Why teaching functional programming to undergraduates at CUNY is important

Evan Misshula

2018-03-28

Some lessons we've skipped

Defining types

- *data* will define a new algebraic type
- *type* creates a type synonym
- *newtype* creates new types from old types

Applicative Functor in two ways

function left, each argument right

```
:m Control.Applicative
[(+1),(*100),(*5)] <*> [1..3]
1==1


[2,3,4,100,200,300,5,10,15]
```

function left, every argument right

```
:set +m
:{
instance Applicative ZipList wh
pure x = ZipList (repeat x)
ZipList fs <*> ZipList xs = Zip
:}
  getZipList $ ZipList [(+1),(*
  -- getZipList $
  -- ZipList [(+1),(*100),(*5)]
  --  <*> ZipList [1,2,3]
1==1


Prelude Control.Applicative| Pr
```

The newtype keyword

'newtype' takes one type and wrap it

- to present it as another type

ZipList [a]}~

- data can have multiple value contstructors

type vs. newtype vs. data examples

'data' to make new types

- Here are additive and multiplicative types with multiple constructors

```
data Profession = Fighter | Archer | Wizard
data Species = Human | Elf | Orc | Goblin
data PlayerCharacter = PlayerCharacter Species Profession
```

Using newtype to drive typeclass properties

newtype

```
newtype CharList = CharList {getCharList :: [Char]} deriving(Eq
CharList "this will be shown!"
CharList "benny" == CharList "benny"
CharList "benny" == CharList "oisters"
1==1


CharList {getCharList = "this will be shown!"}
True
False
```