Contents

1 Literate programming

Literally what?

> Literally explain more, code less. - Literate me

Or more formally:

> the idea that one could create software as works of literature, by embedding source code inside descriptive text, rather than the reverse (as is common practice in most programming languages), in an order that is convenient for exposition to human readers, rather than in the order demanded by the compiler. [2]

```
(org-babel-do-load-languages
 'org-babel-load-languages
 '((shell . t)))
```

The fact that you can write code...

```
echo "...and the get results inserted..."
```

```
> ...and the get results inserted...
```

...directly in the file you write ideas, format and export in any you want gives you the *dis* advantage of having no excuses to let your thoughts flow and document your problem resolution process.[1]

`The program becomes the documentation.`

That being said, start `emacs`

[1] So anyone can read, understand, and replicate. The world would be a beautiful place.

1.1 TODO Exporting

Will address later.

1.1.1 Dependencies on OSX

```
brew install tlmgr Pygments
tlmgr install wrapfig capt-of dvipng latex \
      collection-fontsrecommended censor \
      pbox ifnextok relsize minted \
      fvextra xstring framed
```

1.1.2 Random config to syntax highlight pdfs

```
(setq org-latex-listings 'minted)
(setq org-latex-packages-alist '(("" "minted")))
(add-to-list 'org-latex-minted-langs '(elisp "common-lisp"))
(add-to-list 'org-latex-minted-langs '(purescript "haskell"))
(add-to-list 'org-latex-minted-langs '(sh "bash"))

(setq org-latex-minted-options
      '(("frame" "leftline")
        ("fontsize" "\\scriptsize")
        ("linenos" "")
        ("framerule" "0.4pt")
        ("framesep" "4pt")))
```

2 On purescript

2.1 Setup

A good starting point i found is Literate programming with Monroe and org-mode [3], the org and emacs documentation is good but sometimes i'm in `M-x show-me-the-code-mode` so here's the `org-babel-execute:purescript` function:

I grabbed the `psci` process (line 3) and send (line 5) function from package `psci.el` [1] to avoid the never ending boilerplate of creating my own.

The rest inspired from [3].

Since i don't want to be limited by the version of the tools: compiler, package manager, etc, i don't have `spago`, `purescript` or `bower` installed globally (except maybe `psvm`).

This means the path to the executable needs to be specified

```
(setq psci/spago-path "./node_modules/.bin/spago")
```

`./node_modules/.bin/spago`

```elisp
(defun org-babel-execute:purescript (body params)
  (setq output "")
  (-if-let (process (get-buffer-process (psci--process-name psci/buffer-name)))
      (progn
        (psci--run-psci-command! body)
        (accept-process-output process 0.5 nil t)
        (add-hook 'comint-preoutput-filter-functions 'collect)
        (while (not (end-p output))
          (accept-process-output process 0.5 nil t))
        (remove-hook 'comint-preoutput-filter-functions 'collect)
        (clean output))
    "REPL is not running"))
```

Listing 1: This is the function run when `C-c C-c` *ing*

Remember to install first!

```
#npm ci
npm run deps
```

```
> applicative-parser-combinators@1.0.0 deps /Users/lalo/dev/notes/applicative-parser-co
> spago install

Installation complete.
```

Then you can just run `M-x psci` and be able to evaluates code directly from emacs org-mode. (`C-c C-c`)

```
:h
```

```
The following commands are available:

    :?                      Show this help menu
    :quit                   Quit PSCi
    :reload                 Reload all imported modules while discarding bindings
    :clear                  Discard all imported modules and declared bindings
    :browse      <module>   See all functions in <module>
    :type        <expr>     Show the type of <expr>
    :kind        <type>     Show the kind of <type>
    :show        import     Show all imported modules
    :show        loaded     Show all loaded modules
    :show        print      Show the repl's current printing function
    :paste       paste      Enter multiple lines, terminated by ^D
    :complete    <prefix>   Show completions for <prefix> as if pressing tab
```

```
    :print       <fn>         Set the repl's printing function to <fn> (which must be
Further information is available on the PureScript documentation repository:
 --> https://github.com/purescript/documentation/blob/master/guides/PSCi.md
```

Something interesting to note is that unlike originals `psci` load commands, this doesn't wrap your code in `import module`, which means you can send whatever you want to the repl, `:clear`, `:reload`????

Here's the rest of the code which i find trivial and unrelated to the subject, just helpers for the main function.

```
(setq prompt "^> $")

(defun prompt-p (text)
  "Returns `t' if text matches `prompt'"
  (string-match-p prompt text))

(defun empty-p (text)
  "Returns `t' if `text' is empty or nil"
  (= (length text) 0))

(defun collect (value)
  "Append `value' to `output'

Don't do anything to the output here, just redirect it.
"
  (setq output (concat output value))
  value)

(defvar output "" "Last execution output")

(defun end-p (output)
  "Returns `t' if `output' has ended"
  (let ((lines (reverse (s-split "\n" output))))
    (prompt-p (car lines))))

(defun clean (output)
  "Removes `prompt' from `output' along with empty-lines"
  (let ((lines (reverse (s-split "\n" output))))
    (while (and (not (null (car lines)))
                (or
                 (prompt-p (car lines))
                 (empty-p (car lines))))
      (setq lines (cdr lines)))
    (s-join "\n" (reverse lines))))
```

2.2 TODO Applicative parser combinators

With the setup covered, i'll jump right into the codedumimplentation[2] and try to uncover knowledge by experimentation.

`Disclaimer: i have no idea what i'm doing`

2.2.1 Defining the domain of the problem:

Theres a library called fast-check on TypeScript that allows you to do property based testing,a kind of generative testing inwhich you just provide the types you need and data is generated for you to test on your model and see that it checks.

From its `README`:

> Property based testing frameworks check the truthfulness of properties. A property is a statement like: for all (x, y, ...) such as precondition(x, y, ...) holds property(x, y, ...) is true.

This is good, except for the fact that it's too verbose (the code below)

```
describe('createArray', () => {
  it('should always produce an array taking care of settings', () =>
     fc.assert(
       fc.property(
         fc.record(
           {
             minimum_size: fc.nat(100),
             maximum_size: fc.nat(100)
           },
           { withDeletedKeys: true }
         ),
         settings => {
           const out = createArray(() => 0, settings);
           if (settings.minimum_size != null) assert.ok(out.length >= settings.minimum_size);
           if (settings.maximum_size != null) assert.ok(out.length <= settings.maximum_size);
         }
       )
     ));
});
```

Here's how it looks if we remove `fast=check`

[2] A mix of code, documentation, implementation. And bugs implicit from the fact that *this is a test*

```
describe('createArray', () => {
  it('should always produce an array taking care of settings', settings => {
    const out = createArray(() => 0, settings);
    if (settings.minimum_size != null) assert.ok(out.length >= settings.minimum_size);
    if (settings.maximum_size != null) assert.ok(out.length <= settings.maximum_size);
  });
});
```

From 18 to 6 lines .

I'm already typing my code with typescript. It would make sense to use the same data i declared to avoid all the hassle of writing it twice.

The problem is typescript's `typelevel` information does not live on the same domain as the code and it is unaccesible from the instances and runtime in which it lives.

The data either needs to be lifted to type level, or the types downcasted to data?

... cont

References

[1] Spacemacs develop. Purescript layer. Last accessed Sun Oct 27 04:53:51 2019.

[2] Wikipedia. WEB — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=WEB&oldid=920021210`, 2019. [Online; accessed 27-October-2019].

[3] Sanel Z. Literate programming with monroe and org-mode. Last accessed Sun Oct 27 05:05:58 2019.