Contents

1 TODO Emacs config

TODO: Fix the following

1. LSP Broken
2. Figure out how to properly set up dictionary
3. Org mode load times absurdly long
   (a) The code blocks / stuff I configured for org modern, and the org bigger headlines aren't at all working, likely something in my cfg BEFORE that is breaking it
   (b) Furthermore, look into why the Latex fragments aren't correctly rendering the trees (as seen in the)Predicate Logic notes
   (c) lsp in org src blocks still non-existent
   (d) Doom modeline not working
4. Configure the symbols that you want to use properly
5. Set up ORG-TOC, or find some sort of outline view that I can use to jump around headings of a larger org file like this
6. fix the minimap, cause it looks like ass and has linespacing that is wayyyy too large
7. See if I can rebind the stuff that is under ctrl-h, ctrl-j, c-k, and c-l to be more like my vim config, where those keybinds move you to the different views you have open if your window is split

Doom Emacs Configuration
another helpful config I found recently: another one

1.1 Leftover block from the original config.el file, has some helpful reminders so I'll keep it

```
;; Whenever you reconfigure a package, make sure to wrap your config in an
;; `after!' block, otherwise Doom's defaults may override your settings. E.g.
;;
;;   (after! PACKAGE
;;     (setq x y))
;;
;; The exceptions to this rule:
;;
;;   - Setting file/directory variables (like `org-directory')
;;   - Setting variables which explicitly tell you to set them before their
;;     package is loaded (see 'C-h v VARIABLE' to look up their documentation).
;;   - Setting doom variables (which start with 'doom-' or '+').
;;
;; Here are some additional functions/macros that will help you configure Doom.
;;
```

```
;; - `load!' for loading external *.el files relative to this one
;; - `use-package!' for configuring packages
;; - `after!' for running code after a package has loaded
;; - `add-load-path!' for adding directories to the `load-path', relative to
;;   this file. Emacs searches the `load-path' when you load packages with
;;   `require' or `use-package'.
;; - `map!' for binding new keys
;;
;; To get information about any of these functions/macros, move the cursor over
;; the highlighted symbol at press 'K' (non-evil users must press 'C-c c k').
;; This will open documentation for it, including demos of how they are used.
;; Alternatively, use `C-h o' to look up a symbol (functions, variables, faces,
;; etc).
;;
;; You can also try 'gd' (or 'C-c c d') to jump to their definition and see how
;; they are implemented.
```

2 Config Boilerplate

2.1 Telling Emacs who I am (they are in my skin)

```
;; Some functionality uses this to identify you, e.g. GPG configuration, email
;; clients, file templates and snippets. It is optional.
 (setq user-full-name "Liam Wirth"
       user-mail-address "ltwirth@asu.edu")


;; If you use `org' and don't want your org files in the default location
↪  below,
;; change `org-directory'. It must be set before org loads!
(setq org-directory "~/org/")
;; I've been on-and off trying to use the org agenda, and i like the ideas of
↪  org-roam-daily as a way to quickly make/maintain daily notes.
;; I thought to myself "why not try to combine the two?"
(setq org-agenda-files '("~/org/roam/daily/"))
```

2.1.1 Cache and such?

```
(unless (file-exists-p (expand-file-name "persp" doom-cache-dir))
  (make-directory (expand-file-name "persp/" doom-cache-dir) t))
(defun my/persp-save-session-with-name (name)
  "save the current session with a specified NAME."
  (interactive "sEnter session name: ")
  (persp-save-state-to-file (concat persp-save-dir name)))



(after! persp-mode)
  ;;by default persp save dir is .config/emacs/.local/etc/workspaces I'm chill
  ↪  w/ that
```

2.1.2 Epic Sauce Defaults

```
(setq undo-limit 80000000                         ; Raise undo-limit to 80Mb
      evil-want-fine-undo t                       ; By default while in insert
      ↪  all changes are one big blob. Be more granular
      auto-save-default t                         ; Nobody likes to loose work,
      ↪  I certainly don't
      truncate-string-ellipsis "..."                ; Unicode ellispis are
      ↪  nicer than "...", and also save /precious/ space
      password-cache-expiry nil                   ; I can trust my computers
      ↪  ... can't I?
      scroll-preserve-screen-position 'always     ; Don't have `point' jump
      ↪  around
      scroll-margin 2                             ; It's nice to maintain a
      ↪  little margin
      display-time-default-load-average nil       ; I don't think I've ever
      ↪  found this useful
      display-line-numbers-type 'relative         ; RelNum ON TOP
      )


(display-time-mode 1)                             ; Enable time in the
↪  mode-line
(global-subword-mode 1)                           ; Iterate through CamelCase
↪  words
(pixel-scroll-precision-mode t)                   ; Turn on pixel scrolling



(setq-default
 delete-by-moving-to-trash t                      ; Delete files to trash
 window-combination-resize t                      ; take new window space from
 ↪  all other windows (not just current)
 x-stretch-cursor t                               ; Stretch cursor to the glyph
 ↪  width
 show-paren-mode 1                                ; Highlight Matching
 ↪  Parenthesis
 abbrev-mode t                                    ; erm..
)
```

Its stupid to me that doom wont start in fullscreen by default

```
(add-to-list 'default-frame-alist '(width . 92))
(add-to-list 'default-frame-alist '(height . 40))
```

if you for some unholy reason want to split the window to the left, or above when there's only one window open, you're a psychopath here, I am setting things up to NOT do that

```
(setq evil-vsplit-window-right t
      evil-split-window-below t)
```

frame title stuff, stolen straight from that second config thing I found, looks interesting, might keep, might not

```
(setq frame-title-format
      '(""
        (:eval
         (if (s-contains-p org-roam-directory (or buffer-file-name ""))
             (replace-regexp-in-string
              ".*/[0-9]*-?" " "
              (subst-char-in-string ?_ ?  buffer-file-name))
           "%b"))
        (:eval
         (let ((project-name (projectile-project-name)))
           (unless (string= "-" project-name)
             (format (if (buffer-modified-p)  "  %s" "  %s") project-name))))))
```

2.1.3 Setting up the custom-file (to be used sparingly)

```
(setq-default custom-file (expand-file-name ".custom.el" doom-private-dir))
(when (file-exists-p custom-file)
  (load custom-file))
```

t

Prompting for which buffer to open just a nice little QOL thing

```
(defadvice! prompt-for-buffer (&rest _)
  :after '(evil-window-split evil-window-vsplit)
  (consult-buffer))
```

2.1.4 Window Rotation

good ol keybinds and such

```
(map! :map evil-window-map
      "SPC" #'rotate-layout
      ;; Navigation
      "<left>"     #'evil-window-left
      "<down>"     #'evil-window-down
      "<up>"       #'evil-window-up
      "<right>"    #'evil-window-right
      ;; Swapping windows
      "C-<left>"       #'+evil/window-move-left
      "C-<down>"       #'+evil/window-move-down
      "C-<up>"         #'+evil/window-move-up
      "C-<right>"      #'+evil/window-move-right)

(setq evil-vsplit-window-right t
      evil-split-window-below t)
(defadvice! prompt-for-buffer (&rest _)
  :after '(evil-window-split evil-window-vsplit)
  (consult-buffer))
```

2.2 Hippie Expand stuff

```
(global-set-key [remap dabbrev-expand] #'hippie-expand)
(setq hippie-expand-try-functions-list
      '(try-complete-file-name-partially
        try-complete-file-name
        try-expand-all-abbrevs
        try-expand-list
        try-expand-dabbrev
        try-expand-dabbrev-all-buffers
        try-expand-dabbrev-from-kill
        try-expand-line
        try-complete-lisp-symbol-partially
        try-complete-lisp-symbol))
```

3 Looks

I intend to configure the overal look and feel of my emacs configuration here, as well as any packages/modules that would affect how it looks and feels I.E

To See stuff responsible for org mode look and feel, see:

3.1 Font!

```
;; Doom exposes five (optional) variables for controlling fonts in Doom:
;;
;; - `doom-font' -- the primary font to use
;; - `doom-variable-pitch-font' -- a non-monospace font (where applicable)
;; - `doom-big-font' -- used for `doom-big-font-mode'; use this for
;;   presentations or streaming.
;; - `doom-symbol-font' -- for symbols
;; - `doom-serif-font' -- for the `fixed-pitch-serif' face

(set-face-attribute 'default nil
                    :font "JetBrains Mono NerdFont"
                    :height 140
                    :weight 'medium)
(set-face-attribute 'variable-pitch nil
                    :font "Overpass"
                    :height 120
                    :weight 'medium)
(set-face-attribute 'fixed-pitch nil
                    :font "JetBrains Mono"
                    :height 120
                    :weight 'medium);; This is working in emacsclient but not
                    ↪  emacs.
;; Your font must have an italic face available.
(set-face-attribute 'font-lock-comment-face nil
                    :slant 'italic)
;;(set-face-attribute 'font-lock-keyword-face nil
;; :slant 'italic)
(set-face-attribute 'doom-serif-font (font-spec :family "IBM Plex Mono" :size
↪  22 :weight 'light))
(set-face-attribute 'doom-symbol-font (font-spec :family "JuliaMono"))
```

```
(add-to-list 'default-frame-alist '(font . "JetBrains Mono-15"))

(setq-default line-spacing 0.05)
```

[? :?[= + >] 0 font-shape-gstring]

3.2 Theme!

```
(setq doom-theme 'doom-gruvbox
      doom-themes-treemacs-enable-variable-pitch nil)
```

3.3 Action! Variables relevant to look and feel!

```
(blink-cursor-mode -1)
(column-number-mode t)
(transient-mark-mode t)
```

3.4 Doom Modeline

```
(after! doom-modeline
  (setq doom-modeline-enable-word-count t)
  (setq doom-modeline-icon t)
  (setq doom-modeline-persp-name t)
  (setq doom-modeline-height 45)
  (setq doom-modeline-lsp-icon t)
  (setq doom-modeline-total-line-number t)
  (setq doom-modeline-lsp t)
  (setq doom-modeline-modal-icon t)
  (setq doom-modeline-modal-modern-icon t)
  (setq doom-modeline-battery t)
  (setq doom-modeline-time t)
  (setq doom-modeline-env-version t)
  (setq doom-modeline-time-clock-size 0.65)
  ;;(setq      doom-modeline-hud nil)
  (setq      doom-themes-padded-modeline t)
  (add-hook! 'doom-modeline-mode-hook
    (progn
      (set-face-attribute 'header-line nil
                          :background (face-background 'mode-line)
                          :foreground (face-foreground 'mode-line))
      ))
  )
```

3.5 Startup Screen

this one is gonna be pretty long to configure, but it's largely cause I've lifted a hefty chunk of code that will add silly/stupid splash phrases to the startup screen

3.5.1 Splash-Phrase Tomfoolery

Setting Up The Source Folder

```
(defvar splash-phrase-source-folder
  (expand-file-name "misc/splash-phrases" doom-private-dir)
  "A folder of text files with a fun phrase on each line.")
```

Actually getting the splash-phrases from the source folder

```
(defvar splash-phrase-sources
  (let* ((files (directory-files splash-phrase-source-folder nil "\\.txt\\'"))
         (sets (delete-dups (mapcar
                             (lambda (file)
                               (replace-regexp-in-string
                               ↪  "\\(?:-[0-9]+-\\w+\\)?\\.txt" "" file))
                             files))))
    (mapcar (lambda (sset)
              (cons sset
                    (delq nil (mapcar
                               (lambda (file)
                                 (when (string-match-p (regexp-quote sset)
                                 ↪  file)
                                   file))
                               files))))
            sets))
  "A list of cons giving the phrase set name, and a list of files which contain
  ↪  phrase components.")
```

No Caching Here

Some Functions

```
(defvar splash-phrase--cached-lines nil)
```

Randomly choosing the Splash Phrase (and making some custom variable hold that value)

```
(defvar splash-phrase-set
  (nth (random (length splash-phrase-sources)) (mapcar #'car
  ↪  splash-phrase-sources))
  "The default phrase set. See `splash-phrase-sources'.")
```

Picking A Set From The Folder

```
(defun splash-phrase-set-random-set ()
  "Set a new random splash phrase set."
  (interactive)
  (setq splash-phrase-set
        (nth (random (1- (length splash-phrase-sources)))
             (cl-set-difference (mapcar #'car splash-phrase-sources) (list
             ↪  splash-phrase-set)))))
```

```
  (+doom-dashboard-reload t))
```

or allowing the user to choose one explicitly:

```
(defun splash-phrase-select-set ()
  "Select a specific splash phrase set."
  (interactive)
  (setq splash-phrase-set (completing-read "Phrase set: " (mapcar #'car
  ↪  splash-phrase-sources)))
  (+doom-dashboard-reload t))
```

Getting One From A File

```
(defun splash-phrase-get-from-file (file)
  "Fetch a random line from FILE."
  (let ((lines (or (cdr (assoc file splash-phrase--cached-lines))
                   (cdar (push (cons file
                                     (with-temp-buffer
                                       (insert-file-contents (expand-file-name
                                       ↪  file splash-phrase-source-folder))
                                       (split-string (string-trim
                                       ↪  (buffer-string)) "\n")))
                               splash-phrase--cached-lines)))))
    (nth (random (length lines)) lines)))
```

Getting The Phrase

```
(defun splash-phrase (&optional set)
  "Construct a splash phrase from SET. See `splash-phrase-sources'."
  (mapconcat
   #'splash-phrase-get-from-file
   (cdr (assoc (or set splash-phrase-set) splash-phrase-sources))
   " "))
```

Making it all Look Pretty

```
(defun splash-phrase-dashboard-formatted ()
  "Get a splash phrase, flow it over multiple lines as needed, and fontify it."
  (mapconcat
   (lambda (line)
     (+doom-dashboard--center
      +doom-dashboard--width
      (with-temp-buffer
        (insert-text-button
         line
         'action
         (lambda (_) (+doom-dashboard-reload t))
         'face 'doom-dashboard-menu-title
         'mouse-face 'doom-dashboard-menu-title
         'help-echo "Random phrase"
         'follow-link t)
        (buffer-string))))
   (split-string
    (with-temp-buffer
```

```
      (insert (splash-phrase))
      (setq fill-column (min 70 (/ (* 2 (window-width)) 3)))
      (fill-region (point-min) (point-max))
      (buffer-string))
    "\n")
   "\n"))
```

Inserting the (now beautified) Splash Phrase into the Dashboard

```
(defun splash-phrase-dashboard-insert ()
  "Insert the splash phrase surrounded by newlines."
  (insert "\n" (splash-phrase-dashboard-formatted) "\n"))
```

3.5.2 TODO Configuring the Dashboard

uhhh.... uhhhhmmmmm

3.6 Centaur Tabs

I still have little to no clue how emacs window management works

```
(after! centaur-tabs

  (setq centaur-tabs-height 36
        centaur-tabs-set-icons t
        centaur-tabs-modified-marker "o"
        centaur-tabs-close-button "×"
        centaur-tabs-set-bar 'above
        centaur-tabs-gray-out-icons 'buffer)
  )
```

3.7 Info-Colors

```
(use-package! info-colors
:commands (info-colors-fontify-node))
```

3.8 Transparency Shenaninigans

in newer versions of emacs they seem to have added the functionality to have a transparent window. I typically dont mess around with transparent windows but they can be nice to have on occasion. Here I am going to mess around and see if the functionality is worthwile, as well as possibly add a function that lets me update the value in place/toggle it

```
(defvar my-window-alpha 100
  "I like my window transparency opaque by default")
(defun kb/toggle-window-transparency ()
  "Toggle transparency."
  (interactive)
  (let ((alpha-transparency 0))
    (pcase (frame-parameter nil 'alpha-background)
```

```
      (alpha-transparency (set-frame-parameter nil 'alpha-background 100))
      (t (set-frame-parameter nil 'alpha-background alpha-transparency)))))
(global-set-key (kbd "<f12>") 'kb/toggle-window-transparency)
```

4 Configuring Plugins (Misc)

4.1 Which-Key

it's like the one from neovim? (or is it the other way around?)

```
(after! which-key
  (setq which-key-idle-delay 0.2))

(after! which-key
  (pushnew!
   which-key-replacement-alist
   '(("" . "\\`+?evil[-:]?\\(?:a-\\)?\\(.*\\)") . (nil . "\\1"))
   '(("\\`g s" . "\\`evilem--?motion-\\(.*\\)") . (nil . "\\1"))
   ))
(setq which-key-allow-multiple-replacements t)
```

4.2 TODO Elcord

everyone MUST KNOW I'M USING EMACS (as well as vim)

```
(use-package! elcord
  :commands elcord-mode
  :config
  (setq elcord-use-major-mode-as-main-icon t))
```

4.3 Mixed Pitch

4.3.1 Setting the Variable-Pitch Serrif Font

```
(defface variable-pitch-serif
    '((t (:family "serif")))
    "A variable-pitch face with serifs."
    :group 'basic-faces)

(defcustom variable-pitch-serif-font (font-spec :family "serif")
  "The font face used for `variable-pitch-serif'."
  :group 'basic-faces
  :type '(restricted-sexp :tag "font-spec" :match-alternatives (fontp))
  :set (lambda (symbol value)
         (set-face-attribute 'variable-pitch-serif nil :font value)
         (set-default-toplevel-value symbol value)))
```

(lifted straight from the tecosaur config)

```
;;(defvar mixed-pitch-modes '(org-mode LaTeX-mode markdown-mode gfm-mode
↪  Info-mode)
;;  "Modes that `mixed-pitch-mode' should be enabled in, but only after UI
↪  initialisation.")
;;(defun init-mixed-pitch-h ()
;;  "Hook `mixed-pitch-mode' into each mode in `mixed-pitch-modes'.
;;Also immediately enables `mixed-pitch-modes' if currently in one of the
↪  modes."
;;  (when (memq major-mode mixed-pitch-modes)
;;    (mixed-pitch-mode 1))
;;  (dolist (hook mixed-pitch-modes)
;;    (add-hook (intern (concat (symbol-name hook) "-hook"))
↪  #'mixed-pitch-mode)))
;;(add-hook 'doom-init-ui-hook #'init-mixed-pitch-h)
;;
;;(autoload #'mixed-pitch-serif-mode "mixed-pitch"
;;  "Change the default face of the current buffer to a serifed variable pitch,
↪  while keeping some faces fixed pitch." t)
;;
;;(setq! variable-pitch-serif-font (font-spec :family "Alegreya" :size 27))
;;
;;(after! mixed-pitch
;;  (setq mixed-pitch-set-height t)
;;  (set-face-attribute 'variable-pitch-serif nil :font
↪  variable-pitch-serif-font)
;;  (defun mixed-pitch-serif-mode (&optional arg)
;;    "Change the default face of the current buffer to a serifed variable
↪  pitch, while keeping some faces fixed pitch."
;;    (interactive)
;;    (let ((mixed-pitch-face 'variable-pitch-serif))
;;      (mixed-pitch-mode (or arg 'toggle)))))
;;
;;(set-char-table-range composition-function-table ?f '(["\\(?:ff?[fijlt]\\)" 0
↪  font-shape-gstring]))
;;(set-char-table-range composition-function-table ?T '(["\\(?:Th\\)" 0
↪  font-shape-gstring]))
```

4.4 Persp Mode

This is largely for session management

4.5 WriteWroom

```
(defvar +zen-serif-p t
  "Whether to use a serifed font with `mixed-pitch-mode'.")
(defvar +zen-org-starhide t
  "The value `org-modern-hide-stars' is set to.")

(after! writeroom-mode
  (defvar-local +zen--original-org-indent-mode-p nil)
  (defvar-local +zen--original-mixed-pitch-mode-p nil)
  (defun +zen-enable-mixed-pitch-mode-h ()
    "Enable `mixed-pitch-mode' when in `+zen-mixed-pitch-modes'."
    (when (apply #'derived-mode-p +zen-mixed-pitch-modes)
```

```
    (if writeroom-mode
        (progn
          (setq +zen--original-mixed-pitch-mode-p mixed-pitch-mode)
          (funcall (if +zen-serif-p #'mixed-pitch-serif-mode
          ↪  #'mixed-pitch-mode) 1))
      (funcall #'mixed-pitch-mode (if +zen--original-mixed-pitch-mode-p 1
      ↪  -1)))))
(defun +zen-prose-org-h ()
  "Reformat the current Org buffer appearance for prose."
  (when (eq major-mode 'org-mode)
    (setq display-line-numbers nil
          visual-fill-column-width 60
          org-adapt-indentation nil)
    (when (featurep 'org-modern)
      (setq-local org-modern-star '("" "" "" "")
                  ;; org-modern-star '("" "" "" "" "" "" "" "")
                  org-modern-hide-stars +zen-org-starhide)
      (org-modern-mode -1)
      (org-modern-mode 1))
    (setq
     +zen--original-org-indent-mode-p org-indent-mode)
    (org-indent-mode -1)))
(defun +zen-nonprose-org-h ()
  "Reverse the effect of `+zen-prose-org'."
  (when (eq major-mode 'org-mode)
    (when (bound-and-true-p org-modern-mode)
      (org-modern-mode -1)
      (org-modern-mode 1))
    (when +zen--original-org-indent-mode-p (org-indent-mode 1))))
(pushnew! writeroom--local-variables
          'display-line-numbers
          'visual-fill-column-width
          'org-adapt-indentation
          'org-modern-mode
          'org-modern-star
          'org-modern-hide-stars)
(add-hook 'writeroom-mode-enable-hook #'+zen-prose-org-h)
(add-hook 'writeroom-mode-disable-hook #'+zen-nonprose-org-h))
```

5 Org

the swag

5.1 Hooks

5.1.1 Org Modern Hooks

the key to it looking **pretty**

```
(after! org
  (use-package! org-modern
 :config
 (setq
  org-special-ctrl-a/e t
```

```
org-insert-heading-respect-content t
;; appearance
org-modern-radio-target    '("" t "")
org-modern-internal-target '(" " t "") ; TODO: make this not be an emoji, and
↪  instead a font lig
org-modern-todo t
org-modern-todo-faces
'(("TODO" :inverse-video t :inherit org-todo)
 ("PROJ" :inverse-video t :inherit +org-todo-project)
 ("STRT" :inverse-video t :inherit +org-todo-active)
 ("[-]"  :inverse-video t :inherit +org-todo-active)
 ("HOLD" :inverse-video t :inherit +org-todo-onhold)
 ("WAIT" :inverse-video t :inherit +org-todo-onhold)
 ("[?]"  :inverse-video t :inherit +org-todo-onhold)
 ("KILL" :inverse-video t :inherit +org-todo-cancel)
 ("NO"   :inverse-video t :inherit +org-todo-cancel))
org-modern-footnote (cons nil (cadr org-script-display))
 org-modern-block-name
 '((t . t)
   ("src" "»" "«")
   ("example" "»-" "-«")
   ("quote" "" "")
   ("export" "" ""))
 org-modern-priority nil
 org-modern-progress nil
 ; org-modern-horizontal-rule (make-string 36 ?)
 org-modern-horizontal-rule ""
; org-modern-hide-stars "·"
 org-modern-star '("" "" "" "" "" "" "" "")
      org-modern-keyword
      '((t . t)
        ("title" . "")
        ("subtitle" . "")
        ("author" . "")
        ("email" . #("" 0 1 (display (raise -0.14))))
        ("date" . "")
        ("property" . "")
        ("options" . "")
        ("startup" . "")
        ("macro" . "")
        ("bind" . #("" 0 1 (display (raise -0.1))))
        ("bibliography" . "")
        ("print_bibliography" . #("" 0 1 (display (raise -0.1))))
        ("cite_export" . "")
        ("print_glossary" . #("" 0 1 (display (raise -0.1))))
        ("glossary_sources" . #("" 0 1 (display (raise -0.14))))
        ("include" . "")
        ("setupfile" . "")
        ("html_head" . "")
        ("html" . "")
        ("latex_class" . "")
        ("latex_class_options" . #("" 1 2 (display (raise -0.14))))
        ("latex_header" . "")
        ("latex_header_extra" . "")
        ("latex" . "")
        ("beamer_theme" . "")
        ("beamer_color_theme" . #("" 1 2 (display (raise -0.12))))
```

```
          ("beamer_font_theme" . "")
          ("beamer_header" . "")
          ("beamer" . "")
          ("attr_latex" . "")
          ("attr_html" . "")
          ("attr_org" . "")
          ("call" . #("" 0 1 (display (raise -0.15))))
          ("name" . "")
          ("header" . ">")
          ("caption" . "")
          ("results" . "")))
  (custom-set-faces! '(org-modern-statistics :inherit
  ↪  org-checkbox-statistics-todo)))

(after! org (add-hook 'org-mode-hook #'org-modern-mode))
)
```

5.2 Org Variables

```
(use-package! org
:config
(setq org-fontify-quote-and-verse-blocks t
org-highlight-latex-and-related '(native script entities)
org-list-demote-modify-bullet '(("+" . "-") ("-" . "+") ("*" . "+") ("1." .
↪  "a.")))
;(setq org-export-directory "~/org/exported")

(require 'org-src)
(add-to-list 'org-src-block-faces '("latex" (:inherit default :extend t))))
  (custom-set-faces!
    `((org-quote)
      :foreground ,(doom-color 'blue) :extend t)
    `((org-block-begin-line org-block-end-line)
      :background ,(doom-color 'bg)))
  ;; Change how LaTeX and image previews are shown
  (setq org-highlight-latex-and-related '(native entities script)
        org-image-actual-width (min (/ (display-pixel-width) 3) 800))
```

5.3 Look and Feel

5.3.1 Custom Faces

Using medium weights and stuff for our headers, as well as making them larger

```
(after! org-mode
  (custom-set-faces!
    '((org-document-title)
      :foreground ,(face-attribute 'org-document-title :foreground)
      :height 2.0
      :weight bold
      )
    '((org-level-1)
```

```
      :height 1.7
      :weight medium
      :foreground ,(face-attribute 'outline-1 :foreground)
      )
    '((org-level-2)
      :height 1.6
      :weight medium
      :foreground ,(face-attribute 'outline-2 :foreground)
      )
    '((org-level-3)
      :height 1.5
      :weight medium
      :foreground ,(face-attribute 'outline-3 :foreground)
      )
    '((org-level-4)
      :height 1.4
      :weight medium
      :foreground ,(face-attribute 'outline-4 :foreground)
      )
    '((org-level-5)
      :height 1.3
      :weight medium
      :foreground ,(face-attribute 'outline-5 :foreground)
      )
    '((org-level-6)
      :height 1.2
      :weight medium
      :foreground ,(face-attribute 'outline-6 :foreground)
      )
    '((org-level-7)
      :height 1.1
      :weight medium
      :foreground ,(face-attribute 'outline-7 :foreground)
      )
    ))
```

5.3.2 Org-Ellipsis

```
(after! org
(setq org-ellipsis "  ")
(setq org-hide-leading-stars t)
(setq org-priority-highest ?A)
(setq org-priority-lowest ?E)
(setq org-priority-faces
      '((?A . 'nerd-icons-red)          ;
        (?B . 'nerd-icons-orange)
        (?C . 'nerd-icons-yellow)
        (?D . 'nerd-icons-green)
        (?E . 'nerd-icons-blue)))


(appendq! +ligatures-extra-symbols
          (list :list_property ""
                :em_dash       "-"
                :ellipses      "..."
                :arrow_right   "→"
```

```
                :arrow_left    "←"
                :arrow_lr      ""
                :properties    ""
                :end           ""
                :priority_a    #("" 0 1 (face nerd-icons-red))
                :priority_b    #("" 0 1 (face nerd-icons-orange))
                :priority_c    #("" 0 1 (face nerd-icons-yellow))
                :priority_d    #("" 0 1 (face nerd-icons-green))
                :priority_e    #("" 0 1 (face nerd-icons-blue)))))
```

5.4 Keybind

5.4.1 TODO General

```
(map! :after org
      :map org-mode-map
      :localleader
      :desc "Org-Mark-Ring jump" "gj" #'org-mark-ring-goto
      )
(map! :after org
      :map org-mode-map
      :localleader
      :desc "Org-Mark-Ring Save" "gs" #'org-mark-ring-push)
```

5.4.2 A silly little keybind idea

open up a custom little swag baby gangster type thing whenever I hit a keybind while in a src block to enter a temp buffer

```
(defun open-temp-buffer-src ()
"Open Temporary Buffer When Editing Src Blocks"
(interactive)
(org-edit-src-code)
)
```

```
(map! :after org
      :map org-mode-map
      :localleader
      :desc "Org Set Property" "O" #'org-set-property)
(map! :after org
      :map org-mode-map
      :localleader
      :n "o" #'org-edit-src-code)
```

5.4.3 Spell-Fu

```
(after! spell-fu
  (cl-pushnew 'org-modern-tag (alist-get 'org-mode
  ↪  +spell-excluded-faces-alist)))
```

5.4.4 Org-Appear

```
(use-package! org-appear
  :hook (org-mode . org-appear-mode)
  :config
  (setq org-hide-emphasis-markers t
        org-appear-autolinks 'just-brackets))
```

5.5 Babel

5.5.1 Default Header Args

```
(setq org-babel-default-header-args
      '((:session . "none")
        (:results . "replace")
        (:exports . "code")
        (:cache . "no")
        (:noweb . "no")
        (:hlines . "yes")
        (:tangle . "yes")
        (:comments . "link")))
```

5.5.2 Load Languages:

```
(org-babel-do-load-languages
 'org-babel-load-languages
 '((dot . t)
   '(emacs-lisp . t)
   '(mips . t)
   '(python . t)
   '(latex . t)
   '(rust . t)
   '(C . t)
   '(cpp . t)))
```

```
(require 'org)
(require 'ob)

(require 'ob-C)
```

5.6 Org-Latex

```
(add-hook 'org-mode-hook 'turn-on-org-cdlatex)
(defadvice! +org-edit-latex-env-after-insert-a (&rest _)
  :after #'org-cdlatex-environment-indent
  (org-edit-latex-environment))
```

```
(setq org-highlight-latex-and-related '(native script entities))
(require 'org-src)
(add-to-list 'org-src-block-faces '("latex" (:inherit default :extend t)))
(setq org-latex-preview-preamble
```

```
      (concat
       "\n% Custom font\n\\usepackage{arev}\n\n"
       ;<<grab("latex-maths-conveniences")>>))
       ))
```

Defining our font size:

```
;; Calibrated based on the TeX font and org-buffer font.
(plist-put org-format-latex-options :zoom 1.93)
(after! org (plist-put org-format-latex-options :scale 2.0))
```

5.7 Org-Roam

```
(after! org
  (setq org-roam-directory  "~/org/roam/")
  (setq org-roam-completion-everywhere t))
```

5.7.1 TODO Modeline something something

```
(defadvice! doom-modeline--buffer-file-name-roam-aware-a (orig-fun)
  :around #'doom-modeline-buffer-file-name ; takes no args
  (if (s-contains-p org-roam-directory (or buffer-file-name ""))
      (replace-regexp-in-string

       ↪  "\\(?:^\\|.*/\\)\\([0-9]\\{4\\}\\)\\([0-9]\\{2\\}\\)\\([0-9]\\{2\\}\\)[0-9]*-"
       "(\\1-\\2-\\3) "
       (subst-char-in-string ?_ ?  buffer-file-name))
    (funcall orig-fun)))
```

5.7.2 Yasssss

I don't understand this plugin as much as I should tbh

Src-Header stuff

```
(defun +yas/org-src-header-p ()
  "Determine whether `point' is within a src-block header or header-args."
  (pcase (org-element-type (org-element-context))
    ('src-block (< (point) ; before code part of the src-block
                   (save-excursion (goto-char (org-element-property :begin
                   ↪  (org-element-context)))
                                   (forward-line 1)
                                   (point))))
    ('inline-src-block (< (point) ; before code part of the inline-src-block
                          (save-excursion (goto-char (org-element-property
                          ↪  :begin (org-element-context)))
                                          (search-forward "]{")
                                          (point))))
    ('keyword (string-match-p "^header-args" (org-element-property :value
    ↪  (org-element-context))))))
```

More Src-Header Stuff

```
(defun +yas/org-prompt-header-arg (arg question values)
  "Prompt the user to set ARG header property to one of VALUES with QUESTION.
The default value is identified and indicated. If either default is selected,
or no selection is made: nil is returned."
  (let* ((src-block-p (not (looking-back "^#\\+property:[ \t]+header-args:.*"
  ↪  (line-beginning-position))))
         (default
          (or
           (cdr (assoc arg
                       (if src-block-p
                           (nth 2 (org-babel-get-src-block-info t))
                         (org-babel-merge-params
                          org-babel-default-header-args
                          (let ((lang-headers
                                 (intern (concat
                                 ↪  "org-babel-default-header-args:"
                                                 (+yas/org-src-lang)))))
                            (when (boundp lang-headers) (eval lang-headers
                            ↪  t))))))))
           ""))
         default-value)
    (setq values (mapcar
                  (lambda (value)
                    (if (string-match-p (regexp-quote value) default)
                        (setq default-value
                              (concat value " "
                                      (propertize "(default)" 'face
                                      ↪  'font-lock-doc-face)))
                      value))
                  values))
    (let ((selection (consult--read values :prompt question :default
    ↪  default-value)))
      (unless (or (string-match-p "(default)$" selection)
                  (string= "" selection))
        selection))))
```

Yas Src Blocks

```
(defun +yas/org-src-lang ()
  "Try to find the current language of the src/header at `point'.
Return nil otherwise."
  (let ((context (org-element-context)))
    (pcase (org-element-type context)
      ('src-block (org-element-property :language context))
      ('inline-src-block (org-element-property :language context))
      ('keyword (when (string-match "^header-args:\\([^ ]+\\)"
      ↪  (org-element-property :value context))
                  (match-string 1 (org-element-property :value context)))))))
```

Last Lang Used

```
(defun +yas/org-last-src-lang ()
  "Return the language of the last src-block, if it exists."
  (save-excursion
    (beginning-of-line)
```

```
    (when (re-search-backward "^[ \t]*#\\+begin_src" nil t)
      (org-element-property :language (org-element-context)))))
```

Most Common Language

```
(defun +yas/org-most-common-no-property-lang ()
  "Find the lang with the most source blocks that has no global header-args,
  ↪  else nil."
  (let (src-langs header-langs)
    (save-excursion
      (goto-char (point-min))
      (while (re-search-forward "^[ \t]*#\\+begin_src" nil t)
        (push (+yas/org-src-lang) src-langs))
      (goto-char (point-min))
      (while (re-search-forward "^[ \t]*#\\+property: +header-args" nil t)
        (push (+yas/org-src-lang) header-langs)))

    (setq src-langs
          (mapcar #'car
                  ;; sort alist by frequency (desc.)
                  (sort
                   ;; generate alist with form (value . frequency)
                   (cl-loop for (n . m) in (seq-group-by #'identity src-langs)
                            collect (cons n (length m)))
                   (lambda (a b) (> (cdr a) (cdr b))))))

    (car (cl-set-difference src-langs header-langs :test #'string=))))
```

5.8 Org-Plot

tecosaur has a nice thing that sets plot to use the same colors

```
(defvar +org-plot-term-size '(1050 . 650)
  "The size of the GNUPlot terminal, in the form (WIDTH . HEIGHT).")

(after! org-plot
  (defun +org-plot-generate-theme (_type)
    "Use the current Doom theme colours to generate a GnuPlot preamble."
    (format "
fgt = \"textcolor rgb '%s'\" # foreground text
fgat = \"textcolor rgb '%s'\" # foreground alt text
fgl = \"linecolor rgb '%s'\" # foreground line
fgal = \"linecolor rgb '%s'\" # foreground alt line

# foreground colors
set border lc rgb '%s'
# change text colors of  tics
set xtics @fgt
set ytics @fgt
# change text colors of labels
set title @fgt
set xlabel @fgt
set ylabel @fgt
# change a text color of key
```

```
set key @fgt

# line styles
set linetype 1 lw 2 lc rgb '%s' # red
set linetype 2 lw 2 lc rgb '%s' # blue
set linetype 3 lw 2 lc rgb '%s' # green
set linetype 4 lw 2 lc rgb '%s' # magenta
set linetype 5 lw 2 lc rgb '%s' # orange
set linetype 6 lw 2 lc rgb '%s' # yellow
set linetype 7 lw 2 lc rgb '%s' # teal
set linetype 8 lw 2 lc rgb '%s' # violet

# border styles
set tics out nomirror
set border 3

# palette
set palette maxcolors 8
set palette defined ( 0 '%s',\
1 '%s',\
2 '%s',\
3 '%s',\
4 '%s',\
5 '%s',\
6 '%s',\
7 '%s' )
"
            (doom-color 'fg)
            (doom-color 'fg-alt)
            (doom-color 'fg)
            (doom-color 'fg-alt)
            (doom-color 'fg)
            ;; colours
            (doom-color 'red)
            (doom-color 'blue)
            (doom-color 'green)
            (doom-color 'magenta)
            (doom-color 'orange)
            (doom-color 'yellow)
            (doom-color 'teal)
            (doom-color 'violet)
            ;; duplicated
            (doom-color 'red)
            (doom-color 'blue)
            (doom-color 'green)
            (doom-color 'magenta)
            (doom-color 'orange)
            (doom-color 'yellow)
            (doom-color 'teal)
            (doom-color 'violet)))

  (defun +org-plot-gnuplot-term-properties (_type)
    (format "background rgb '%s' size %s,%s"
            (doom-color 'bg) (car +org-plot-term-size) (cdr
            ↪  +org-plot-term-size)))

  (setq org-plot/gnuplot-script-preamble #'+org-plot-generate-theme)
```

```
  (setq org-plot/gnuplot-term-extra #'+org-plot-gnuplot-term-properties))
```

5.9 Exporting

5.9.1 Org Export Backends:

yanked this from my .custom thing cause I want it to setup here

```
(after! org
 (setq org-export-backends '(ascii beamer html icalendar latex man md odt))
 )
```

5.9.2 Latex

Compiling

```
(after! org
(use-package! ox-latex
  :config

  ;; Default packages
(setq org-export-headline-levels 8
        org-latex-default-packages-alist
        '(("AUTO" "inputenc" t ("pdflatex" "lualatex"))
          ("T1" "fontenc" t ("pdflatex"))
          ;; Microtype
          ;; - pdflatex: full microtype features, fast, however no fontspec
          ;; - lualatex: good microtype feature support, however slow to
          ↪  compile
          ;; - xelatex: only protrusion support, fast compilation

          ↪  ("activate={true,nocompatibility},final,tracking=true,kerning=true,spacing=true,factor=1100,stretc
           "microtype" nil ("pdflatex")         )

          ↪  ("activate={true,nocompatibility},final,tracking=true,factor=1100,stretch=10,shrink=10"
           "microtype" nil ("lualatex"))

          ↪  ("protrusion={true,nocompatibility},final,factor=1100,stretch=10,shrink=10"
           "microtype" nil ("xelatex"))
          ("dvipsnames,svgnames" "xcolor" nil)  ; Include xcolor package

          ↪  ("headings=optiontoheadandtoc,footings=optiontofootandtoc,headlines=optiontoheadandtoc"
           "scrextend" nil)  ; Include scrextend package
          ("colorlinks=true,  citecolor=BrickRed, urlcolor=DarkGreen"
          ↪  "hyperref" nil))))))
```

```
(after! org
(after! ox
 ;; Additional LaTeX classes
  (after! ox
    (add-to-list 'org-latex-classes
                 '("article"
                   "\\documentclass{article}"
```

```
                 ("\\section{%s}" . "\\section*{%s}")
                 ("\\subsection{%s}" . "\\subsection*{%s}")
                 ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
                 ("\\paragraph{%s}" . "\\paragraph*{%s}")
                 ("\\subparagraph{%s}" . "\\subparagraph*{%s}")))
    (add-to-list 'org-latex-classes
                 '("koma-letter" "\\documentclass[11pt]{scrletter}"
                   ("\\section{%s}" . "\\section*{%s}")
                   ("\\subsection{%s}" . "\\subsection*{%s}")
                   ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
                   ("\\paragraph{%s}" . "\\paragraph*{%s}")
                   ("\\subparagraph{%s}" . "\\subparagraph*{%s}")))
    (add-to-list 'org-latex-classes
                 '("koma-article" "\\documentclass[11pt]{scrartcl}"
                   ("\\section{%s}" . "\\section*{%s}")
                   ("\\subsection{%s}" . "\\subsection*{%s}")
                   ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
                   ("\\paragraph{%s}" . "\\paragraph*{%s}")
                   ("\\subparagraph{%s}" . "\\subparagraph*{%s}")))
    (add-to-list 'org-latex-classes
                 '("koma-report" "\\documentclass[11pt]{scrreprt}"
                   ("\\part{%s}" . "\\part*{%s}")
                   ("\\chapter{%s}" . "\\chapter*{%s}")
                   ("\\section{%s}" . "\\section*{%s}")
                   ("\\subsection{%s}" . "\\subsection*{%s}")
                   ("\\subsubsection{%s}" . "\\subsubsection*{%s}")))
    (add-to-list 'org-latex-classes
                 '("koma-book" "\\documentclass[11pt]{scrbook}"
                   ("\\part{%s}" . "\\part*{%s}")
                   ("\\chapter{%s}" . "\\chapter*{%s}")
                   ("\\section{%s}" . "\\section*{%s}")
                   ("\\subsection{%s}" . "\\subsection*{%s}")
                   ("\\subsubsection{%s}" . "\\subsubsection*{%s}"))))


  ;; Table of contents customization
(after! org
  ;; Customize table of contents style
  (setq org-latex-custom-id '("\\usepackage{tocloft}"
                              "\\setlength{\\cftbeforesecskip}{1ex}"
                              "\\setlength{\\cftbeforesubsecskip}{0.5ex}"
                              "\\setlength{\\cftbeforesubsubsecskip}{0.5ex}")))

(after! org
  ;; Define common style for table of contents
  (setq common-toc-style '("\\usepackage{tocloft}"
                           "\\setlength{\\cftbeforesecskip}{1ex}"
                           "\\setlength{\\cftbeforesubsecskip}{0.5ex}"
                           "\\setlength{\\cftbeforesubsubsecskip}{0.5ex}"
                           ("\\tableofcontents" .
                           ↪  "\\tableofcontents\\thispagestyle{empty}\\vspace*{\\fill}\\clearpage")))
  ;; Apply the common style to all classes
  (dolist (class org-latex-classes)
    (let ((class-name (car class))
          (class-content (cdr class)))
      ;; Append common style to each class content
      (setcdr class (append class-content common-toc-style)))))
```

```
(after! org
  ;; Customize specific class style for table of contents
  (setq org-latex-toc-command "\\tableofcontents\\newpage"))

(after! org
  (add-to-list 'org-latex-classes
        '(("report"
           "\\documentclass{report}"
           ("\\chapter{%s}" . "\\chapter*{%s}")
           ("\\section{%s}" . "\\section*{%s}")
           ("\\subsection{%s}" . "\\subsection*{%s}")
           ("\\subsubsection{%s}" . "\\subsubsection*{%s}")
           ("\\paragraph{%s}" . "\\paragraph*{%s}")
           ("\\subparagraph{%s}" . "\\subparagraph*{%s}"))))))

(after! ox-latex
  (setq org-latex-src-block-backend 'engraved))
```

```
(use-package! ox-chameleon
  :after ox
  :config
  (setq! ox-chameleon-engrave-theme 'doom-gruvbox))
```

trying to get org to export to a default directory by default

```
;(defun org-export-output-file-name-modified (orig-fun extension &optional
↪  subtreep pub-dir)
  "Modifies org-export to place exported files in a custom directory with a
  ↪  subdirectory for each file."
;  (let* ((base-name (file-name-base (buffer-file-name)))
;         (pub-dir (concat "~/org/exported/" base-name "/"))) ; Build path with
↪  subdirectory
;    (unless (file-directory-p pub-dir)
;      (make-directory pub-dir t)) ; Create the directory and parents if needed
;    (apply orig-fun extension subtreep pub-dir nil))) ; Pass the modified
↪  pub-dir

;(advice-add 'org-export-output-file-name :around
↪  #'org-export-output-file-name-modified)

;; Update the org-latex-pdf-process to use the correct output-directory
↪  placeholder
(setq org-latex-pdf-process '("LANGUAGE=en_US.UTF-8 LC_ALL=en_US.UTF-8 latexmk
↪  -f -pdf -%latex -shell-escape -interaction=nonstopmode -output-directory=%o
↪  %f"))
```

Functions? not sure how to categorize these, but all of the following is taken from the tecosaur emacs config

```
(defun +org-export-latex-fancy-item-checkboxes (text backend info)
  (when (org-export-derived-backend-p backend 'latex)
    (replace-regexp-in-string
     "\\\\item\\[{$\\\\\\(\\w+\\)$}\\]"
```

```
    (lambda (fullmatch)
      (concat "\\\\item[" (pcase (substring fullmatch 9 -3) ; content of
      ↪  capture group
                           ("square"   "\\\\checkboxUnchecked")
                           ("boxminus" "\\\\checkboxTransitive")
                           ("boxtimes" "\\\\checkboxChecked")
                           (_ (substring fullmatch 9 -3))) "]"))
    text)))

(add-to-list 'org-export-filter-item-functions
             '+org-export-latex-fancy-item-checkboxes)
```

Cover Pages

```
 (defvar org-latex-cover-page 'auto
  "When t, use a cover page by default. When auto, use a cover page when the
  ↪  document's wordcount exceeds Set with #+option: coverpage:{yes,auto,no}
  ↪  in org buffers.")
(defvar org-latex-cover-page-wordcount-threshold 5000
  "Document word count at which a cover page will be used automatically.
This condition is applied when cover page option is set to auto.")
(defvar org-latex-subtitle-coverpage-format
↪  "\\\\\\bigskip\n\\LARGE\\mdseries\\itshape\\color{black!80} %s\\par"
  "Variant of `org-latex-subtitle-format' to use with the cover page.")
(defvar org-latex-cover-page-maketitle
  "\\usepackage{tikz}
\\usetikzlibrary{shapes.geometric}
\\usetikzlibrary{calc}

\\newsavebox\\orgicon
\\begin{lrbox}{\\orgicon}
  \\begin{tikzpicture}[y=0.80pt, x=0.80pt, inner sep=0pt, outer sep=0pt]
```

```
    \\path[fill=black!6] (16.15,24.00) .. controls (15.58,24.00) and
    ↪  (13.99,20.69) .. (12.77,18.06)arc(215.55:180.20:2.19) .. controls
    ↪  (12.33,19.91) and (11.27,19.09) .. (11.43,18.05) .. controls
    ↪  (11.36,18.09) and (10.17,17.83) .. (10.17,17.82) .. controls
    ↪  (9.94,18.75) and (9.37,19.44) .. (9.02,18.39) .. controls (8.32,16.72)
    ↪  and (8.14,15.40) .. (9.13,13.80) .. controls (8.22,9.74) and
    ↪  (2.18,7.75) .. (2.81,4.47) .. controls (2.99,4.47) and (4.45,0.99) ..
    ↪  (9.15,2.41) .. controls (14.71,3.99) and (17.77,0.30) .. (18.13,0.04)
    ↪  .. controls (18.65,-0.49) and (16.78,4.61) .. (12.83,6.90) .. controls
    ↪  (10.49,8.18) and (11.96,10.38) .. (12.12,11.15) .. controls
    ↪  (12.12,11.15) and (14.00,9.84) .. (15.36,11.85) .. controls
    ↪  (16.58,11.53) and (17.40,12.07) .. (18.46,11.69) .. controls
    ↪  (19.10,11.41) and (21.79,11.58) .. (20.79,13.08) .. controls
    ↪  (20.79,13.08) and (21.71,13.90) .. (21.80,13.99) .. controls
    ↪  (21.97,14.75) and (21.59,14.91) .. (21.47,15.12) .. controls
    ↪  (21.44,15.60) and (21.04,15.79) .. (20.55,15.44) .. controls
    ↪  (19.45,15.64) and (18.36,15.55) .. (17.83,15.59) .. controls
    ↪  (16.65,15.76) and (15.67,16.38) .. (15.67,16.38) .. controls
    ↪  (15.40,17.19) and (14.82,17.01) .. (14.09,17.32) .. controls
    ↪  (14.70,18.69) and (14.76,19.32) .. (15.50,21.32) .. controls
    ↪  (15.76,22.37) and (16.54,24.00) .. (16.15,24.00) -- cycle(7.83,16.74)
    ↪  .. controls (6.83,15.71) and (5.72,15.70) .. (4.05,15.42) .. controls
    ↪  (2.75,15.19) and (0.39,12.97) .. (0.02,10.68) .. controls (-0.02,10.07)
    ↪  and (-0.06,8.50) .. (0.45,7.18) .. controls (0.94,6.05) and (1.27,5.45)
    ↪  .. (2.29,4.85) .. controls (1.41,8.02) and (7.59,10.18) .. (8.55,13.80)
    ↪  -- (8.55,13.80) .. controls (7.73,15.00) and (7.80,15.64) ..
    ↪  (7.83,16.74) -- cycle;
  \\end{tikzpicture}
\\end{lrbox}

\\makeatletter
\\g@addto@macro\\tableofcontents{\\clearpage}
\\renewcommand\\maketitle{
  \\thispagestyle{empty}
  \\hyphenpenalty=10000 % hyphens look bad in titles
  \\renewcommand{\\baselinestretch}{1.1}
  \\NewCommandCopy{\\oldtoday}{\\today}
  \\renewcommand{\\today}{\\LARGE\\number\\year\\\\\\large%
    \\ifcase \\month \\or Jan\\or Feb\\or Mar\\or Apr\\or May \\or Jun\\or
    ↪  Jul\\or Aug\\or Sep\\or Oct\\or Nov\\or Dec\\fi
    ~\\number\\day}
  \\begin{tikzpicture}[remember picture,overlay]
    %% Background Polygons %%
    \\foreach \\i in {2.5,...,22} % bottom left
    {\\node[rounded corners,black!3.5,draw,regular polygon,regular polygon
    ↪  sides=6, minimum size=\\i cm,ultra thick] at ($(current
    ↪  page.west)+(2.5,-4.2)$) {} ;}
    \\foreach \\i in {0.5,...,22} % top left
    {\\node[rounded corners,black!5,draw,regular polygon,regular polygon
    ↪  sides=6, minimum size=\\i cm,ultra thick] at ($(current page.north
    ↪  west)+(2.5,2)$) {} ;}
    \\node[rounded corners,fill=black!4,regular polygon,regular polygon
    ↪  sides=6, minimum size=5.5 cm,ultra thick] at ($(current page.north
    ↪  west)+(2.5,2)$) {};
    \\foreach \\i in {0.5,...,24} % top right
```

```
    {\\node[rounded corners,black!2,draw,regular polygon,regular polygon
    ↪  sides=6, minimum size=\\i cm,ultra thick] at ($(current page.north
    ↪  east)+(0,-8.5)$) {} ;}
    \\node[fill=black!3,rounded corners,regular polygon,regular polygon
    ↪  sides=6, minimum size=2.5 cm,ultra thick] at ($(current page.north
    ↪  east)+(0,-8.5)$) {};
    \\foreach \\i in {21,...,3} % bottom right
    {\\node[black!3,rounded corners,draw,regular polygon,regular polygon
    ↪  sides=6, minimum size=\\i cm,ultra thick] at ($(current page.south
    ↪  east)+(-1.5,0.75)$) {} ;}
    \\node[fill=black!3,rounded corners,regular polygon,regular polygon
    ↪  sides=6, minimum size=2 cm,ultra thick] at ($(current page.south
    ↪  east)+(-1.5,0.75)$) {};
    \\node[align=center, scale=1.4] at ($(current page.south
    ↪  east)+(-1.5,0.75)$) {\\usebox\\orgicon};
    %% Text %%
    \\node[left, align=right, black, text width=0.8\\paperwidth, minimum
    ↪  height=3cm, rounded corners,font=\\Huge\\bfseries] at ($(current
    ↪  page.north east)+(-2,-8.5)$)
    {\\@title};
    \\node[left, align=right, black, text width=0.8\\paperwidth, minimum
    ↪  height=2cm, rounded corners, font=\\Large] at ($(current page.north
    ↪  east)+(-2,-11.8)$)
    {\\scshape \\@author};
    \\renewcommand{\\baselinestretch}{0.75}
    \\node[align=center,rounded corners,fill=black!3,text=black,regular
    ↪  polygon,regular polygon sides=6, minimum size=2.5 cm,inner sep=0,
    ↪  font=\\Large\\bfseries ] at ($(current page.west)+(2.5,-4.2)$)
    {\\@date};
  \\end{tikzpicture}
  \\let\\today\\oldtoday
  \\clearpage}
\\makeatother
\\newpage"
  "LaTeX preamble snippet that sets \\maketitle to produce a cover page.")

(eval '(cl-pushnew '(:latex-cover-page nil "coverpage" org-latex-cover-page)
                   (org-export-backend-options (org-export-get-backend
                   ↪  'latex))))
```

```
(defun org-latex-cover-page-p ()
  "Whether a cover page should be used when exporting this Org file."
  (pcase (or (car
              (delq nil
                    (mapcar
                     (lambda (opt-line)
                       (plist-get (org-export--parse-option-keyword opt-line
                       ↪  'latex) :latex-cover-page))
                     (cdar (org-collect-keywords '("OPTIONS"))))))
             org-latex-cover-page)
    ((or 't 'yes) t)
    ('auto (when (> (count-words (point-min) (point-max))
    ↪  org-latex-cover-page-wordcount-threshold) t))
    (_ nil)))
```

```
(defadvice! org-latex-set-coverpage-subtitle-format-a (contents info)
  "Set the subtitle format when a cover page is being used."
  :before 'org-latex-template
  (when (org-latex-cover-page-p)
    (setf info (plist-put info :latex-subtitle-format
    ↪  org-latex-subtitle-coverpage-format))))

(setq org-latex-custom-id '("\\usepackage{tocloft}"
"\\setlength{\\cftbeforesecskip}{1ex}"
"\\setlength{\\cftbeforesubsecskip}{0.5ex}"
"\\setlength{\\cftbeforesubsubsecskip}{0.5ex}"
"\\newpage"))
```

```
(setq org-latex-custom-id '("\\usepackage{tocloft}"
"\\setlength{\\cftbeforesecskip}{1ex}"
"\\setlength{\\cftbeforesubsecskip}{0.5ex}"
"\\setlength{\\cftbeforesubsubsecskip}{0.5ex}"
"\\newpage"))
```

5.10 Org-Agenda and Dailies

5.10.1 Define My Daily Template:

TODO implement function that will link to last daily node

```
(defun insert-previous-daily-link ()
  "Insert link to the previous daily note, if available."
  (interactive)
  (let ((prev-note (org-roam-dailies-find-previous-note)))
    (when prev-note
      (insert (format "[[%s][Previous Daily Note]]\n" prev-note)))))
```

search through roam/dailies directory → find most recently created node (by date) and insert link to that node at the top of the created daily file

Also, fix this such that it actually works, cause I had to open my config and c-c-c-c this to make it work

```
(setq org-roam-dailies-capture-templates
          (let ((head
                 (concat "#+title: %<%Y-%m-%d (%A)>\n#+startup:
                 ↪  showall\n#+filetags: Dailies\n* Daily Overview\n"
                         "#+begin_src emacs-lisp :results value raw\n"
                         "(as/get-daily-agenda \"%<%Y-%m-%d>\")\n"
                         "#+end_src\n"
                         "* [/] Do Today\n* [/] Maybe Do Today\n* Journal\n")))
            `(("j" "journal" entry
               "* %<%H:%M> %?"
               :if-new (file+head+olp "%<%Y-%m-%d>.org" ,head ("Journal")))
              ("t" "do today" item
               "[ ] %i%?"
               :if-new (file+head+olp "%<%Y-%m-%d>.org" ,head ("TODO Do
               ↪  Today"))
               :immediate-finish nil)
              ("m" "maybe do today" item
               "[ ] %a"
```

```
                :if-new (file+head+olp "%<%Y-%m-%d>.org" ,head ("Maybe Do
                ↪  Today"))
                :immediate-finish t))))
```

Hello ? I would like this to work, but honestly don't know if it will work all to well for me, lets see I guess

```
;; Set up org-agenda-files to include Org Roam dailies directory
(setq org-agenda-files (append org-agenda-files (list "~/org/roam/daily")))
```

5.10.2 Defining Some Custom Commands

```
; preface, I stole this straight from the internet, so I dunno even if this
↪  will work, and only have a loose Idea as to how it should work
(defun as/org-roam-today-mk-agenda-link ()
  (interactive)
  (let* ((marker (or (org-get-at-bol 'org-marker)
                     (org-agenda-error)))
         (buffer (marker-buffer marker))
         (pos (marker-position marker)))
    (with-current-buffer buffer
      (save-excursion
        (goto-char pos)
        (org-roam-dailies-capture-today)))))

(defun as/get-daily-agenda (&optional date)
  "Return the agenda for the day as a string."
  (interactive)
  (let ((file (make-temp-file "daily-agenda" nil ".txt")))
    (org-agenda nil "d" nil)
    (when date (org-agenda-goto-date date))
    (org-agenda-write file nil nil "*Org Agenda(d)*")
    (kill-buffer)
    (with-temp-buffer
      (insert-file-contents file)
      (goto-char (point-min))
      (kill-line 2)
      (while (re-search-forward "^  " nil t)
        (replace-match "- " nil nil))
      (buffer-string))))
```

5.10.3 Tell Org-Agenda About The Custom Commands

```
;; Customize the default Org agenda command to include Org Roam daily files
(setq org-agenda-custom-commands
      '(("d" "Org Roam Daily Files"
         ((agenda "" ((org-agenda-files (list "~/org/roam/daily"))))
          (function as/org-roam-today-mk-agenda-link)
          (function as/get-daily-agenda)))))
```

6 Language Stuff

doom emacs is super nice in having a lot of easy configuration found in the init.el file, but for anything that doesn't come with doom, I likely have to add it to the packages.el file, and handle it here either that, or just specify options for stuff that needs it

6.1 Flycheck

```
(use-package! flycheck
  :ensure t
  :defer t
  :diminish
  :init (global-flycheck-mode))
(ispell-change-dictionary "en_US" t)
```

6.2 LSP Stuff in particular

6.2.1 File Templates:

```
(set-file-template! "\\.pro" :trigger "__" :mode 'prolog-mode)
```

6.2.2 Prolog

I've been using prolog for some classes, and am honestly enjoying using the language, it's growing on me for sure. What's nice is that Prolog is largely a GNU project, and so it's already included in base emacs with a prolog-mode. Just needs some configuring, and an lsp backend to make things real nice

```
  (after! lsp-mode
    (lsp-register-client
     (make-lsp-client
      :new-connection
      (lsp-stdio-connection (list "swipl"
                                  "-g" "use_module(library(lsp_server))."
                                  "-g" "lsp_server:main"
                                  "-t" "halt"
                                  "--" "stdio"))
      :major-modes '(prolog-mode)
      :priority 1
      :multi-root t
      :server-id 'prolog-ls))
    )
(when (not (executable-find "swipl"))
  (warn! "Swipl not found in the system, prolog might not work as expected"))
```

Hooks and such

```
(add-hook 'find-file-hook #'my-prolog-mode-setup)

(defun my-prolog-mode-setup ()
```

```
  "Custom setup for .pro files."
  (when (and (stringp buffer-file-name)
             (string= (file-name-extension buffer-file-name) "pro"))
    (prolog-mode)
    (lsp)))
```

6.3 TODO Lexic

```
(use-package! lexic
  :commands lexic-search lexic-list-dictionary
  :config
  (map! :map lexic-mode-map
        :n "q" #'lexic-return-from-lexic
        :nv "RET" #'lexic-search-word-at-point
        :n "a" #'outline-show-all
        :n "h" (cmd! (outline-hide-sublevels 3))
        :n "o" #'lexic-toggle-entry
        :n "n" #'lexic-next-entry
        :n "N" (cmd! (lexic-next-entry t))
        :n "p" #'lexic-previous-entry
        :n "P" (cmd! (lexic-previous-entry t))
        :n "E" (cmd! (lexic-return-from-lexic) ; expand
                     (switch-to-buffer (lexic-get-buffer)))
        :n "M" (cmd! (lexic-return-from-lexic) ; minimise
                     (lexic-goto-lexic))
        :n "C-p" #'lexic-search-history-backwards
        :n "C-n" #'lexic-search-history-forwards
        :n "/" (cmd! (call-interactively #'lexic-search))))
```

lsfdkjsdflksajdlkjafsd

```
(defadvice! +lookup/dictionary-definition-lexic (identifier &optional arg)
  "Look up the definition of the word at point (or selection) using
  ↪  `lexic-search'."
  :override #'+lookup/dictionary-definition
  (interactive
   (list (or (doom-thing-at-point-or-region 'word)
             (read-string "Look up in dictionary: "))
         current-prefix-arg))
  (lexic-search identifier nil nil t))
```

6.4 Spell-Checking

6.4.1 Abbrev

gangster swagger

```
(setq-default abbrev-mode t)

(defvar abbrev-fn (expand-file-name "misc/abbrev.el" doom-user-dir))
(setq abbrev-file-name abbrev-fn)
```

6.4.2 Jinx

```
(use-package! jinx
       :defer t
       :init
       (add-hook 'doom-init-ui-hook #'global-jinx-mode)
       :config
       ;; Use my custom dictionary
       (setq jinx-languages "en-custom")
       ;; Extra face(s) to ignore
       (push 'org-inline-src-block
       (alist-get 'org-mode jinx-exclude-faces))
       ;; Take over the relevant bindings.
       (after! ispell
       (global-set-key [remap ispell-word] #'jinx-correct))
       (after! evil-commands
       (global-set-key [remap evil-next-flyspell-error] #'jinx-next)
       (global-set-key [remap evil-prev-flyspell-error] #'jinx-previous))
       ;; I prefer for `point' to end up at the start of the word,
       ;; not just after the end.
       (advice-add 'jinx-next :after (lambda (_) (left-word))))
```

6.5 LaTeX

```
(after! cdlatex
  (setq cdlatex-env-alist
        '(("bmatrix" "\\begin{bmatrix}\n?\n\\end{bmatrix}" nil)
          ("equation*" "\\begin{equation*}\n?\n\\end{equation*}" nil)))
  (setq ;; cdlatex-math-symbol-prefix ?\; ;; doesn't work at the moment :(
   cdlatex-math-symbol-alist
   '( ;; adding missing functions to 3rd level symbols
     (?_    ("\\downarrow"  ""           "\\inf"))
     (?2    ("^2"           "\\sqrt{?}"     ""     ))
     (?3    ("^3"           "\\sqrt[3]{?}"  ""     ))
     (?^    ("\\uparrow"    ""           "\\sup"))
     (?k    ("\\kappa"      ""           "\\ker"))
     (?m    ("\\mu"         ""           "\\lim"))
     (?c    (""             "\\circ"     "\\cos"))
     (?d    ("\\delta"      "\\partial"  "\\dim"))
     (?D    ("\\Delta"      "\\nabla"    "\\deg"))
     ;; no idea why \Phi isnt on 'F' in first place, \phi is on 'f'.
     (?F    ("\\Phi"))
     ;; now just convenience
     (?.    ("\\cdot" "\\dots"))
     (?:    ("\\vdots" "\\ddots"))
     (?*    ("\\times" "\\star" "\\ast")))
   cdlatex-math-modify-alist
   '( ;; my own stuff
     (?B    "\\mathbb"        nil          t    nil  nil)
     (?a    "\\abs"           nil          t    nil  nil))))
```

6.6 GraphViz

```
(use-package! graphviz-dot-mode
  :commands graphviz-dot-mode
  :mode '("\\.dot\\'" . graphviz-dot-mode)
  :init
  (after! org
    (setcdr (assoc "dot" org-src-lang-modes)
            'graphviz-dot)))

(use-package! company-graphviz-dot
  :after graphviz-dot-mode)
```

6.7 Snippets

I use yasnippets like a good sheeple

```
(setq yas-triggers-in-field t)
```

auto expanding snippets

```
(use-package! aas
  :commands aas-mode)
```

7 Unsorted Config (temporary)

Unsorted config as I fix my fuckups

```
;;"A variable-pitch face with serifs."
;;:group 'basic-faces)
;;
;;(defcustom variable-pitch-serif-font (font-spec :family "serif")
;;"The font face used for `variable-pitch-serif'."
;;:group 'basic-faces
;;:set (lambda (symbol value)
;;(set-face-attribute 'variable-pitch-serif nil :font value)
;;(set-default-toplevel-value symbol value)))
;;(setq org-pretty-mode t)









;;(after!
;;:and (org flycheck)
;;(defconst flycheck-org-lint-form
;;  (flycheck-prepare-emacs-lisp-form
;;    (require 'org)
;;    (require 'org-lint)
;;    (require 'org-attach)
```

```
;;    (let ((source (car command-line-args-left))
;;          (process-default-directory default-directory))
;;      (with-temp-buffer
;;        (insert-file-contents source 'visit)
;;        (setq buffer-file-name source)
;;        (setq default-directory process-default-directory)
;;        (delay-mode-hooks (org-mode))
;;        (setq delayed-mode-hooks nil)
;;        (dolist (err (org-lint))
;;          (let ((inf (cl-second err)))
;;            (princ (elt inf 0))
;;            (princ ": ")
;;            (princ (elt inf 2))
;;            (terpri)))))))
;;
;;(defconst flycheck-org-lint-variables
;;  '(org-directory
;;    org-id-locations
;;    org-id-locations-file
;;    org-attach-id-dir
;;    org-attach-use-inheritance
;;    org-attach-id-to-path-function-list
;;    org-link-parameters)
;;  "Variables inherited by the org-lint subprocess.")
;;
;;(defun flycheck-org-lint-variables-form ()
;;  (require 'org-attach)  ; Needed to make variables available
;;  `(progn
;;     ,@(seq-map (lambda (opt) `(setq-default ,opt ',(symbol-value opt)))
;;                (seq-filter #'boundp flycheck-org-lint-variables))))
;;
;;(eval ; To preveant eager macro expansion form loading flycheck early.
;; '(flycheck-define-checker org-lint
;;   "Org buffer checker using `org-lint'."
;;   :command ("emacs" (eval flycheck-emacs-args)
;;             "--eval" (eval (concat "(add-to-list 'load-path \""
;;                                    (file-name-directory (locate-library
↪  "org"))
;;                                    "\")"))
;;             "--eval" (eval (flycheck-sexp-to-string
;;                             (flycheck-org-lint-variables-form)))
;;             "--eval" (eval (flycheck-sexp-to-string
;;                             (flycheck-org-lint-customisations-form)))
;;             "--eval" (eval flycheck-org-lint-form)
;;             "--" source)
;;   :error-patterns
;;   ((error line-start line ": " (message) line-end))
;;   :modes org-mode))
;;
;;(add-to-list 'flycheck-checkers 'org-lint)
;;
;;(defun flycheck-org-lint-customisations-form ()
;;  `(progn
;;     (require 'ox)
;;     (cl-pushnew '(:latex-cover-page nil "coverpage" nil)
;;                 (org-export-backend-options (org-export-get-backend
↪  'latex)))
```

```
;;     (cl-pushnew '(:latex-font-set nil "fontset" nil)
;;                 (org-export-backend-options (org-export-get-backend
↪  'latex))))))
```