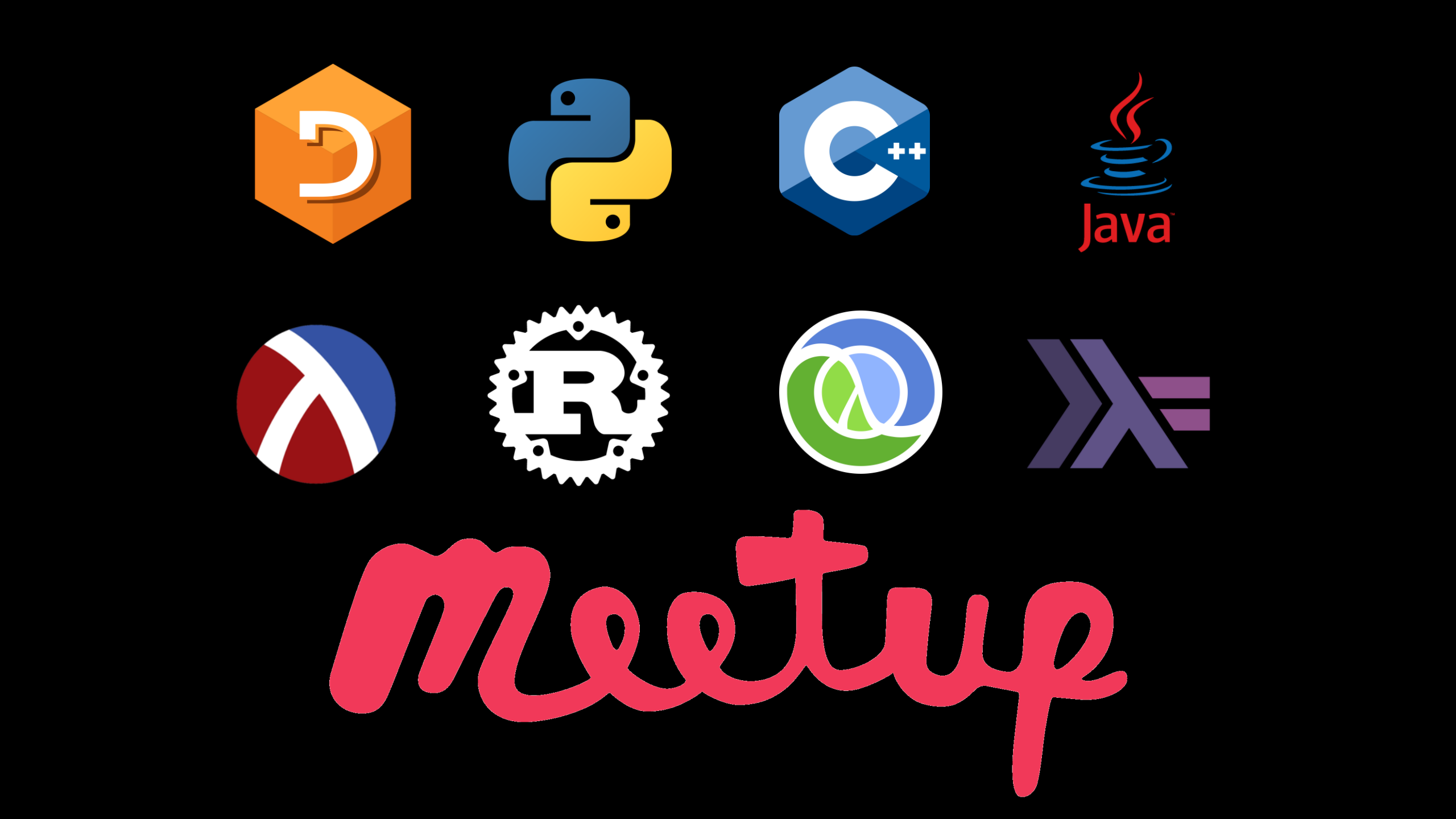
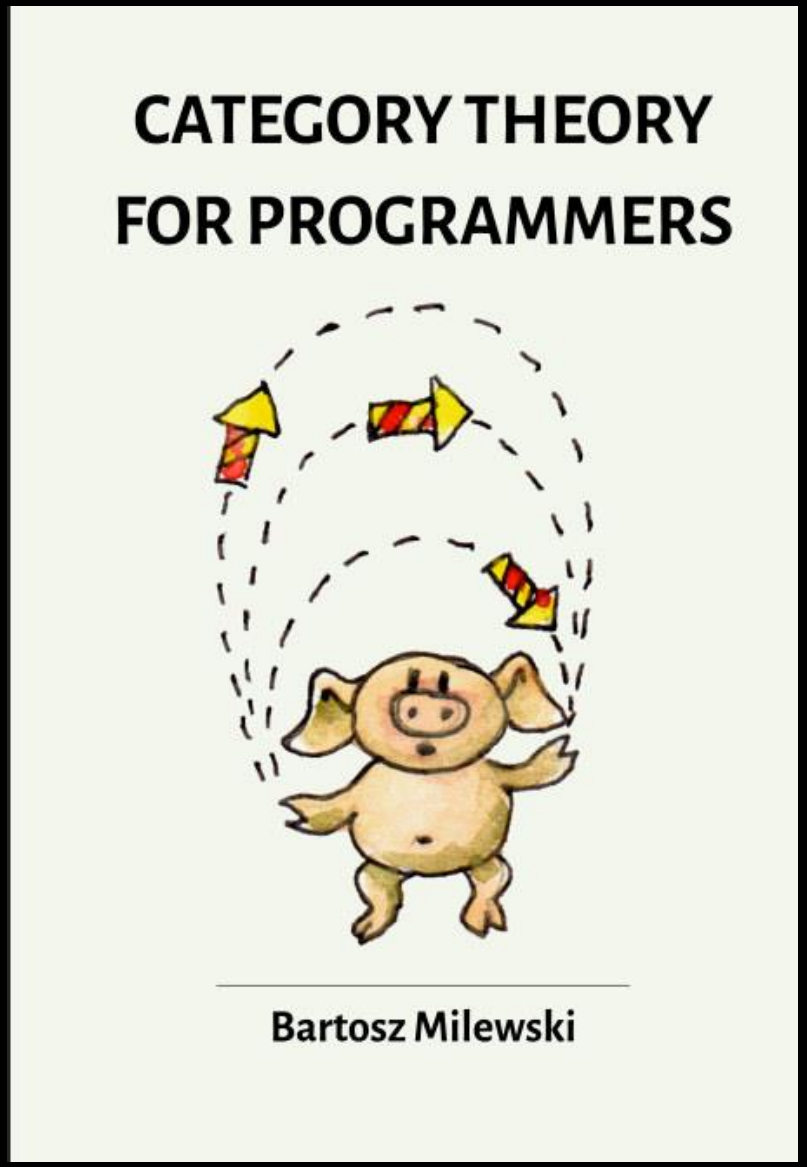

Java
meetup

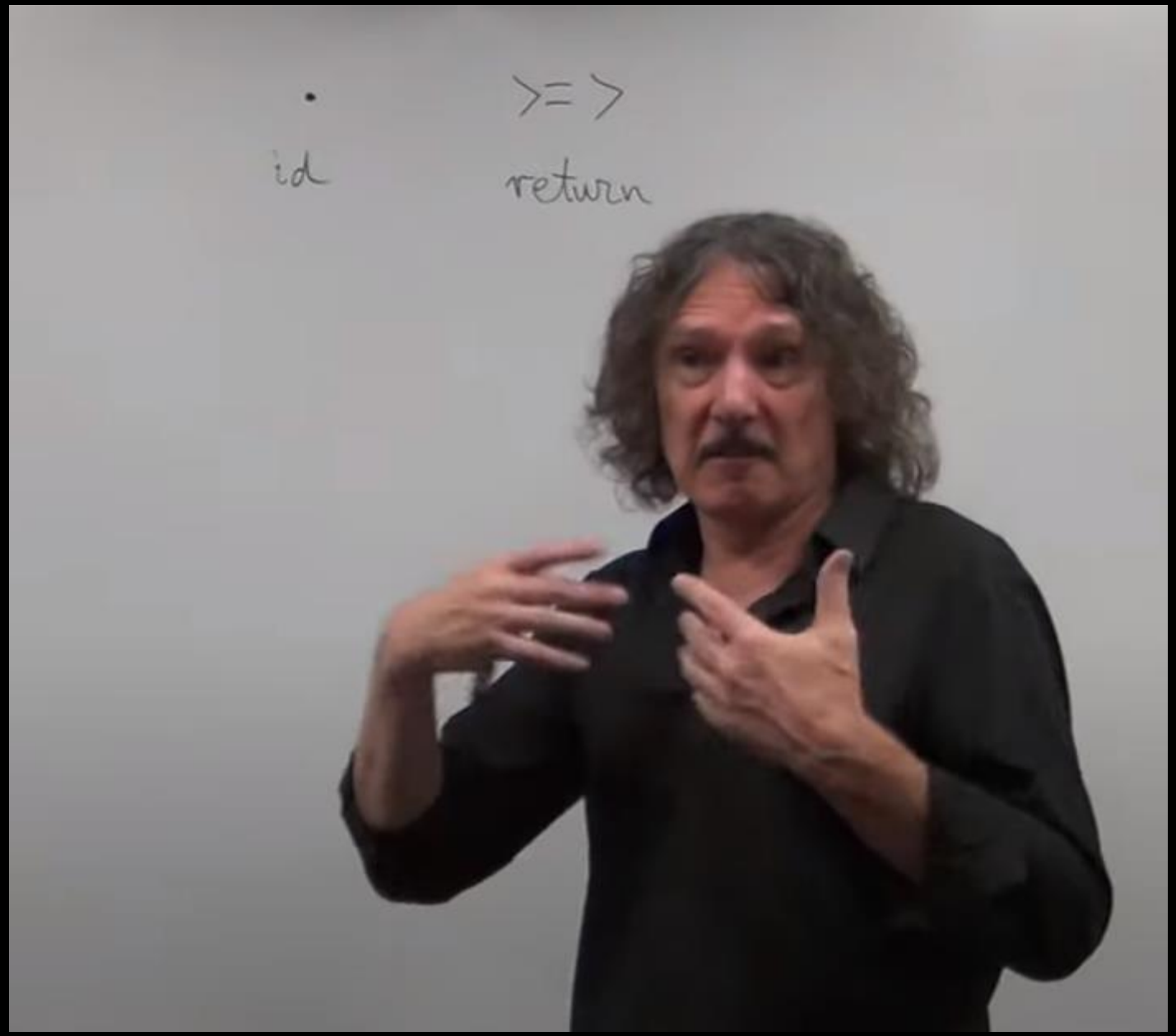
# Friendly Environment Policy


# Berlin Code of Conduct


# Category Theory for Programmers

## Chapter 20-22: Monads


.
>=>
id
return

```haskell
class Monad m where
    (>=>) :: (a -> m b) -> (b -> m c) -> (a -> m c)
    return :: a -> m a
```

```haskell
class Monad m where
    (>=>) :: (a -> m b) -> (b -> m c) -> (a -> m c)
    return :: a -> m a
```

For every monad, instead of defining the fish operator, we may instead define bind. In fact the standard Haskell definition of a monad uses bind:

```haskell
class Monad m where
    (>>=) :: m a -> (a -> m b) -> m b
    return :: a -> m a
```

```haskell
class Monad m where
    (>=>) :: (a -> m b) -> (b -> m c) -> (a -> m c)
    return :: a -> m a
```

For every monad, instead of defining the fish operator, we may instead define bind. In fact the standard Haskell definition of a monad uses bind:

```haskell
class Monad m where
    (>>=) :: m a -> (a -> m b) -> m b
    return :: a -> m a
```

That leads us to the third option for defining a monad:

```haskell
class Functor m => Monad m where
    join :: m (m a) -> m a
    return :: a -> m a
```

```
instance Monad [] where
    join = concat
    return x = [x]
```

In the list monad — Haskell's implementation of nondeterministic computations — `join` is implemented as `concat`. Remember that `join` is supposed to flatten a container of containers — `concat` concatenates a list of lists into a single list. `return` creates a singleton list:

```
instance Monad [] where
    join = concat
    return x = [x]
```

```
instance Monad Maybe where
    Nothing >>= k = Nothing
    Just a  >>= k = k a
    return a = Just a
```

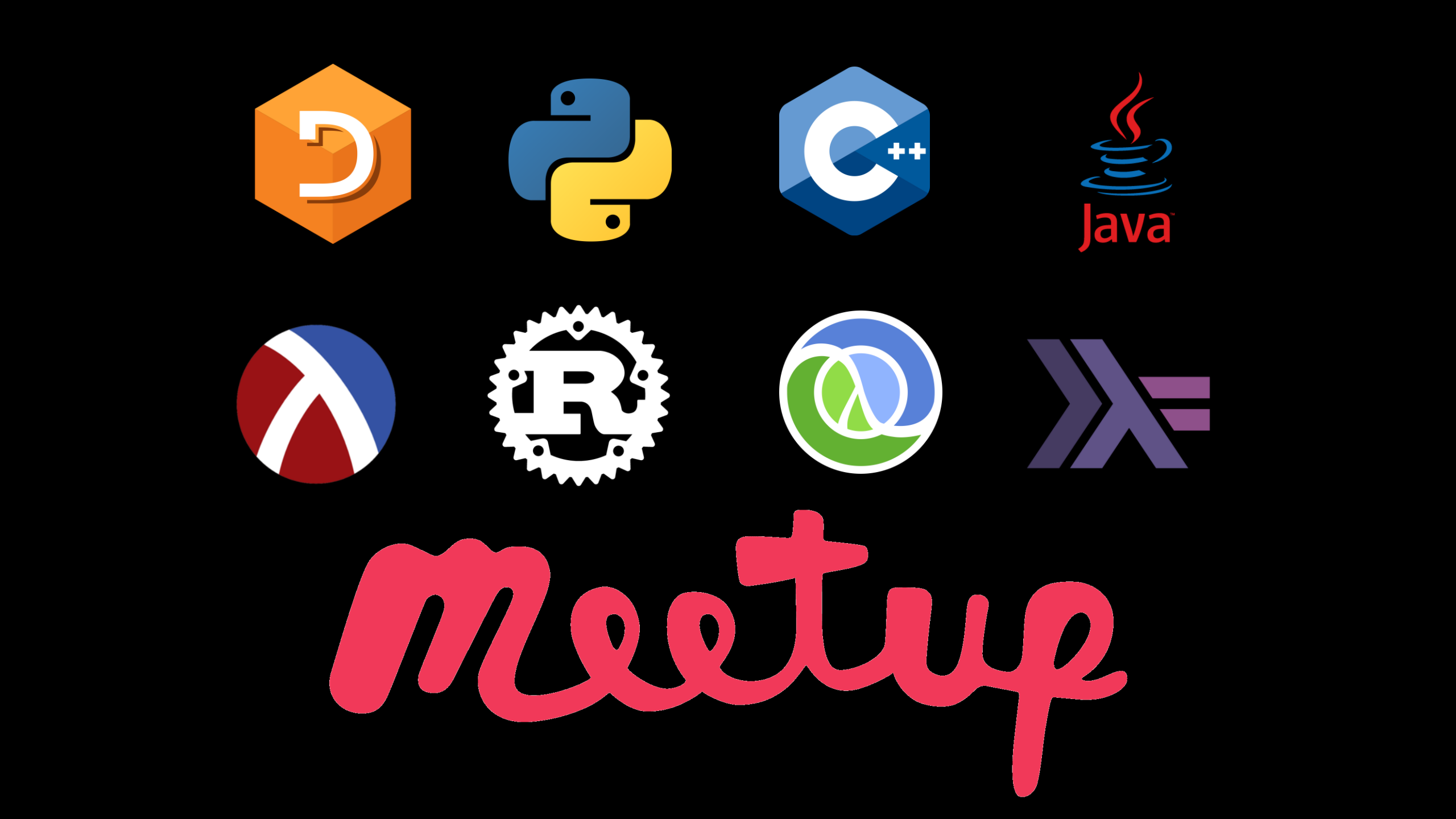
```haskell
import Control.Monad
import Data.Maybe

-- not generic to avoid error
safeHead :: [Int] -> Maybe Int
safeHead []    = Nothing
safeHead (x:_) = Just x

safeDouble :: Int -> Maybe Int
safeDouble x = Just (2 * x)

double :: Int -> Int
double = (2*)

main :: IO ()
main = do
    print $ safeHead [42,1729,343]
    print $ safeHead []
    print $ (safeHead >=> safeDouble) [42,1729,343]
    print $ fmap double $ safeHead [42,1729,343]
    print $ (fmap double . safeHead) [42,1729,343]
    print $ double <$> (safeHead [42,1729,343])
    print $ (<$>) double (safeHead [42,1729,343])
    print $ (safeHead >=> (return . double)) [42,1729,343]
    print $ (Just [42,1729,343]) >>= safeHead >>= safeDouble
    print $ (safeHead [42,1729,343]) >>= safeDouble
```


Java
meetup