

PROGRAMMING IN HASKELL

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Monads

Introduction

In the course we have made two uses of the do-notation so far:

Assignment Project Exam Help

- Parsers
- IO

<https://eduassistpro.github.io/>

Parsers and IO are two instances of a monad called a monad.

Add WeChat `edu_assist_pro`

strict structure

Parsing using the do-notation

A sequence of parsers can be combined as a single composite parser using the keyword do.

Assignment Project Exam Help

For example:

<https://eduassistpro.github.io/>

```
p :: Parser (Char,Char)WeChat edu_assist_pro
p = do x ← item
      item
      y ← item
      return (x,y)
```

IO using the do-notation

A sequence of actions can be combined as a single composite action using the keyword do.

Assignment Project Exam Help

For example:

<https://eduassistpro.github.io/>

a :: IO (Char,Char) **Add WeChat edu_assist_pro**

a = do x ← getChar

getChar

y ← getChar

return (x,y)

Monads

- Monads are a structure composed of two basic operations (bind and return), which capture a common pattern that occurs in many types.
Assignment Project Exam Help
- In Haskell Mo <https://eduassistpro.github.io/> using type
classes: **Add WeChat edu_assist_pro**

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

Do-notation

- The do-notation is just simple syntactic sugar on top of the monad operations:

Assignment Project Exam Help

```
do  pattern      https://eduassistpro.github.io/  
    morelinesAdd WeChat edu_assist_pro
```

Is converted to code using bind:

```
exp >>= (\pattern -> do morelines)
```

Do-notation

- The do-notation is just simple syntactic sugar on top of the monad operations:

Assignment Project Exam Help

do exp
more lines [Add WeChat edu_assist_pro](https://eduassistpro.github.io/)

Is converted to code using bind:

```
exp >>= (\ -> do morelines)
```

Do-notation

- The do-notation is just simple syntactic sugar on top of the monad operations:

Assignment Project Exam Help

```
do    return e      https://eduassistpro.github.io/  
          Add WeChat edu_assist_pro
```

Is simplified to:

```
return exp
```

Parsing using the do-notation

Lets rewrite a program using the do-notation!

```
do x ← item  
    item      Assignment Project Exam Help  
              https://eduassistpro.github.io/  
    y ← item Add WeChat edu_assist_pro  
    return (x,y)
```

Parsing using the do-notation

Lets rewrite a program using the do-notation!

```
item >>= (\x->  
    item >>= (  
        item >>=  
            return AddWeChat edu_assist_pro  
    )  
)  
Assignment Project Exam Help  
https://eduassistpro.github.io/  
AddWeChat edu_assist_pro
```

What are the steps?

Parsing using the do-notation

Lets rewrite a program using the do-notation!

```
item >>= \x
  do item      https://eduassistpro.github.io/
    y ← itemAdd WeChat edu_assist_pro
  return (x,y)
```

Parsing using the do-notation

Lets rewrite a program using the do-notation!

```
item >>= \x Assignment Project Exam Help  
item >>= \https://eduassistpro.github.io/  
do y ← item Add WeChat edu_assist_pro  
return (x,y)
```

Parsing using the do-notation

Lets rewrite a program using the do-notation!

```
item >>= \x Assignment Project Exam Help  
item >>= \https://eduassistpro.github.io/  
item >>= Add-WeChat edu_assist_pro  
    do return (x,y)
```

Parsing using the do-notation

Lets rewrite a program using the do-notation!

```
item >>= \x Assignment Project Exam Help  
item >>= \https://eduassistpro.github.io/  
item >>= Add-WeChat edu_assist_pro  
return (x,y)
```

Creating Monads

- Monads are created using an instance of the Monad type class.

Assignment Project Exam Help

- The Parser m Monad <https://eduassistpro.github.io/>
- The IO Monad is built-in Add WeChat edu_assist_pro

Parser Monad

Here is the Parser Monad:

data Parser a = P (String -> [(a, String)])

Assignment Project Exam Help

instance Monad <https://eduassistpro.github.io/>

— return :: a -> Parser a

return v = P (\inp -> [(v, inp)])

— (>>=) :: Parser a -> (a -> Parser b) -> Parser b

p >>= f = P (\inp -> case parse p inp of

[] -> []

[(v, out)] -> parse (f v) out)

Parser Monad

- Simply providing this instance allows us to use the do-notation!

Assignment Project Exam Help

<https://eduassistpro.github.io/>
newtype Parser a = P (String -> a)]])
Add WeChat edu_assist_pro

instance Monad Parser where

return v = P (\inp -> [(v,inp)])
p >>= f = P (\inp -> case parse p inp of
[] -> []
[(v,out)] -> parse (f v) out)

IO

IO is special in Haskell, since it is a type built-in the language. The compiler provides an instance of the Monad class with suitable return and \gg functions:

return :: a -> IO <https://eduassistpro.github.io/>

($\gg=$) :: IO a -> (a -> IO b) -> IO b

Add WeChat edu_assist_pro

Creating a Simple Monad

Once we understand monads it is easy to create our own monads. An example is a variation of the Maybe type

Assignment Project Exam Help

data Option a =

<https://eduassistpro.github.io/>

instance Monad Option where
??? [Add WeChat edu_assist_pro](#)

Lets create the Option Monad!

Creating a Simple Monad

With option we can create a safer version of arithmetic expressions:

Assignment Project Exam Help
sdiv :: Option Int n Int
<https://eduassistpro.github.io/>
sadd :: Option Int -> Option In n Int
Add WeChat edu_assist_pro

That track whether division by 0 errors occur.

Monad Laws

It is not enough to implement bind and return. A proper monad is also required to satisfy some laws:

Assignment Project Exam Help

return a >>= k

m >>= return =<https://eduassistpro.github.io/>

m >>= (\x -> k x >>= h) =
Add WeChat edu_assist_pro >>= h

Assignment Project Exam Help

Mor<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Monads, Functional Programming and Interpreters

- Monads were introduced to Functional Programming by Philip Wadler
- See the paper by <https://eduassistpro.github.io/> which motivates monads through interpreters (much like the interpreters in the class)

The essence of Functional Programming, Philip Wadler, 1992