

# Slideshow Tutorial



*Press the spacebar to continue*

# About Slideshow

**Slideshow** is a library for creating slide presentations

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

## How to Control this Viewer

Alt-q, Cmd-q, or Meta-q	: end show
Esc	: if confirmed, end show
→, Space, <b>f</b> , <b>n</b> , or click	: next slide
←, Backspace, Delete, or <b>b</b>	: previous slide
<b>g</b>	: last slide
<b>1</b>	: first slide
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
Shift-→, etc.	: move window 1 pixel
Alt-→, Cmd-→, or Meta-→, etc.	: move window 10 pixels

# Slideshow Programs

A Slideshow program has the form

```
(module mytalk (lib "run.ss" "slideshow")  
  ... code to generate slide content ...)
```

in a file named `mytalk.scm`

---

Run a Slideshow program in DrScheme as follows:

- Open `mytalk.scm` in DrScheme
- Select **Choose Language** from the **Language** menu
- Choose the `(module ...)` language
- Click **Execute**

# Slideshow Programs

A Slideshow program has the form

```
(module mytalk (lib "run.ss" "slideshow")  
  ... code to generate slide content ...)
```

in a file named `mytalk.scm`

---

You can also execute it from the command line:

```
slideshow mytalk.scm
```

To print the talk:

```
slideshow --print mytalk.scm
```

*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

# Slides versus Picts

Technically, the pict concept comes from the "**texpict**" collection, and the "**slideshow**" collection builds on it

- The distinction between Slideshow and taxpict matters when you use Help Desk to find information
- For now, we ignore the distinction

# 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

</home/mflatt/proj/plt/collects/slideshow/tutorial-show.ss>



# 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?

Breaking up text into lines is painful, so the `page-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 `page-para` function puts space between separate strings, but not before punctuation!

The **slide/center** function centers the slide body vertically

All of the **slide** functions center the body pict horizontally, but **page-para** makes a slide-width 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

# Titles

The `slide/title` function takes a title string before the content  
pics

# Titles and Centering

The `slide/title/center` function centers the slide body vertically

## More Centering

The `page-para/c` function generates a paragraph with centered lines of text

This line uses the `page-para*` function

The `page-para*` function 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 `page-para/r`

And there's `page-para*/r`, which is different from `page-para*` or `page-para*/c` only if the paragraph takes multiple lines

For comparison, the same text using `page-para/r`:

And there's `page-para*/r`, which is different from `page-para*` or `page-para*/c` only if the paragraph takes multiple lines

Unless your font happens to make the `page-para*/r` box exactly as wide as this slide, the last box will be slightly wider with extra space to the left

# Spacing

The `slide` functions insert space between each body pict

The amount of space is 24, which is the value of `gap-size`

## Controlling Space

If you want to control the space, simply append the pict 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 `page-para` function uses `line-sep`, which is 2

# 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

**hbl-append** aligns text baselines

It's especially useful for font mixtures

**htl-append** is the same for single lines

The difference between **htl-append** and **hbl-append** shows up with multiple lines:

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

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

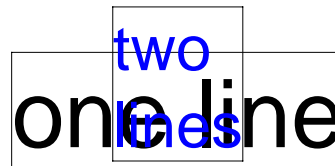
# Superimposing



The `cc-superimpose` function puts picts 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, `cbl-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

## More on Paragraphs

The `page-` in `page-para` makes the paragraph take the width of the slide

The `para` function  
requires an explicit  
size for the width of  
the paragraph, 300  
in this case

So `page-para` is a shorthand for `para` with `client-w`

Naturally, there is `para*`,  
`para*/c`, and `para*/r`

# 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*

# Scheme Code

For Scheme code, the `(lib "code.ss" "slideshow")` 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 `page-item` function makes a bulleted paragraph that is as wide as the slide
- + You can set the bullet, if you like, by using `page-item/bullet`
  - Naturally, there is also `page-subitem`

## Itemize

You could write `page-item` yourself:

```
(define (page-item . l)
  (html-append
    (/ gap-size 2)
    bullet
    (apply para
      (- client-w (pict-width bullet)
        (/ gap-size 2))
      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
- In addition to body pics, 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 **page-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

- 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.

The libraries

```
(lib "mrpict.ss" "tex pict")  
and (lib "utils.ss" "tex pict")
```

provide many more functions for creating pict

Slideshow re-exports all of those functions, so you can just use them

# Bitmaps

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



# Symbols

The `(lib "symbol.ss" "tex pict")` library provides various symbol constants, such as

$\in$  `sym:in`

$\rightarrow$  `sym:rightarrow`

$\infty$  `sym:infinity`

Slideshow does not re-export this library, so you must **require** it to use `sym:in`, etc.

Unless otherwise stated in the following slides, however, all definitions are provided by Slideshow



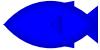


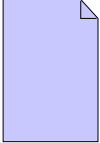
# Clickbacks

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

Click Me

# Tables

The **table** function makes rows and columns

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

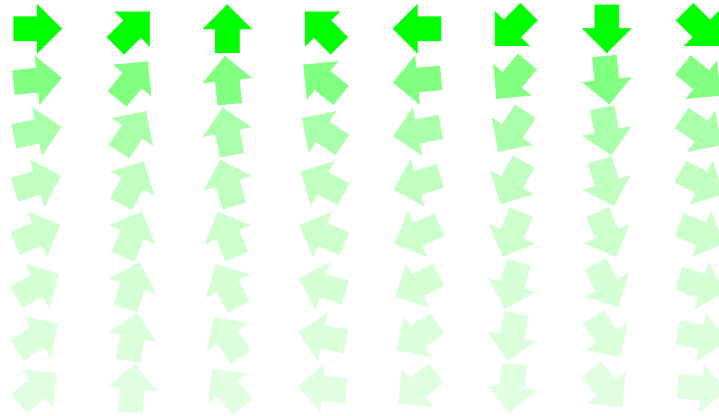
The above also uses **standard-fish**, **jack-o-lantern**, **cloud**,  
and **file-icon**

# Arrows

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

Simple: ←

Fun:



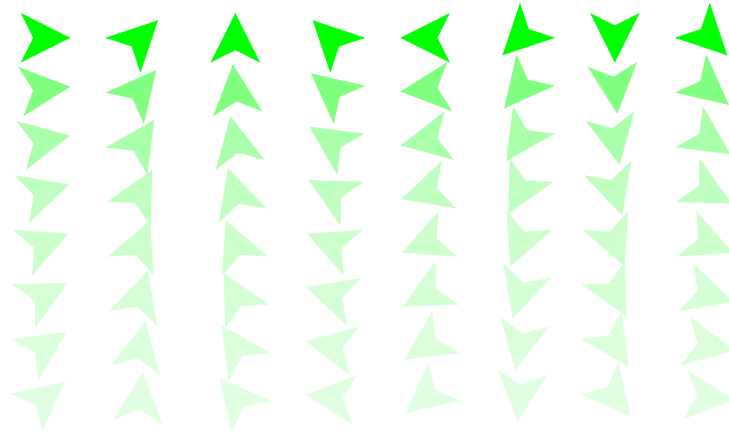
(That's 64 uses of **arrow**)

# 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 `(lib "face.ss" "tex pict")` library makes faces

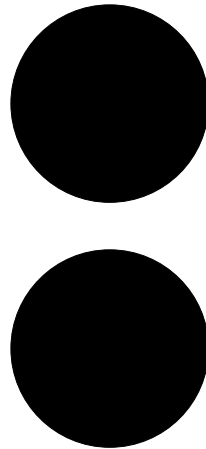


## Arbitrary Drawing

The `dc` function provides an escape hatch to the underlying MrEd toolkit

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)
```

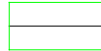


# Frames

- As we've already seen, the **frame** function wraps a `frame` around a pict
- The **color-frame** function 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 **pict-ascent**, **pict-ascent**, **drop**, and **lift**

(Granted, that's a little tricky)



## Placing Picts

- Another underline strategy is to use **place-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)

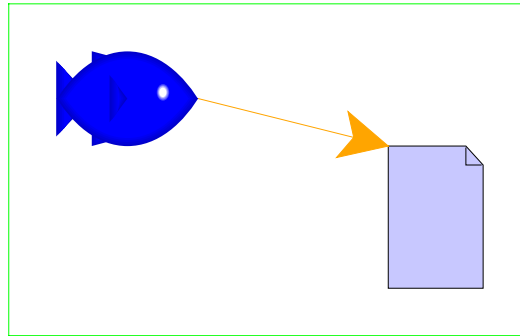
- The **place-over** function is useful with **arrow-line** to draw an outgoing arrow without changing the layout



# Finding Picts

Typically, an arrow needs to go from one pict to another

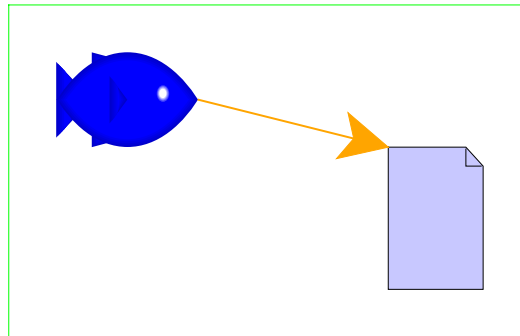
Functions like **find-rc** locate a point of a pict (such as "right center") inside a larger pict



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

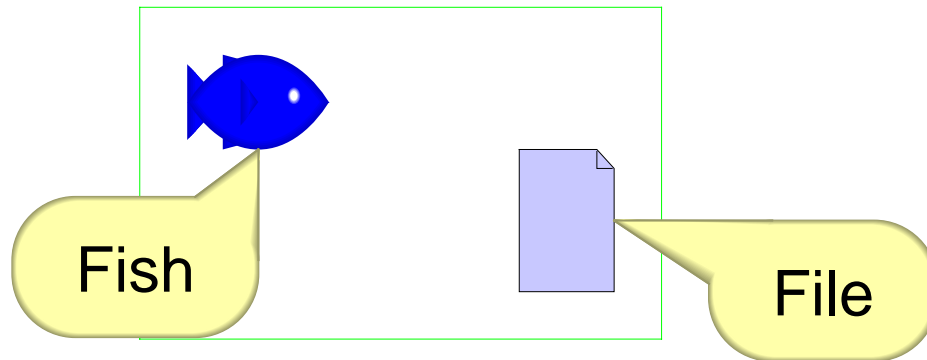
# Connecting with Arrows

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



# Balloons

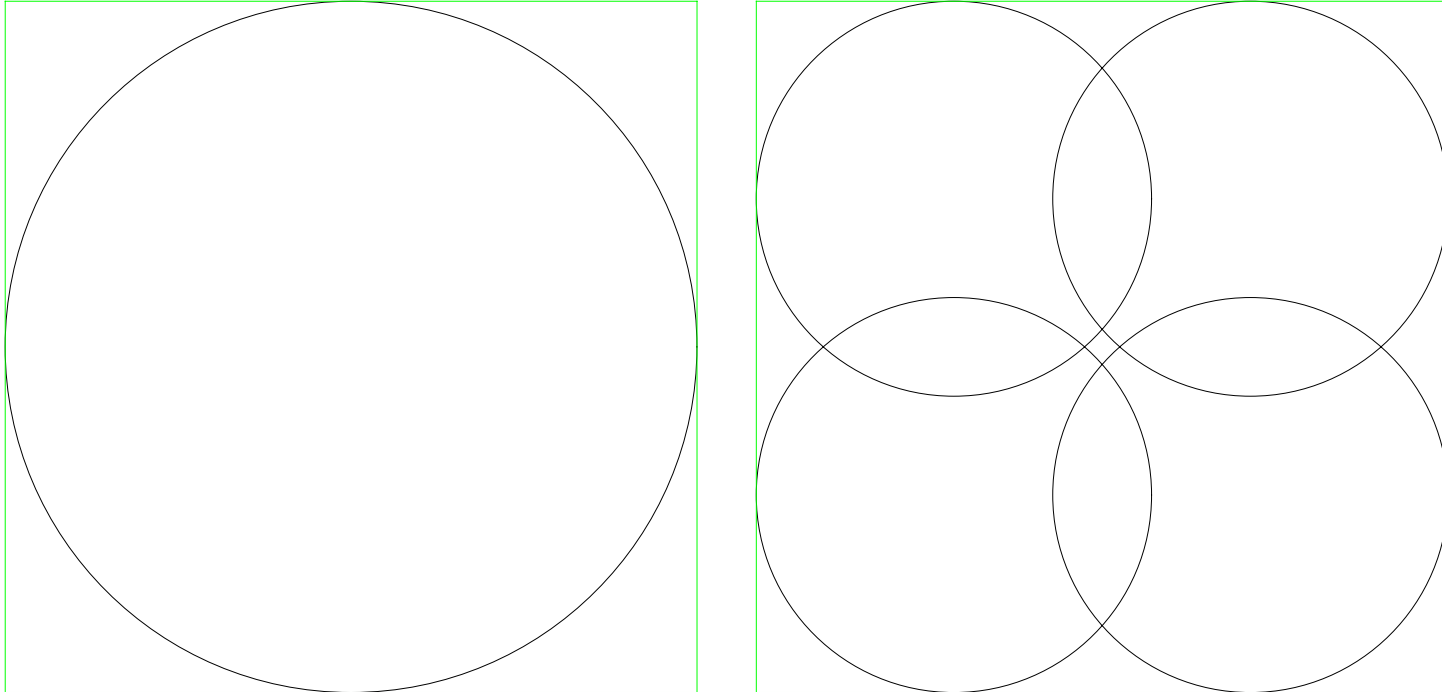
The `(lib "balloon.ss" "tex pict")` library provides cartoon balloons – another reason to use `find-` functions



# 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



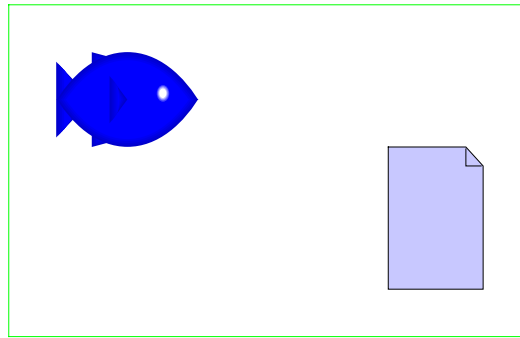
- **Part I: Basic Concepts**
- **Part II: Practical Slides**
- **Part III: Fancy Picts**
- **Part IV: Advanced Slides**

Beyond `'next` and `'alts`

- **Part V: Controlling the Background**
- **Part VI: Printing**
- **Conclusion**

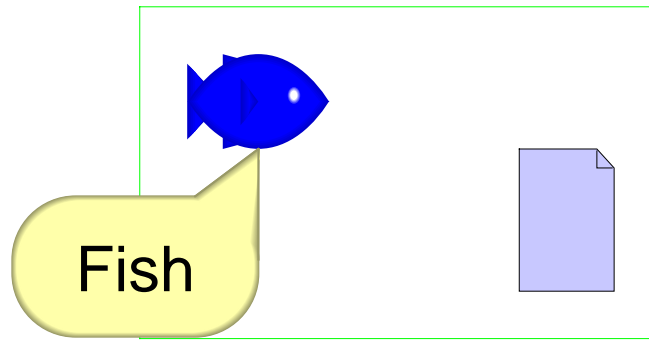
# In-Picture Sequences

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



# In-Picture Sequences

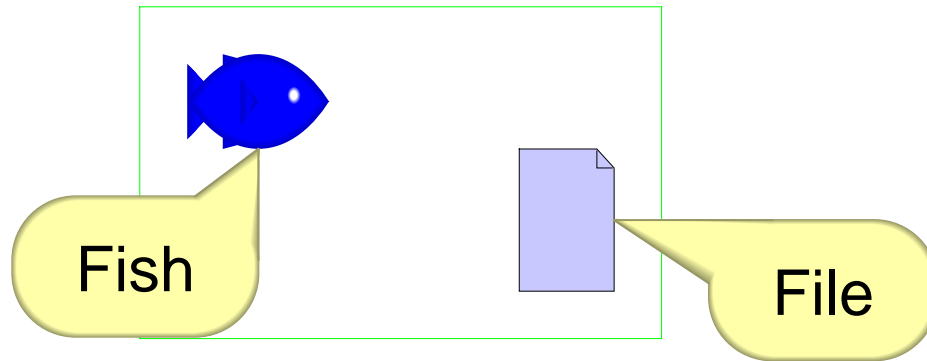
Although `'next` and `'alts` can create simple sequences, use procedure abstraction and `ghost` to create complex sequences inside pict assemblies





# 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 `(lib "step.ss" "slideshow")` library provides a `with-steps` form to better express complex sequences

## Named Steps

The `(lib "step.ss" "slideshow")` 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 `(lib "step.ss" "slideshow")` 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 `(lib "step.ss" "slideshow")` library provides a `with-steps` form to better express complex sequences

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

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

In the body expression, use `((only step-name) pict-expr)` to make *pict-expr* visible only during *step-name*

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

## Named Steps

The `(lib "step.ss" "slideshow")` 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 `(lib "step.ss" "slideshow")` library provides a `with-steps` form to better express complex sequences

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

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

# Transition Animations

The `scroll-transition` function scrolls some part of the current slide before shifting to the next slide.





## Transition Animations

The `scroll-transition` function scrolls some part of the current slide before shifting to the next slide.



The face should have 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 myttalk.scm
```

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 myttalk.scm
```

You can also condense without printing

```
slideshow --condense myttalk.scm
```

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

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

# Condensing Steps

The `(lib "step.ss" "slideshow")` 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 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 Scheme program in a module, so to make your own:

```
(module mytalk (lib "run.ss" "slideshow")  
  ... your code here ...)
```

For further information, search for `slideshow` and `tex pict` in Help Desk