## Resources

- Hoogle (look up diagrams functions)
- Diagrams homepage
- Diagrams official tutorials
- Haskell's prelude (standard functions and types)
- Diagram list of colours

## Installing Haskell and getting started

Install using GHCup

- Commands for each OS to install Haskell:
  - Linux, macOS, etc
    `curl --proto '=https' --tlsv1.2 -sSf https://get-ghcup.haskell.org | sh`
  - Windows: too long to fit on this PDF because Windows is Windows. It's on the website just above though.
- Then to make a new project, in a new empty directory, run:
  - `cabal init .`
- Leave most things default. This'll make a new cabal project (cabal is Haskell's package manager/build system) so you can then install the diagrams library.
- To install diagrams in your project, open its `.cabal` file (mine is `diagrams-workshop.cabal`)
  - Make the executable part look like this (change the build-depends part):

```
executable diagrams-workshop
    import:           warnings
    main-is:          Main.hs
    build-depends:    base ^>=4.17.2.1, diagrams-lib, diagrams-svg
    hs-source-dirs:   app
    default-language: Haskell2010
```

- `cabal build` builds your project
- `cabal exec diagrams-workshop -- -o out.svg -w 512` runs it (build it first!!), generating a 512x512 SVG image
- `cabal exec diagrams-workshop -- -o out.svg -w 512 -l -s app/Main.hs` runs it, and also watches the file `app/Main.hs` for changes

## Diagrams cheat-sheet

- Shapes
  - `circle <size>`
  - `square <size>`
  - `rect <width> <height>`

  - `regPoly <sides> <size>`
  - `wedge <size> <start direction> <sweep angle>`
  - `fromOffsets <list of vectors>`: a path of lines of the given vectors
    * e.g. `fromOffsets [V 0 1]` is a vertical line of length 1
  - lots more! see here
- Transformations
  - `translate <vector2>` (e.g. `translate (V2 1 1)`)
  - `translateX <amount>`, `translateY <amount>`
  - `reflectX`, `reflectY`, `reflectXY`
  - `scale <amount>`, `scaleX <amount>`, `scaleY <amount>`
  - `rotate <angle>`, `rotateBy <angle (as a double or float)>`
- Types
  - Vectors (i.e. offsets): `V2 <x> <y>` or `r2 (<x>, <y>)`
  - Angles: `<angle> @@ deg`, `<angle> @@ rad`, `<proportion of circle> @@ turn`
- Styles
  - `<diagram> # <style>`: apply style to diagram
  - `lw <line width>` (e.g. `thin`, `thick`, `none`, or a number)
  - `lc`, `fc`: line colour, foreground colour
  - `showOrigin`: show a red dot at the transformation origin
- Composition
  - `l <> r` or `atop l r`: place on top of each other
  - `l ||| r`: side by side
  - `l === r`: top and bottom
  - `hcat <list of diagrams>`, `vcat <list of diagrams>`, `mconcat <list of diagrams>`: combine a list of diagrams horizontally, vertically, or placed atop each other
  - `strutX <distance>`, `strutY <distance>`: a blank "spacer"

## Haskell cheat-sheet

- ```
  functionName :: Arg1 -> Arg2 -> ReturnType
  functionName x1 y1 = ...
  ```
- Main function: `main :: IO ()`
- Function application: `f <arg1> <arg2> <etc>`
- Data types
  - Integers: `1`, `523`, `(-50)`, etc
  - Doubles/floats: `1.3`, `542.35`, `(-0.5)`
    * `fromInteger <n>` turns integer `n` into double/float
    * `toInteger` does the opposite
  - Strings: `"hacknotts"`, `"i like strings"`
  - Lists: `[1, 2, 3]`, `[ V2 1 3, V2 5 6 ]`, etc
  ```

        * Lists as ranges: `[1..5]` is `[1, 2, 3, 4, 5]`
        * Ranges with step: `[1,3..10]` is `[1, 3, 5, 7, 9]`
    - Tuples: `(1, "one")`, `(2, "two")`
- List comprehensions: do something to each element in a list
    - e.g. `[ x * 2 | x <- [3, 6, 10] ]` is `[6, 12, 20]`
- Conditions: `if <condition> then <if true> else <if false>`

## Diagrams starter file

```
{-# LANGUAGE NoMonomorphismRestriction #-}
{-# LANGUAGE FlexibleContexts #-}
{-# LANGUAGE TypeFamilies #-}

module Main where

import Diagrams.Prelude
import Diagrams.Backend.SVG.CmdLine

diagram :: Diagram B
diagram = circle 1

main :: IO ()
main = mainWith diagram
```