

but the whole is something  
beside the parts...

Aristotle, talking about  
emergent properties

Today's Superpower

Do a lot with a little by  
combining functions

But First...

# Cabal

The Haskell build tool

```
> mkdir MyProject
```

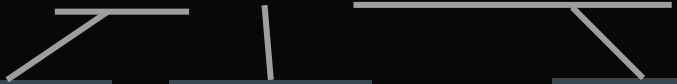
```
> cd MyProject/
```

```
>
```

> **mkdir MyProject**

> **cd MyProject/**

> **cabal init -n --is-executable**



Make this  
directory a  
Cabal  
project

Don't  
prompt for  
project info

We're  
building a  
running  
program

> **mkdir MyProject**

> **cd MyProject/**

> **cabal init -n --is-executable**

Guessing dependencies...

Generating LICENSE...

Warning: unknown license type, you must put a copy in LICENSE yourself.

Generating Setup.hs...

Generating CHANGELOG.md...

Generating Main.hs...

Generating MyProject.cabal...

Warning: no synopsis given. You should edit the .cabal file and add one.

You may want to edit the .cabal file and add a Description field.

>

```
> ls
```

```
CHANGELOG.md Main.hs MyProject.cabal Setup.hs
```

> ls

CHANGELOG.md Main.hs MyProject.cabal Setup.hs

```
module Main where
```

```
main :: IO ()
```

```
main = putStrLn "Hello, Haskell!"
```

```
name:           MyProject
version:        0.1.0.0
license-file:   LICENSE
author:         pragdave
maintainer:     dave@pragdave.me
build-type:     Simple
extra-source-files: CHANGELOG.md
cabal-version:  >=1.10
```

```
executable MyProject
  main-is:           Main.hs
  -- other-modules:
  -- other-extensions:
  build-depends:     base >=4.12 && <4.13
  -- hs-source-dirs:
  default-language:  Haskell2010
```



> ls

CHANGELOG.md Main.hs MyProject.cabal Setup.hs

> cabal v2-run

Resolving dependencies...

Build profile: -w ghc-8.6.5 -O1

In order, the following will be built (use -v for more details):

- MyProject-0.1.0.0 (exe:MyProject) (first run)

Configuring executable 'MyProject' for MyProject-0.1.0.0..

Warning: The 'license-file' field refers to the file 'LICENSE' which does not exist.

Preprocessing executable 'MyProject' for MyProject-0.1.0.0..

Building executable 'MyProject' for MyProject-0.1.0.0..

[1 of 1] Compiling Main ( Main.hs,

/home/dave/Me/EastSide2021/daythree/code/MyProject/dist-newstyle/build/x86\_64-linux/ghc-8.6.5/MyProject-0.1.0.0/x/MyProject/build/MyProject/MyProject-tmp/Main.o )

Linking

/home/dave/Me/EastSide2021/daythree/code/MyProject/dist-newstyle/build/x86\_64-linux/ghc-8.6.5/MyProject-0.1.0.0/x/MyProject/build/MyProject/MyProject ...

Hello, Haskell!

> cabal v2-run

Resolving dependencies...

Build profile: -w ghc-8.6.5 -O1

In order, the following will be built (use -v for more details):

- MyProject-0.1.0.0 (exe:MyProject) (first run)

Configuring executable 'MyProject' for MyProject-0.1.0.0..

Warning: The 'license-file' field refers to the file 'LICENSE' which does not exist.

Preprocessing executable 'MyProject' for MyProject-0.1.0.0..

Building executable 'MyProject' for MyProject-0.1.0.0..

[1 of 1] Compiling Main ( Main.hs,

/home/dave/Me/EastSide2021/daythree/code/MyProject/dist-newstyle/build/x86\_64-linux/ghc-8.6.5/MyProject-0.1.0.0/x/MyProject/build/MyProject/MyProject-tmp/Main.o )

Linking

/home/dave/Me/EastSide2021/daythree/code/MyProject/dist-newstyle/build/x86\_64-linux/ghc-8.6.5/MyProject-0.1.0.0/x/MyProject/build/MyProject/MyProject ...

Hello, Haskell!

> cabal v2-run

Up to date..

Hello, Haskell!

# Using External Libraries

Two-step process:

- Getting them onto your machine
- Adding them to your project

# Using External Libraries

Two-step process:

- Getting them onto your machine
- Adding them to your project

```
> cabal v2-install gloss
```

```
Resolving dependencies...
```

```
Up to date
```

# Using External Libraries

Two-step process:

- Getting them onto your machine
- Adding them to your project

```
name:           MyProject
version:        0.1.0.0
license-file:   LICENSE
author:         pragdave
maintainer:     dave@pragdave.me
build-type:     Simple
cabal-version:  >=1.10

executable MyProject
  main-is:       Main.hs
  build-depends: base >=4.12 && <4.13
```

# Using External Libraries

Two-step process:

- Getting them onto your machine
- Adding them to your project

```
name:           MyProject
version:        0.1.0.0
license-file:   LICENSE
author:         pragdave
maintainer:     dave@pragdave.me
build-type:     Simple
cabal-version:  >=1.10
```

```
build-depends:  base >=4.12 && <4.13, gloss <4.13, gloss
```

# Gloss

Simple Haskell Graphics Library

Main.hs

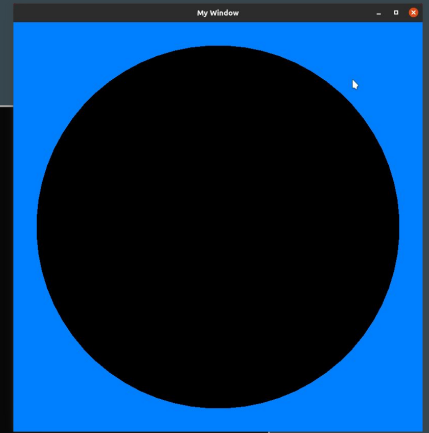
```
module Main where
```

```
import Graphics.Gloss
```

```
main = display windowed azure (circleSolid 350)
```

```
  where
```

```
    windowed = (InWindow "My Window" (800, 800) (10, 10))
```





Main.hs

```
module Main where
```

```
import Graphics.Gloss
```

```
main = display windowed azure (circleSolid 350)  
  where  
    windowed = (InWindow "My Window" (800, 800) (10, 10))
```

Main.hs

```
module Main where  
import Graphics.Gloss
```

Hey, Gloss: display  
the following...

And here's what  
to draw

```
main = display windowed azure (circleSolid 350)
```

```
  where  
    windowed = (InWindow "My Window" (800, 800) (10, 10))
```

Here's where  
you display it

Use this background

Main.hs

```
module Main where
```

```
import Graphics.Gloss
```

```
main = display windowed azure (circleSolid 350)
```

```
  where
```

```
    windowed = (InWindow "My Window" (800, 800) (10, 10))
```

In a window (not  
full screen)



Window title



size



coord of  
top-left



Main.hs

```
module Main where

import Graphics.Gloss

main =
  display windowed azure drawing
  where
    windowed = InWindow . . .
    drawing  = circleSolid 350
```

## Main.hs

```
module Main where  
import Graphics.Gloss
```

```
face =  
    circleSolid 350
```

```
main =  
    display windowed azure drawing  
    where  
        windowed = InWindow "My Window" (800, 800) (10, 10)  
        drawing = face
```

## Main.hs

```
module Main where
import Graphics.Gloss
```

```
face =
  circleSolid 350
```

```
hub =
  color orange $ circleSolid 40
```

```
main =
  display windowed azure drawing
  where
    windowed = InWindow "My Window" (800, 800) (10, 10)
    drawing = pictures [
      face,
      hub
    ]
```

# What's Going On?

- The 3rd parameter to `display` is the picture to display. It is a value of type `Picture`.
- `circleSolid` is a function that takes a number (the radius) and returns a `Picture`

`circleSolid :: Float -> Picture`

- `color` is a function that takes a `Color` and a `Picture`, and returns a new `Picture` drawn in that color

`color :: Color -> Picture -> Picture`

# What's Going On?

```
circleSolid :: Float -> Picture
```

```
color :: Color -> Picture -> Picture
```

```
color orange $ circleSolid 40
```

**Picture**

**Picture**

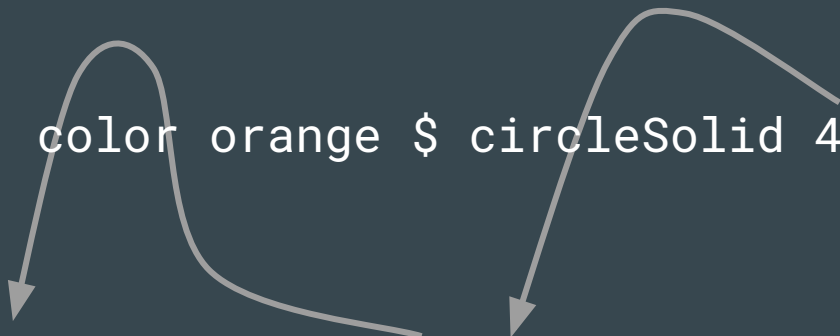


*Seems backwards!*

color orange \$ circleSolid 40

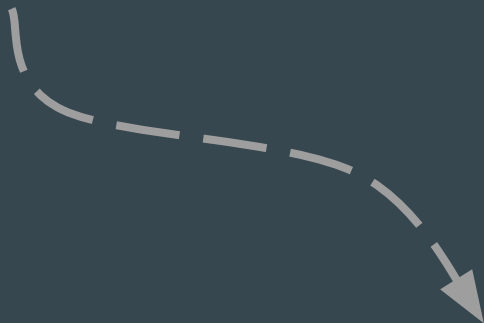
**Picture**

**Picture**



# I want to write

```
circleSolid 40 |> color orange
```



```
color orange (circleSolid 40)
```

Not part of Haskell

So let's add it

`|>` is an operator that takes what's on the left and adds it as a parameter to what's on the right

Sounds pretty complex...

$$(|>) \ x \ f = f \ x$$

## Main.hs

```
module Main where
import Graphics.Gloss

(|>) x f = f x      -- forward pipeline

face =
  circleSolid 350

hub =
  circleSolid 40
  |> color orange

main =
  display windowed azure drawing
  where
    windowed = InWindow "My Window" (800, 800) (10, 10)
    drawing = pictures [
      face,
      hub
    ]
```

## Main.hs

```
face =  
  circleSolid 350  
  
hand =  
  rectangleSolid 5 340  
  |> color yellow  
  |> translate (-2.5) 170  
  
hub =  
  circleSolid 40  
  |> color orange  
  
main =  
  display windowed azure drawing  
  where  
    windowed = InWindow "My Window" (800, 800) (10, 10)  
    drawing = pictures [  
      face,  
      hand,  
      hub  
    ]
```

# Boring!

Let's animate it

*(Remember I said early on: ease of change is the driver of good design...)*

## Main.hs

```
face =  
  circleSolid 350  
  
3 hand seconds =  
  rectangleSolid 5 340  
  |> color yellow  
  |> translate (-2.5) 170  
  |> rotate (seconds * 6)  
  
4  
  
hub =  
  circleSolid 40  
  |> color orange  
  
1 main =  
  animate windowed azure renderer  
  where  
    windowed = InWindow "My Window" (800, 800) (10, 10)  
    renderer seconds =  
      pictures [  
        face,  
        2 hand seconds,  
        hub  
      ]
```

1. Use `animate` instead of `display`.  
(this calls the 3rd parameter at intervals, passing in the elapsed number of seconds)
2. Pass the number of seconds to the hand drawing code.
3. Accept the parameter
4. Use it to rotate the hand.

## Main.hs

```
face =  
  circleSolid 350  
  
hand seconds =  
  rectangleSolid 5 340  
  |> color yellow  
  |> translate (-2.5) 170  
  |> rotate (seconds * 6)  
  
hub =  
  circleSolid 40  
  |> color orange  
  
main =  
  animate windowed azure renderer  
  where  
    windowed = InWindow "My Window" (800, 800) (10, 10)  
    renderer seconds =  
      pictures [  
        face,  
        hand seconds,  
        hub  
      ]
```

Data **flows** through **transformations**

Forward pipelines ( **|>** ) make  
flow **explicit**

**Type checking** ensures **interfaces**  
between transformations are correct

**Homework...**



# Due Tuesday, April 27

Fork and clone the repo <https://github.com/Eastside-FP/daythree> You'll be working on code the the `assignment/` directory, and submitting using a pull request.

Fork and clone the repo <https://github.com/Eastside-FP/daythree>. You'll be working on code the the `assignment/` directory, and submitting using a pull request.

In that directory, you'll find a version of the code we developed in class. Before you change anything, use `cabal v2-run` to make sure it runs for you.

The four parts of the assignment are on the next slide. Before you tackle them, take a moment to think about the ideas of pipelines and transformations. Each of the parts can be solved in many ways, and different parts have different styles of solution. But see if you can identify and express some of those solutions as transformations.

1. The hand currently takes 60s to complete a revolution, because it treats the animation time (**seconds**) as the time to display. Change the code so that our internal time is 30 times faster than animation time. Think about how this change can be expressed in a functional way (as a transformation).
2. Add another hand that rotates 60 times slower than the first. Try to reuse existing code as much as possible, rather than just copying the **hand** function.
3. Add a time display that shows the value of our internal time as **mm:ss**. The seconds field should display 0 to 59 as the second hand sweeps, and the minute field will increase by one for each of the second hand's revolutions. (You'll need to search <https://hackage.haskell.org/> for useful functions)
4. Split the time display, and show the seconds by the end of the second hand, and the minutes by the minutes hand. You choose whether the numbers are shown horizontally or parallel to the hand.