Control.Lens cheat sheet

Getting with Getters

Any function $(s \rightarrow a)$ can be flipped into continuation passing style, $(a \rightarrow r) \rightarrow s \rightarrow r$ and decorated with **Const** to obtain:

```
type Getting r s a =
  (a -> Const r a) -> s -> Const r s
```

A **Getter** describes how to retrieve a single value in a way that can be composed with other **LensLike** constructions.

When you see this in a type signature it indicates that you can pass the function a **Lens**, **Getter**, **Traversal**, **Fold**, **Prism**, **Iso**, or one of the indexed variants, and it will just "do the right thing".

Safe head

Perform a safe head of a **Fold** or **Traversal** or retrieve **Just** the result from a **Getter** or **Lens**.

$(\hat{}?) \equiv \mathit{flip\ preview}$

```
(^?) :: s -> Getting (First a) s a -> Maybe a

>>> Right 4 ^?_Left
Nothing
>>> "world" ^? ix 3
Just 'l'
```

Viewing lenses

View the value pointed to by a **Getter** or **Lens** or the result of folding over all the results of a **Fold** or **Traversal** that points at a monoidal values.

This is the same operation as **view** with the arguments flipped.

```
(^.) :: s -> Getting a s a -> a

>>> (0, -5)^._2.to abs
5
>>> ["a", "b", "c"] ^. traversed
"abc"
```

Using MonadState

Use the target of a **Lens**, **Iso**, or **Getter** in the current state, or use a summary of a **Fold** or **Traversal** that points to a monoidal value.

```
use :: MonadState s m => Getting a s a -> m a

>>> evalState (use _1) (1,2)
1
>>> evalState (uses _1 length) ("hello","")
5
```

Folding Foldables

```
type Fold s a =
  forall m. Monoid m => Getting m s a
```

A **Fold s a** is a generalization of something **Foldable**. It allows you to extract multiple results from a container. Every **Getter** is a valid **Fold** that simply doesn't use the Monoid it is passed.

If there exists a **foo** method that expects a **Foldable (f a)**, then there should be a **fooOf** method that takes a **Fold s a** and a value of type **s**.

Extracting lists from Folds

Extract a list of the targets of a **Fold**, an infix version of **toListOf**.

$\mathit{toList}\ \mathit{xs} \equiv \mathit{xs}\hat{}..\mathit{folded}$

```
(^..) :: s -> Getting (Endo [a]) s a -> [a]

>>> [[1,2],[3]]^..traverse.traverse
[1,2,3]
>>> (1,2)^..both
[1,2]
```

Checking for matches

Check to see if this Fold or Traversal matches 1 or more entries. For the opposite, use **hasn't**.

```
has :: Getting Any s a -> s -> Bool

>>> has (element 0) []
False
>>> has _Right (Left 12)
False
>>> hasn't _Right (Left 12)
True
```

Indexed Getters

For most operations, there is an indexed variant which will work as expected if the underlying target supports a notion of **Indexing**.

```
>>> ["ab", "c"]^@..itraversed<.>itraversed
[((0,0),'a'),((0,1),'b'),((1,0),'c')]
>>> "hello" ^@..itraversed.indices even
[(0,'h'),(2,'l'),(4,'o')]

>>> ifind (\i k -> i > k) [1,2,2,2]
Just (3,2)
```