**, and **r** with **t**, **c**, **b**, **bl**, and **tl**

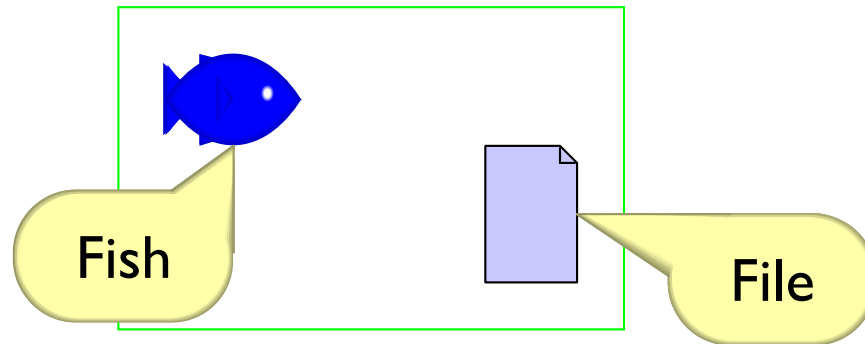
Connecting with Arrows

Actually, straight-arrow combinations are so common that Slideshow provides **pin-arrow-line**



Balloons

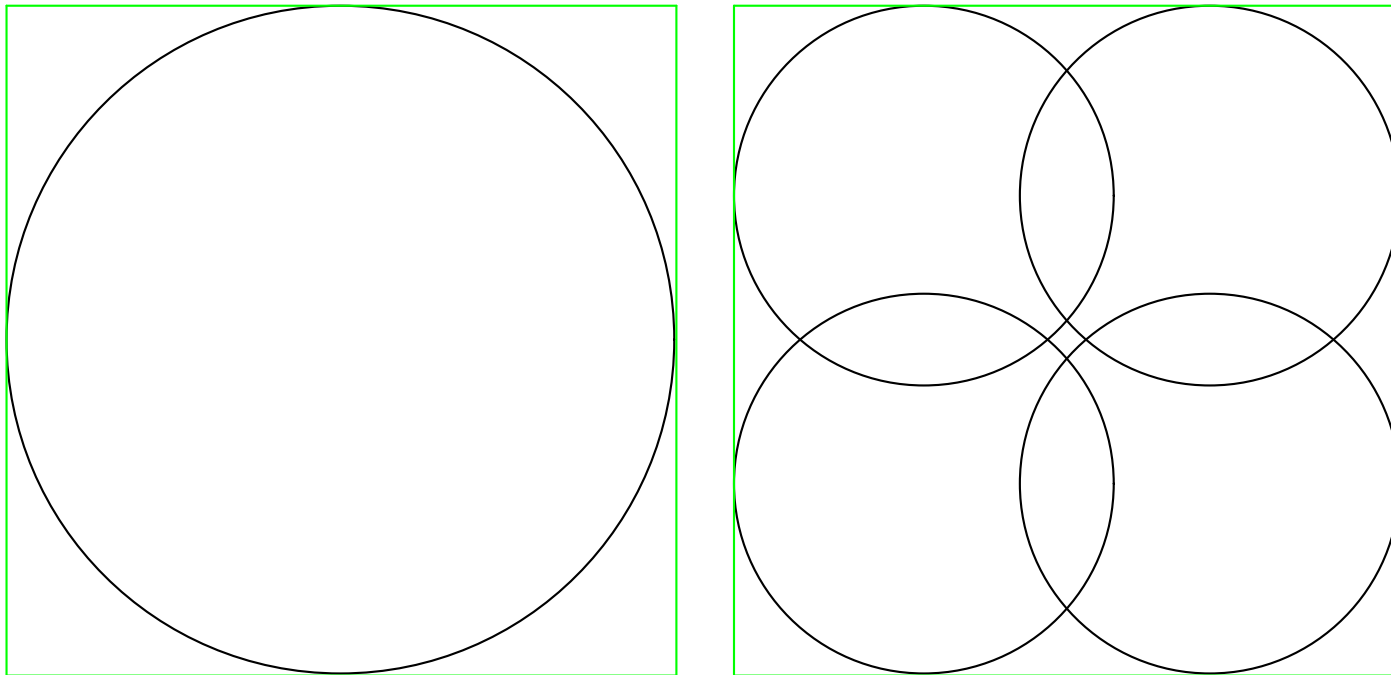
The `pict/balloon` library provides cartoon balloons — another reason to use `-find` functions



Ghosting

The **ghost** function turns a picture invisible

For example, the figure on the left and the figure on the right are the same size, because the right one uses the **ghost** of the left one



➤ **Part I: Basic Concepts**

➤ **Part II: Practical Slides**

➤ **Part III: Fancy Picts**

➤ **Part IV: Advanced Slides**

Beyond `'next` and `'alts`

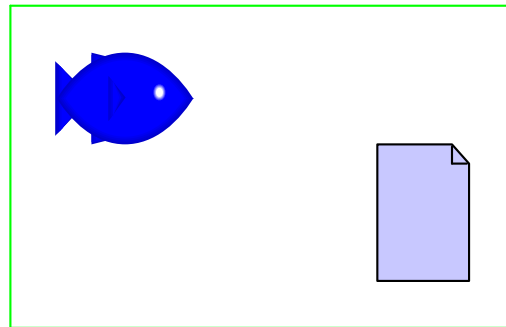
➤ **Part V: Controlling the Background**

➤ **Part VI: Printing**

➤ **Conclusion**

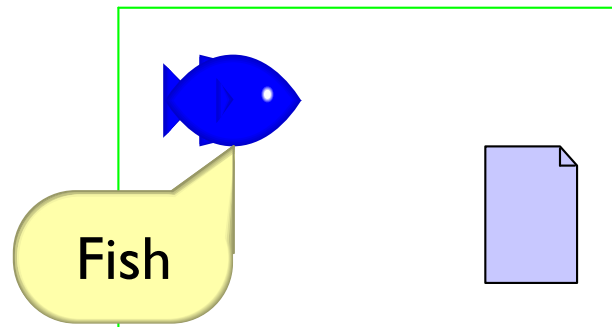
In-Picture Sequences

Although `'next` and `'alts` can create simple sequences, use procedure abstraction and `ghost` to create complex sequences inside pict assemblies



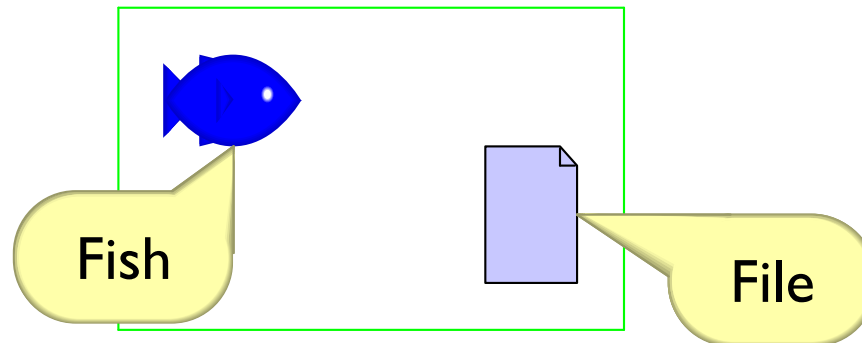
In-Picture Sequences

Although **'next** and **'alts** can create simple sequences, use procedure abstraction and **ghost** to create complex sequences inside pict assemblies



In-Picture Sequences

Although `'next` and `'alts` can create simple sequences, use procedure abstraction and `ghost` to create complex sequences inside pict assemblies



Larger example: [run code](#)

Named Steps

The `slideshow/step` library provides a `with-steps` form to better express complex sequences

Named Steps

The `slideshow/step` library provides a `with-steps` form to better express complex sequences

```
(with-steps
  (step-name ...)
  slide-expr)
```

A `with-steps` form has a sequences of step names followed by an expression to evaluate once for each step

Named Steps

The `slideshow/step` library provides a `with-steps` form to better express complex sequences

```
(with-steps
  (intro detail conclusion)
  slide-expr)
```

For example, the above has three steps: `intro`, `detail`, and `conclusion`

Named Steps

The `slideshow/step` library provides a `with-steps` form to better express complex sequences

```
(with-steps
  (intro detail conclusion)

  ((vonly intro)
   (t "For a start..."))))
```

In the body expression, use

`((vonly step-name) pict-expr)` to make *pict-expr* visible only during *step-name*

The expression `(vonly step-name)` produces either `ghost` or the identity function

Named Steps

The `slideshow/step` library provides a `with-steps` form to better express complex sequences

```
(with-steps
  (intro detail conclusion)
  ((vafter detail)
   (t "like this"))) )
```

Use `((vafter step-name) pict-expr)` to make *pict-expr* visible after *step-name*

Named Steps

The `slideshow/step` library provides a `with-steps` form to better express complex sequences

```
(with-steps
  (intro detail conclusion)
  ((vafter detail)
   (t "like this"))) )
```

There's also `vbefore`, `vbetween`, and more

Animations

The `#:timeout` option causes a slide to auto-advance, which can be used for animations.



Animations

The `#:timeout` option causes a slide to auto-advance, which can be used for animations.



Animations

The `#:timeout` option causes a slide to auto-advance, which can be used for animations.



Animations

The `#:timeout` option causes a slide to auto-advance, which can be used for animations.



Animations

The `#:timeout` option causes a slide to auto-advance, which can be used for animations.



Animations

The `#:timeout` option causes a slide to auto-advance, which can be used for animations.



Animations

The `#:timeout` option causes a slide to auto-advance, which can be used for animations.



Animations

The `#:timeout` option causes a slide to auto-advance, which can be used for animations.



Animations

The `#:timeout` option causes a slide to auto-advance, which can be used for animations.



Animations

The `#:timeout` option causes a slide to auto-advance, which can be used for animations.



Animations

The `#:timeout` option causes a slide to auto-advance, which can be used for animations.



(The face moved from left to right)

- **Part I: Basic Concepts**
- **Part II: Practical Slides**
- **Part III: Fancy Picts**
- **Part IV: Advanced Slides**
- **Part V: Controlling the Background**

Changing the overall look of your talk

- **Part VI: Printing**
- **Conclusion**

Controlling the Background

The `current-slide-assembler` parameter lets you change the overall look of a slide

For this slide and the previous one, the assembler

- Colorizes the uncolored content as dark red
- Left-aligns the title
- Draws a fading box around the slide

- **Part I: Basic Concepts**
- **Part II: Practical Slides**
- **Part III: Fancy Picts**
- **Part IV: Advanced Slides**
- **Part V: Controlling the Background**
- **Part VI: Printing**
 - Exporting slides as PostScript
- **Conclusion**

Printing

To export a set of slides as PostScript, use the `slideshow` command-line program:

```
slideshow --print mytalk.rkt
```

Slideshow steps through slides while producing PostScript pages

The slides will look bad on the screen — because text is measured for printing instead of screen display — but the PostScript will be fine

Condensing

Often, it makes sense to eliminate 'step' staging when printing slides:

```
slideshow --print --condense mytalk.rkt
```

Condensing

Often, it makes sense to eliminate 'step' staging when printing slides:

```
slideshow --print --condense mytalk.rkt
```

You can also condense without printing

```
slideshow --condense mytalk.rkt
```

Condensing

Often, it makes sense to eliminate 'step' staging when printing slides:

```
slideshow --print --condense mytalk.rkt
```

You can also condense without printing

```
slideshow --condense mytalk.rkt
```

For example, in condensed form, this slide appears without steps

Steps and Condensing

If you condense these slides, the previous slide's steps will be skipped

Steps and Condensing

If you condense these slides, the previous slide's steps will be skipped

Not this slide's steps, because it uses **'next!'**

Condensing Alternatives

Condensing *does not* merge 'alts' alternatives

But sometimes you want condensing to just use the last alternative

um...

Condensing Alternatives

Condensing *does not* merge 'alts' alternatives

But sometimes you want condensing to just use the last alternative

'alts~' creates alternatives where only the last one is used when condensing

Condensing Steps

The `slideshow/step` provides `with-steps~` where only the last step is included when condensing

Also, a `with-steps` step name that ends with `~` is skipped when condensing

Printing and Condensing Your Own Abstractions

You can customize your slides using **printing?** and **condensing?**

This particular slide is printed and not condensed

When you skip a whole slide, use **skip-slides** to keep page numbers in sync

- **Part I: Basic Concepts**
- **Part II: Practical Slides**
- **Part III: Fancy Picts**
- **Part IV: Advanced Slides**
- **Part V: Controlling the Background**
- **Part VI: Printing**
- **Conclusion**

This is the end

Your Own Slides

A Slideshow presentation is a Racket program in a module, so to make your own:

```
#lang slideshow  
... your code here ...
```

For further information, search for **slideshow** in the documentation