

An Interactive Tutorial

DoomConf — May 2022

TEC

2022-05-14

Outline

Learning Emacs

Designing the Doom Tutorial

Implementation overview

Writing tutorials

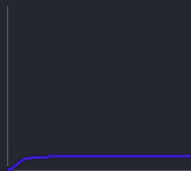
What's next?

Learning Emacs

It has a (deserved) reputation

Classical learning curves for some common editors

Notepad



Pico



Visual Studio



vi



emacs



Another one

DREW NEIL - SHARPENING THE VIM SAW



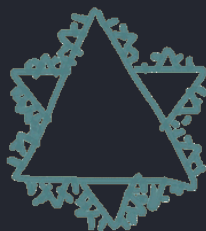
TEXT
MATE

CIRCLE



VIM

TRIANGLE



EMACS

RACTAL

Learning is hard

- Emacs has so much to it it's easy to feel lost

Learning is hard

- Emacs has so much to it it's easy to feel lost
- RTFM^(👉) only works if people:
 - Identify the **M**
 - Actually bother to **R**

Learning is hard

- Emacs has so much to it it's easy to feel lost
- RTFM^(👉) only works if people:
 - Identify the *M*
 - Actually bother to *R*
- Remember the useful bits

Interactivity is good

There is ample evidence that supports true interactivity, both in the interface and in the presentation methodology, will further enhance learning and knowledge retention among students.

— Ibrahim and Al-Shara (2007)

Students using the [interactive] system outperformed those using the [non-interactive] system in the problem-solving test, and needed less time to complete both memory and problem-solving tests. This result is consistent with the hypothesis that interactive systems facilitate deep learning by actively engaging the learner in the learning process.

— Evans and Gibbons (2007)

Interactivity is good, pt.2

It was found that interactivity had a significant effect on the computer's social presence, its social attraction to children and children's involvement, and intrinsic motivation. The findings suggest that enhancing the interactivity of an e-learning environment can stimulate the presence of social actors, which in turn can enrich a children's learning experience and increase their [intrinsic] motivation.

— Tung and Deng (2006)

Animation interactivity impacted students' improvement on understanding ($p = .006$) and lower-level applying ($p = .042$), and 2) animation interactivity did not significantly impact student confidence and program perception.

— Wang, Vaughn, and Liu (2011)

How a system is interactive matters a lot

There is ample evidence that supports true interactivity, both in the interface and in the presentation methodology, will further enhance learning and knowledge retention among students. (Ibrahim and Al-Shara 2007)

... The real challenge is how to implement it

Resources

- Evans, Chris, and Nicola J. Gibbons. 2007. "The Interactivity Effect in Multimedia Learning." *Computers & Education* 49 (4): 1147–60. <https://doi.org/https://doi.org/10.1016/j.compedu.2006.01.008>.
- Ibrahim, Mohamed, and Osama Al-Shara. 2007. "Impact of Interactive Learning on Knowledge Retention." In *Human Interface and the Management of Information. Interacting in Information Environments*, edited by Michael J. Smith and Gavriel Salvendy, 347–55. Berlin, Heidelberg: Springer Berlin Heidelberg.
- Tung, Fang-Wu, and Yi-Shin Deng. 2006. "Designing Social Presence in E-Learning Environments: Testing the Effect of Interactivity on Children." *Interactive Learning Environments* 14 (3): 251–64. <https://doi.org/10.1080/10494820600924750>.
- Wang, Pei-Yu, Brandon K. Vaughn, and Min Liu. 2011. "The Impact of Animation Interactivity on Novices' Learning of Introductory Statistics." *Computers & Education* 56 (1): 300–311. <https://doi.org/https://doi.org/10.1016/j.compedu.2010.07.011>.

Designing the Doom Tutorial

Inspiration: codecademy

Learn

HELLO WORLD

Print

Now what we're going to do is teach our computer to communicate. The gift of speech is valuable: a computer can answer many questions we have about "how" or "why" or "what" it is doing. In Python, the `print()` function is used to tell a computer to talk. The message to be printed should be surrounded by quotes:

```
1 print("Hello World")
```

Run Save Refresh

3/15

Back Next

The anatomy of codecademy

User interaction box

Instructions

The screenshot shows the Codecademy interface for a lesson. It is divided into three main sections: instructions, a code editor, and an output window. The instructions section on the left contains the text 'HELLO WORLD' and a heading 'Print' followed by a paragraph explaining the `print()` function. The code editor in the center shows a single line of Python code: `print("Hello World")`. Below the code editor are buttons for 'Run', a file icon, and a refresh icon. The output window on the right displays 'Hello World'. At the bottom of the interface, there is a progress indicator '3/15' and two buttons labeled 'Back' and 'Next'.

Learn

HELLO WORLD

Print

Now what we're going to do is teach our computer to communicate. The gift of speech is valuable: a computer can answer many questions we have about "how" or "why" or "what" it is doing. In Python, the `print()` function is used to tell a computer to talk. The message to be printed should be surrounded by quotes:

```
1 print("Hello World")
```

Run

3/15

Back Next

Hello World

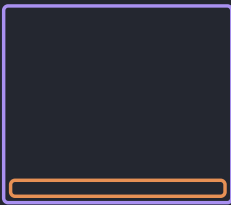
Output

Lesson navigation

The idea for doom tutorial

User interaction box

Instructions

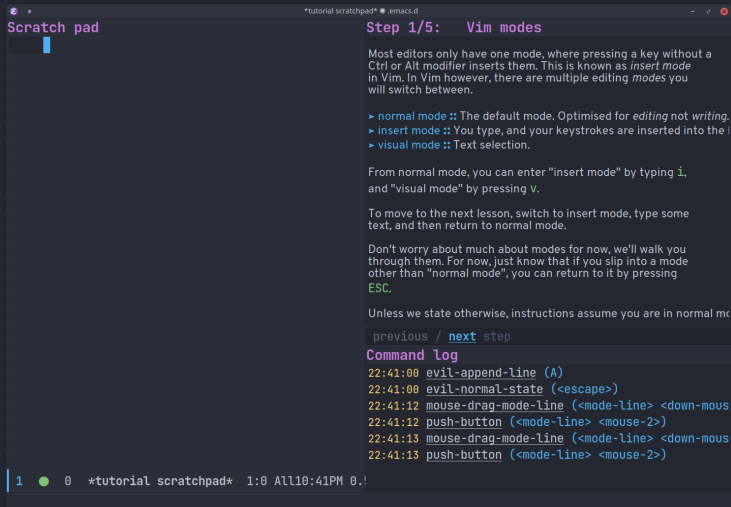


Lesson navigation



Action log

So, how's it turned out?



The screenshot shows an Emacs window titled "Scratch pad" with a cursor at the beginning of the first line. The main content area displays a tutorial titled "Step 1/5: Vim modes". The tutorial text explains that Vim has multiple editing modes, including normal mode (the default), insert mode, and visual mode. It provides instructions on how to switch between these modes: pressing 'i' for insert mode and 'v' for visual mode. A command log at the bottom shows the sequence of events: the user pressed the escape key to enter normal mode, then used the mouse to drag the cursor to the end of the line, and finally pressed the 'i' key to enter insert mode.

```
*tutorial scratchpad* .emacs.d
Scratch pad
Step 1/5: Vim modes

Most editors only have one mode, where pressing a key without a
Ctrl or Alt modifier inserts them. This is known as insert mode
in Vim. In Vim however, there are multiple editing modes you
will switch between.

> normal mode :: The default mode. Optimised for editing not writing.
> insert mode :: You type, and your keystrokes are inserted into the
> visual mode :: Text selection.

From normal mode, you can enter "insert mode" by typing i,
and "visual mode" by pressing v.

To move to the next lesson, switch to insert mode, type some
text, and then return to normal mode.

Don't worry about much about modes for now, we'll walk you
through them. For now, just know that if you slip into a mode
other than "normal mode", you can return to it by pressing
ESC.

Unless we state otherwise, instructions assume you are in normal m

previous / next step

Command log
22:41:00 evil-append-line (A)
22:41:00 evil-normal-state (<escape>)
22:41:12 mouse-drag-mode-line (<mode-line> <down-mous
22:41:12 push-button (<mode-line> <mouse-2>)
22:41:13 mouse-drag-mode-line (<mode-line> <down-mous
22:41:13 push-button (<mode-line> <mouse-2>)

1 ● 0 *tutorial scratchpad* 1:0 All10:41PM 0.!
```

psst. do a demo.

Implementation overview

- `modules/config/tutorial`
 - `config.el`
 - `autoload/tutorial.el`
- `modules/editor/evil/tutorial.el`
- `modules/X/Y/tutorial.el`

Loading tutorials

- Look in every module folder
- If any `tutorial.el` file exists, load it

Tutorials are registered via `doom-tutorial-register`.

A tutorial registry

Tutorials are registered via `doom-tutorial-register`.

This adds them to the `doom-tutorial--registered` alist.

Tutorials are registered via `doom-tutorial-register`.

This adds them to the `doom-tutorial--registered` alist.

Tutorial progress is saved in `doom-tutorial--progress`, which is initialised/saved to the `doom-tutorial-hist-file`.

Writing tutorials

define-tutorial! macro

```
(define-tutorial! name
  "Docstring"
  :triggers modes or functions...
  :setup BODY
  :pages
  (page BODY)...
  :teardown BODY)
```

Diversion — use-package plists to plists

```
(defun doom-tutorial-normalise-plist (somalist)
  (cdr (cl-reduce
        (lambda (result new)
          (if (keywordp new)
              (progn (push new result)
                     (push nil result))
              (push new (car result))))
        result)
        (nreverse somalist)
        :initial-value (list nil))))
```

Page forms

```
(page :title STRINGS...  
      :instructions STRINGS...  
      :template STRINGS...  
      :setup BODY  
      :test BODY)
```

What's next?

Next steps

- Support for sub-tasks
- Support for more actions in the “user pane”, without breaking the setup
- Support for Org tutorials which are converted (using `org-element`) to a `define-tutorial!` macro



That's all Folks!