1 We have a problem...

What is this : "8♥" ?	1 A String.
Yes. What does it represent?	2 An *eight* of *hearts*, or 8♥.
What does "7♣ 6♦ 9♠" represent?	3 Some others cards: 7♣, 6♦, and 9♠.
Right. And what does "A♥ K♥ Q♥ J♥ T♥" represent?	4 It represents victory: it's a *royal flush*.
What is the best hand we can do with the following: 4♦ 2♦ K♠ K♦ 9♦ 3♣ 6♦ ?	5 A *flush*.
And with 9♣ A♥ K♠ K♦ 9♦ 3♣ 6♦ ?	6 Two *pairs*.
Correct. And with A♣ Q♣ K♠ K♦ 9♦ 3♣ ?	7 Nothing, because there are less than seven cards.
And with 9♥ 5♠ ?	8 Nothing, for the same reason.
That's right. What about K♣ 9♠ K♠ K♦ 9♦ 3♣ 6♦ ?	9 It's a *full house*. Say, why are you showing me all these cards?
Because we have a problem, and I wanted to be sure you know the basics about *Poker*.	10 Show me what the problem is.

We have to write a program which, given this input:

```
K♣ 9♠ K♠ K♦ 9♦ 3♣ 6♦
9♣ A♥ K♠ K♦ 9♦ 3♣ 6♦
A♣ Q♣ K♠ K♦ 9♦ 3♣
9♥ 5♠
4♦ 2♦ K♠ K♦ 9♦ 3♣ 6♦
7♠ T♠ K♠ K♦ 9♦
```

– right – ..would output this:

```
K♣ 9♠ K♠ K♦ 9♦ 3♣ 6♦ Full House (winner)
9♣ A♥ K♠ K♦ 9♦ 3♣ 6♦ Two Pair
A♣ Q♣ K♠ K♦ 9♦ 3♣
9♥ 5♠
4♦ 2♦ K♠ K♦ 9♦ 3♣ 6♦ Flush
7♠ T♠ K♠ K♦ 9♦
```

What do you see?

Do you think we can solve the problem?

11 These are the cards of some players in a game of *Texas Hold'em*[1]. Right?

12 I see.

13 Some lines are just left as they are. Some lines are marked with the ranking of the best possible hand given the cards on the line. The line with the best ranking is marked as the winner.

14 Yes, I suppose, provided we have good tools.

[1] See `http://rubyquiz.com/quiz24.html`

2 Haskell Tools

What is the effect of this program?

```
main :: IO ()
main = putStrLn "42"
```

[15] It prints the Answer to the Ultimate Question of Life, The Universe, and Everything. It is a very simple *Haskell* program.

How many functions are defined in this program?

[16] Only one, called *main*.

What is the type of the function?

[17] This function is of type `IO ()`, which means it will perform some input/output operations, and yield a void result that we don't care about.

How do we run this program?

[18] One way is to save it to a file, for example `answer.hs`, and then launch `runhaskell`:

```
runhaskell answer.hs ↩
42
```

How do we execute some *Haskell* code in an interactive way, without having to write a program?

[19] One way is to launch *ghci*: and try things there:

```
ghci ↩
```

```
GHCi, http://www.haskell.org/ghc/ :? for help
6 * 7 ↩
 42
sqrt 2 ↩
 1.4142135623730951
4 > 3 ↩
True
```

What other tools do we have to solve the problem?

[20] Tools that we will code in *Haskell*, I suppose.

That is right. What is: '♥' ?

[21] It's a Char value.

What is: '♠' ?

[22] It's another Char value.

What is: "♥♣♦♠" ?

[23] It's a list of Chars, or String value.

How do you get information about the type of a value?

[24] Using `:type` and `:info` in *ghci*:[2]

```
:type '♥' ↩
 '♥' :: Char
:type "♥♣♦♠" ↩
 "♥♣♦♠" :: [Char]
:info String ↩
 type String = [Char]
```

What is the value of this expression: length "♥♠♦♣" ?

[25] 4.

What is the value of: length [3,2,7,6,8] ?

[26] 5.

What is the type of: length "♥♠♦♣" ?

[27] Int.

What is the type of: length ?

[28] [*a*] → Int[3]

How does this Haskell expression:
length :: [*a*] → Int
read?

[29] length is a function from list of any type, to Int.

What is the value of the expression:
words "time flies like an arrow"

[30] ["time","flies","like","an","arrow"].

What is the type of: words ?

[31] It's a function from String to list of Strings:

```
:type words ↩
 words :: String → [String]
```

What is the value of: 4 == 5?

[32] False.

What is the value of: 4 == 3 + 1?

[33] True.

---

[2] Unicode hex values for '♥', '♣','♦' and '♠' are: 2665, 2663, 2666 and 2660. Using '*H*', '*C*', '*D*' and '*S*' would also work very well.

[3] This is true for old versions of *GHC* only. The type signature of length is actually *Foldable t* ⇒ *t a* → Int. Foldable types are containers that can be folded into a summary value ([*a*] is such a type).

What is the value of: `4 == 2 * 1 + 1`?

[34] False.

What is the value of: `4 == 2 * (1 + 1)`?

[35] True.

What is the type of the expression `4 == 5`?

[36] It's a boolean expression:

```
:type 4 == 5 ↩
 4 == 5 :: Bool
```

Let's create incidents. What is the value of:
`'♥' == True`?

[37] It's not a valid *Haskell* expression:

```
Couldn't match expected type ‘Char’ with actual type ‘Bool’
In the second argument of ‘(==)’, namely ‘True’
    In the expression: '♥' == True
```

What do you infer?

[38] If the first argument of `(==)` is a Char, the second argument should be also a Char.

What else can you infer?

[39] There is a function `(==)` that takes two arguments:

```
(==) 3 4 ↩
 False
```

What is the value of: `(+) 4 5`?

[40] 9.

What do you conclude?

[41] Operators are functions too.

What is the value of expression: `4 + False`?

[42] Again, it's not a valid expression:

```
No instance for (Num Bool) arising from a use of ‘+’
Possible fix: add an instance declaration for (Num Bool)
In the expression: 4 + False
```

But the message is different.

Let's try to understand the message.
What is the type of `(+)`?

[43] `(+) :: Num a ⟹ a → a → a`.

Here some `:info` about Num:

```
Class Num a where
  (+) :: a → a → a
  (*) :: a → a → a
  (-) :: a → a → a
  negate :: a → a
  abs :: a → a
  signum :: a → a
  fromInteger :: Integer → a
instance Num Integer
instance Num Int
instance Num Float
instance Num Double
```

Explain in your own words what the error message is about.

[44] There are 4 types that are instances of the class Num, and Bool is not one of those types.

The operator (+) requires operands of a type that is an instance of Num.

Therefore the expression: `4 + False` is invalid.

Making Bool an instance of Num would make the expression a valid expression. (Although I don't think it's a good idea).

What is the value of the expression: `subtract 1 10` ?

[45] 9.

What is the value of: `subtract 25 100` ?

[46] 75.

What is the type of: `subtract 1 10` ?

[47] Num $a \Longrightarrow a$.

What is the type of `subtract 1` ?

[48] Num $a \Longrightarrow a \rightarrow a$.

What is the type of `subtract` ?

[49] Num $a \Longrightarrow a \rightarrow a \rightarrow a$.

Let's suppose that *dec* = `subtract 1`.
What is the value of *dec* 1000?

[50] Let's ask:

```
let dec = subtract 1 ↩
dec 1000 ↩
 999
```

We just created a new function.

Let's create another one: *inc* = (+) 1.
What is the value of: *inc* (*inc* (*inc* 1))?

[51] Let's ask again:

```
let inc = (+) 1 ↩
inc (inc (inc 1)) ↩
 4
```

And another one: *dbl* = (∗) 2.
What is the value of *dbl* 6?

[52] I think I know the answer.

```
let dbl = (*) 2 ↩
dbl 6 ↩
 12
```

Good. Now let's define something new:
foo = *inc* ∘ *dbl*.
What is the value of *foo* 10?

[53] I have no idea. Let's try:[4]

```
let foo = inc ∘ dbl ↩
foo 10 ↩
 21
```

Why is the result 21?

[54] It is 10 multiplied by 2, plus 1.

What is the type of : *o* ?

[55]

```
:type (∘) ↩
 (∘) :: (b → c) → (a → b) → a → c
```

That is the most complex function I've seen.

[4] In *ghci*, replace ∘ with . .

Suppose we have the following:

$fry :: Batter \to Pancake$
$mix :: (Flour, Milk, Egg) \to Batter$

Then suppose we define a new function:

$cook = fry \circ mix$

What would be the type of this new function?

How would you explain the role of $\circ$ in this expression: $f \circ g$?

In fewer words?

Suppose we want a new function: $wc :: String \to Int$.
The job of this function is to count words in a String.

Create the *wc* function in *ghci*.

Perfect! Do you want to solve the problem?

---

56 Interesting!

---

57 I think it should be something like:

$cook :: (Flour, Milk, Egg) \to Pancake$

---

58 The $\circ$ function takes two functions f and g and compose them into a new function. Applying this new function to an argument x is the same as applying f to the result of applying g to x.

---

59 $(f \circ g)\ x = f\ (g\ x)$

---

60 That seems easy to do.

---

61 Here it is:

```
let wc = length ∘ words ↩
wc "a four words sentence" ↩
4
```

---

62 Let's make some tea first.

3 Simple Values

How many cards are represented in this String:
"A♥ K♥ Q♥ J♥ T♥" ?

63 Five.

Does (length ∘ words) *s*
give us the number of cards in *s*
when *s* is equal to "Q♥ #! J♥" ?

64 It gives us 3, but that's not the exact number of cards in the String.

Why not?

65 Because "#!" does not represent a valid card.

Does "Q♥" represent a valid card?

66 Yes.

Some values of the type String can represent a valid card, some cannot.

67 Right.

What do we need to do with cards?

68 Compare them, sort them, and group them.

What is the value of "4♥" < "5♣"?

69 True.

What is the value of "T♥" < "A♣"?

70 False.

Some comparisons of cards as represented by String work, some don't.

71 Exactly.

What type could represent valid cards for *all* values of this type?

72 I don't know. Maybe we need to make our own type.

What would you call that type?

73 Card.

How would you describe the type Card in your own words?

74 The type Card is the set of all possible cards in the game of *Poker*.

Can you give examples of values of such a type?

75 *Queen of Hearts*, *Two of Spades*, *Ace of Clubs*, *Five of Diamonds*.

What should be required for a value to be of the type Card?

76 It should have a *rank* and *suit*.

What are the possible ranks?

77 *Two*, *Three*, *Four*, *Five*, *Six*, *Seven*, *Eight*, *Nine*, *Ten*, *Jack*, *Queen*, *King*, *Ace*.

What are the possible suits?

78 *Hearts*, *Spades*, *Diamonds*, *Clubs*.

Let's define a new type for the suits.

79 I don't know how to do that.

Do you know about the Bool type?

80 Yes. It is defined with two boolean values: False, True.

What can we do with Bool values?

81 We can compare them for equality or inequality:

```
> False == False
True
> True /= True
False
> False < True
True
```

How is Bool defined?

82 Like this:

```
data Bool = False | True
```

How do you define the type Suit?

83 Like this, I suppose:

```
data Suit = Hearts | Spades | Diamonds | Clubs
```

What should we be able to do with values of type Suit?

84 Compare them for equality, for instance when looking for a *flush* in a group of cards.

What does the expression: `Diamonds == Diamonds` yield?

85 It's an error:

```
No instance for (Eq Suit) arising from a use of '=='
Possible fix: add an instance declaration for (Eq Suit)
```

I don't understand.

What it means is that there are currently no rules for comparing Suit values.

86 We didn't define any.

If we declare the type Suit to be an instance of the class Eq, then default rules will be used for comparing.

87 What are those default rules?

A value is equal to itself and different from all other values.

88 Obviously this is what we need. How do we do that?

We add a deriving clause to the declaration:

```
data Suit = Hearts | Spades | Diamonds | Clubs
    deriving Eq
```

89 Great! Now we can compare values:

```
> Diamonds == Diamonds
True
> Diamonds == Clubs
False
```

Let's define a new type for the ranks now.

90 I think I know how to do that:

```
data Rank = Two | Three | Four | Five | Six | Seven |
      Eight | Nine | Ten | Jack | Queen | King | Ace
```

Why not use numbers for the Rank values from 2 to 10?

91 Rank is a new data type. We can create values of this type by using data constructors. The symbols 2, 3, 4... are already used to denote numbers, and cannot be used as data constructors.

What should we be able do with values of type Rank?

92 Compare them for equality or inequality. For example when looking for a *pair* or a *straight* in group of cards, or just sorting the cards by Rank value.

What does the expression: Ten < Jack yield?

93 It's an error again:

```
No instance for (Ord Rank) arising from a use of '<'
Possible fix: add an instance declaration for (Ord Rank)
In the expression: Ten < Jack
```

In that case we should declare the type to be deriving Eq, and to be deriving Ord as well:

```
data Rank = Two | Three | Four | Five | Six | Seven |
      Eight | Nine | Ten | Jack | Queen | King | Ace
    deriving (Eq, Ord)
```

94 Here again, we do not define the rules for comparing values, determining which value is greater and which is lower..

What is the value of: Two < Three ?

95 True.

What is the value of: Ace > King ?

96 True.

What is the default rule for inequality?

97 The default rule is that the first value declared is the lower, the next one is greater, and so on.

Correct.
What is the value of the expression:
compare 42 4807 ?

98 LT

What is the value of the expression:
compare "TIME" "FLIES" ?

99 GT

Right. And of the expression:
compare King Ace ?

100 Since we correctly defined our values, it's: LT.

What is the value of the expression:
compare Queen Queen ?

101 It is EQ.

What about compare Eight Jack ?

102 It is LT.

And compare Five Three ?

103 It is GT.

What is the type of the value returned by compare ?

104 Here's what *ghci* tells us:

```
:info compare
class Eq a ⟹ Ord a where
  compare :: a → a → Ordering
:info Ordering
data Ordering = LT | EQ | GT
```

Can you explain the compare function in your own words?

105 compare can compare values of any type which is deriving of both type classes Eq and Ord. The result of the comparison is one of the three values LT, EQ, GT meaning respectively *lower than*, *equal*, and *greater than*.

What do we have so far, with regard to representing cards in our poker hand program?

106 We have our own types Suit, and Rank to represent values that will be used for comparisons.

What do we still need?

107 We need to create a type Card to correctly represent cards.

Can you define the type?

108 I'm afraid I can't. I don't know yet how to define a type as a combination of other types.

What is the type of the expressions :
('A', True), ('B', False), ('E', True) ?

109 (Char, Bool)

What is the type of: ('A', True, 'a') ?

110 (Char, Bool, Char)

What do we call such combinations of values?

111 We call them *tuples*. When formed with two values, we can call them *pairs*.

How would you describe the type (Char, Bool) in your own words?

112 It's a set that is a product of all possible values of Char times all possible values of Bool.

What is the type of the expression: (Queen, Hearts) ?

113 I think it is: (Rank, Suit)

How would you describe the type (Rank, Suit) in your own words?

114 It's the set of all possible Ranks times all possible Suits. It's the type we are looking for.

So how would you define the type Card?

115 Exactly like this: type Card = (Rank, Suit).

How would you explain such declaration in your own words?

116 The type Card is equivalent to a *tuple of* Rank *and* Suit.

What do we want to do with values of type Card?

117 We want to know their Rank value, so that we can search for a *pair*, a *straight*, and so on.
We also want to know their Suit value, so that we can search for a *flush*.
And and of course we want to extract Card values from Strings.

What is the value of the expression: fst (65,'A') ?

118 65.

What is the value of the expression: fst (False, 42) ?

119 False

What is the type signature of the function fst ?

120 fst :: (*a*, *b*) → *a*

Can you explain in your own words what this function does?

121 fst takes a pair as its argument, and yields the first element of the pair.

What is the value of the expression:
fst (Queen, Hearts) == Queen ?

122 True, of course.

How would you define a function which takes a Card and gives its Rank, given what we now know about tuples?

123 That's easy:

```
rank :: Card → Rank
rank = fst
```

What is the value of the expression: snd (65,'A') ?

124 'A'.

What is the value of the expression: snd (False, 42) ?

125 42

What is the type of snd ?

126 snd :: $(a, b) \rightarrow b$
snd takes a pair, and yields the second element of the pair.

What is the value of the expression:
snd (Queen, Hearts) == Hearts ?

127 True obviously.

Define a function which takes a Card and gives its Suit.

128 Here it is:

```
suit :: Card → Suit
suit = snd
```

Can we determine if two Card values are equals?

129 Let's try:

```
> let c = (Queen,Hearts)
> let d = (King,Spades)
> c == d
False
> d /= c
True
```

We can.

Using your example values, can we tell if $c < d$?

130 No we can't:

```
No instance for (Ord Suit) arising from a use of '<'
Possible fix: add an instance declaration for (Ord Suit)
In the expression: c < d
```

Should we add an instance declaration for Suit as *ghci* suggest?

131 Probably not a good idea, as we will want to order Cards based on their Rank only, not on their Rank *and* Suit.

What do we have so far?

132 We have types to represent cards values, rank values, and suit values. We also have functions on those types.

How does that help us solving our problem?

133 It gives us safety: the guarantee that we will be dealing with correct values of Rank and Suit instead of possibly incorrect Chars.

Here is the code we have so far in order to solve our problem:

```
data Suit = Hearts | Spades | Diamonds | Clubs
    deriving (Eq)
data Rank = Two | Three | Four | Five | Six | Seven |
     Eight | Nine | Ten | Jack | Queen | King | Ace
    deriving (Eq, Ord)
type Card = (Rank, Suit)
rank :: Card → Rank
suit :: Card → Suit
rank = fst
suit = snd
```

Listing 1: Card.hs

How do we use this? Give some examples.

134 Here are some examples:

```
> let q = (Queen, Hearts)
> let k = (King, Spades)
> let a = (Ace, Diamonds)
> let j = (Jack, Clubs)
> let z = (Ace, Hearts)
> rank q < rank k
True
> rank a > rank j
True
> rank a == rank z
True
> suit q /= suit k
True
> suit q == suit z
True
```

What does the type Card give us?

135 It gives us values on which we can apply two functions, *rank* and *suit*.

What else do we need?

136 A function to create Card values from a String.

→ Use the data construct to create a new type for the values of your specific domain.

→ Values of a type deriving Eq can be compared for equality with == and /=.

→ Values of a type deriving Eq,Ord can be compared for inequality with < > <= >= and compare.

→ You can combine different types into one, using the *tuple* construct (,). Values of a tuple type can be compared for equality if the types composing the tuple are instance of the class Eq, and the same rule applies for inequality, with the class Ord.

→ The type construct allows for the definition of a type synonym, as in type String = [Char].