```
ouzo2:code$ ghci
GHCi, version 8.2.2: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /Users/hallgren/.ghci
Prelude> :l DataTypes.hs
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
d )
Ok, one module loaded.
*Main> :r
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
d )
Ok, one module loaded.
*Main> Diamonds

<interactive>:3:1: error:
    • No instance for (Show Suit) arising from a use of ‘print
’
    • In a stmt of an interactive GHCi command: print it
*Main> :r
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
d )
Ok, one module loaded.
*Main> Spades
Spades
*Main> :r
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
d )
Ok, one module loaded.
*Main> colour Hearts
Red
*Main> :r
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
d )
Ok, one module loaded.
*Main> colour Clubs
Black
*Main> co
colour     compare    concat     concatMap  const      cos
    cosh
*Main> colour Diamonds
Red
*Main> :r
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
```

```
d )
Ok, one module loaded.
*Main> colour Diamonds
*** Exception: DataTypes.hs:(21,1)-(22,20): Non-exhaustive pat
terns in function colour

*Main> :r
Ok, one module loaded.
*Main> :set -Wincomplete-patterns
*Main> :l DataTypes.hs
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
d )

DataTypes.hs:21:1: warning: [-Wincomplete-patterns]
    Pattern match(es) are non-exhaustive
    In an equation for ‘colour’:
        Patterns not matched:
            Hearts
            Diamonds
   |
21 | colour Spades = Black
   | ^^^^^^^^^^^^^^^^^^^^...
Ok, one module loaded.
*Main>
Leaving GHCi.
ouzo2:code$ ghci
GHCi, version 8.2.2: http://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /Users/hallgren/.ghci
Prelude> :l DataTypes.hs
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
d )

DataTypes.hs:23:1: warning: [-Wincomplete-patterns]
    Pattern match(es) are non-exhaustive
    In an equation for ‘colour’:
        Patterns not matched:
            Hearts
            Diamonds
   |
23 | colour Spades = Black
   | ^^^^^^^^^^^^^^^^^^^^...
Ok, one module loaded.
```

```
*Main> :r
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
d )

DataTypes.hs:23:1: warning: [-Wincomplete-patterns]
    Pattern match(es) are non-exhaustive
    In an equation for ‘colour’:
        Patterns not matched:
            Hearts
            Diamonds
   |
23 | colour Spades = Black
   | ^^^^^^^^^^^^^^^^^^^^^...
Ok, one module loaded.
*Main> coluor Black
Red
*Main> :r
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
d )
Ok, one module loaded.
*Main> Jack
Jack
*Main> Queen
Queen
*Main> Num
Num      Numeric
*Main> Numeric 5
Numeric 5
*Main> :t Jack
Jack :: Rank
*Main> :t Numeric
Numeric :: Int -> Rank
*Main> :t Numeric 5
Numeric 5 :: Rank
*Main> Num
Num      Numeric
*Main> Numeric 9
Numeric 9
*Main> Numeric 25
Numeric 25
*Main> :r
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
```

```
d )

DataTypes.hs:42:54: error:
    Multiple declarations of ‘Jack’
    Declared at: DataTypes.hs:30:27
                 DataTypes.hs:42:54
   |
42 | data Rank' = N2 | N3 | N4 | N5 | N6 | N7 | N8 | N9 | Jack
 | Queen | King | Ace
   |                                                      ^^^^

DataTypes.hs:42:61: error:
    Multiple declarations of ‘Queen’
    Declared at: DataTypes.hs:30:34
                 DataTypes.hs:42:61
   |
42 | data Rank' = N2 | N3 | N4 | N5 | N6 | N7 | N8 | N9 | Jack
 | Queen | King | Ace
   |
   ^^^^^

DataTypes.hs:42:69: error:
    Multiple declarations of ‘King’
    Declared at: DataTypes.hs:30:42
                 DataTypes.hs:42:69
   |
42 | data Rank' = N2 | N3 | N4 | N5 | N6 | N7 | N8 | N9 | Jack
 | Queen | King | Ace
   |
         ^^^^

DataTypes.hs:42:76: error:
    Multiple declarations of ‘Ace’
    Declared at: DataTypes.hs:30:49
                 DataTypes.hs:42:76
   |
42 | data Rank' = N2 | N3 | N4 | N5 | N6 | N7 | N8 | N9 | Jack
 | Queen | King | Ace
   |
                ^^^
Failed, no modules loaded.
Prelude> :r
```

```
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
d )
Ok, one module loaded.
*Main> all
all        all_ranks
*Main> all_ranks
[Numeric 2,Numeric 3,Numeric 4,Numeric 5,Numeric 6,Numeric 7,N
umeric 8,Numeric 9,Numeric 10,Jack,Queen,King,Ace]
*Main> :r
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
d )

DataTypes.hs:37:1: error:
    Parse error: module header, import declaration
    or top-level declaration expected.
   |
37 | rankBeats
   | ^^^^^^^^^
Failed, no modules loaded.
Prelude> :r
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
d )
Ok, one module loaded.
*Main> Jack == Queen
False
*Main> Jack == Jack
True
*Main> Jack < Jack
False
*Main> Jack < Ace
True
*Main> Numeric 5 < Numeric 6
True
*Main> Numeric 5 < Numeric 4
False
*Main> "abc"<"bcd"
True
*Main> "06"<"5"
True
*Main> (Ace,Hearts )
(Ace,Hearts)
*Main> :r
```

```
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
d )
Ok, one module loaded.
*Main> Card Ace Hearts
Card Ace Hearts
*Main> :r
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
d )
Ok, one module loaded.
*Main> :r
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
d )
Ok, one module loaded.
*Main> rank example_card_1
Ace
*Main> :r
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
d )
Ok, one module loaded.
*Main> :t rank
rank :: Card -> Rank
*Main> :r
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
d )
Ok, one module loaded.
*Main> example_card_1
Card {rank = Ace, suit = Hearts}
*Main> example_card_2
Card {rank = King, suit = Spades}
*Main> example_card_2 {suit=Diamonds }
Card {rank = King, suit = Diamonds}
*Main> example_card_

<interactive>:38:1: error:
    • Variable not in scope: example_card_
    • Perhaps you meant one of these:
        'example_card_1' (line 65), 'example_card_2' (line 66)
*Main> example_card_2
Card {rank = King, suit = Spades}
*Main> :r
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
d )
```

```
DataTypes.hs:71:39: error:
    • No instance for (Eq Suit) arising from a use of '=='
    • In the first argument of '(&&)', namely 's1 == s2'
      In the expression: s1 == s2 && rankBeats r1 r2
      In an equation for 'cardBeats':
          cardBeats (Card r1 s1) (Card r2 s2) = s1 == s2 && ra
nkBeats r1 r2
   |
71 | cardBeats (Card r1 s1) (Card r2 s2) = s1==s2 && rankBeats
 r1 r2
   |                                       ^^^^^^

DataTypes.hs:76:29: error:
    • No instance for (Eq Suit) arising from a use of '=='
    • In the first argument of '(&&)', namely
        'suit card1 == suit card2'
      In the expression:
        suit card1 == suit card2 && rankBeats (rank card1) (ra
nk card2)
      In an equation for 'cardBeats'':
          cardBeats' card1 card2
            = suit card1 == suit card2 && rankBeats (rank card
1) (rank card2)
   |
76 | cardBeats' card1 card2 =    suit card1 == suit card2
   |                             ^^^^^^^^^^^^^^^^^^^^^^^
Failed, no modules loaded.
Prelude> :r
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
d )
Ok, one module loaded.
*Main> cardBeats example_card_1 example_card_2
False
*Main> cardBeats (Card Ace Spades ) example_card_2
True
*Main> :r
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
d )

DataTypes.hs:73:12: error:
    • Conflicting definitions for 's1'
```

```
      Bound at: DataTypes.hs:73:12-13
                DataTypes.hs:73:25-26
    • In an equation for ‘f’
   |
73 | f (Card r1 s1) (Card r2 s1) = r1
   |            ^^^^^^^^^^^^^^^
Failed, no modules loaded.
Prelude> :r
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
d )
Ok, one module loaded.
*Main> :r
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
d )
Ok, one module loaded.
*Main> example_hand_0
Empty
*Main> example_hand_1
Add (Card {rank = Ace, suit = Hearts}) Empty
*Main> example_hand_2
Add (Card {rank = King, suit = Spades}) (Add (Card {rank = Ace
, suit = Hearts}) Empty)
*Main> :t Add
Add :: Card -> Hand -> Hand
*Main> :r
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
d )
Ok, one module loaded.
*Main> handBeats example_hand_2 (Card Jack Spades )
True
*Main> handBeats example_hand_2 (Card Jack Hearts  )
True
*Main> handBeats example_hand_2 (Card Jack Diamonds   )
False
*Main> handBeats Empty  (Card Jack Diamonds   )
False
*Main> handBeats Empty  (Card (Numeric 2) Hearts )
False
*Main> :r
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
d )
Ok, one module loaded.
```

```
*Main> betterCards example_hand_2 (Card Jack Hearts )
Add (Card {rank = Ace, suit = Hearts}) Empty
*Main> betterCards example_hand_2 (Card Jack Spades  )
Add (Card {rank = King, suit = Spades}) Empty
*Main> betterCards example_hand_2 (Card Jack Diamonds   )
Empty
*Main> :r
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
d )

DataTypes.hs:131:19: error:
    • Variable not in scope: betterCard :: Hand -> Card -> Han
d
    • Perhaps you meant 'betterCards' (line 101)
    |
131 | sameSuit hand s = betterCard hand (Card s (Numeric 1)) -
- quick hack!
    |                   ^^^^^^^^^^

DataTypes.hs:131:41: error:
    • Couldn't match expected type 'Rank' with actual type 'Su
it'
    • In the first argument of 'Card', namely 's'
      In the second argument of 'betterCard', namely
        '(Card s (Numeric 1))'
      In the expression: betterCard hand (Card s (Numeric 1))
    |
131 | sameSuit hand s = betterCard hand (Card s (Numeric 1)) -
- quick hack!
    |                                         ^

DataTypes.hs:131:44: error:
    • Couldn't match expected type 'Suit' with actual type 'Ra
nk'
    • In the second argument of 'Card', namely '(Numeric 1)'
      In the second argument of 'betterCard', namely
        '(Card s (Numeric 1))'
      In the expression: betterCard hand (Card s (Numeric 1))
    |
131 | sameSuit hand s = betterCard hand (Card s (Numeric 1)) -
- quick hack!
    |                                            ^^^^^^^^
```

```
Failed, no modules loaded.
Prelude> :r
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
d )

DataTypes.hs:131:19: error:
    • Variable not in scope: betterCard :: Hand -> Card -> Han
d
    • Perhaps you meant ‘betterCards’ (line 101)
    |
131 | sameSuit hand s = betterCard hand (Card (Numeric 1) s) -
- quick hack!
    |                   ^^^^^^^^^^
Failed, no modules loaded.
Prelude> :r
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
d )

DataTypes.hs:124:1: warning: [-Wincomplete-patterns]
    Pattern match(es) are non-exhaustive
    In an equation for ‘lowestCard’: Patterns not matched: Emp
ty
    |
124 | lowestCard (Add card Empty) = card
    | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^...
Ok, one module loaded.
*Main> :r
[1 of 1] Compiling Main             ( DataTypes.hs, interprete
d )
Ok, one module loaded.
*Main> ex
example_card_1  example_hand_0  example_hand_2  exponent
example_card_2  example_hand_1  exp
*Main> example_hand_2
Add (Card {rank = King, suit = Spades}) (Add (Card {rank = Ace
, suit = Hearts}) Empty)
*Main> chooseCard (Card Jack Spades ) example_hand_2
Card {rank = King, suit = Spades}
*Main> chooseCard (Card A Spades ) example_hand_2
Ace           Add           Applicative
*Main> chooseCard (Card Ace  Spades ) example_hand_2
Card {rank = King, suit = Spades}
```

```
*Main> chooseCard (Card Jack Hearts) example_hand_2
Card {rank = Ace, suit = Hearts}
*Main> chooseCard (Card Jack Diamonds) example_hand_2
Card {rank = King, suit = Spades}
*Main> :i Maybe
data Maybe a = Nothing | Just a         -- Defined in ‘GHC.Bas
e’
instance Applicative Maybe -- Defined in ‘GHC.Base’
instance Eq a => Eq (Maybe a) -- Defined in ‘GHC.Base’
instance Functor Maybe -- Defined in ‘GHC.Base’
instance Monad Maybe -- Defined in ‘GHC.Base’
instance Monoid a => Monoid (Maybe a) -- Defined in ‘GHC.Base’
instance Ord a => Ord (Maybe a) -- Defined in ‘GHC.Base’
instance Show a => Show (Maybe a) -- Defined in ‘GHC.Show’
instance Read a => Read (Maybe a) -- Defined in ‘GHC.Read’
instance Foldable Maybe -- Defined in ‘Data.Foldable’
instance Traversable Maybe -- Defined in ‘Data.Traversable’
*Main> Just Ace
Just Ace
*Main> :t Ace
Ace :: Rank
*Main> :t Just Ace
Just Ace :: Maybe Rank
*Main> :t Just
Just :: a -> Maybe a
*Main> :t min
min :: Ord a => a -> a -> a
*Main>
```