1. What is the difference between the type Char and the type String? Do the two expressions "a" and 'a' represent the same value?
2. Given the function definition
3. square :: Int -> Int

square x = x \* x

and the previous definitions of inc and double. What is the value of

* 1. inc (square 5)
  2. square (inc 5)
  3. average (inc 3) (inc 5)

1. If you remove the optional type annotation from the above definition of square, what type will the compiler infer? You can find out by pressing ⌘-i in Haskell for Mac, while your cursor is on the function name, or by typing :type square or :t square in GHCi.
2. Which of the following identifiers can be function or variable names?
   1. square\_1
   2. 1square
   3. Square
   4. square!
   5. square'

If you are unsure, replace the function name in your square definition from Exercise 2 with each of these identifiers to see whether the compiler complains and what the error message looks like.

1. Define a new function showResult, that, for example, given the number 123, produces a string as follows:

showResult 123  ⇒  "The result is 123"

Use the function show in the definition of the new function.

1. Write a function showAreaOfCircle which, given the radius of a circle, calculates the area of the circle,

showAreaOfCircle 12.3  ⇒  "The area of a circle with radius 12.3cm is about 475.2915525615999 cm^2"

Use the show function, as well as the predefined value pi :: Floating a => a to write showAreaOfCircle.

1. Write a function sort2 :: Ord a => a -> a -> (a, a) which accepts two Int values as arguments and returns them as a sorted pair, so that sort2 5 3 is equal to (3,5). How can you define the function using a conditional, how can you do it using guards?
2. Consider a function

almostEqual :: Eq a => (a, a) -> (a, a) -> Bool

which compares the values of two pairs. It returns True if both pairs contain the same values, regardless of the order. For example, almostEqual (3,4) (4,3) is True, but almostEqual (3,4) (3,5) is False. Which of the following definitions return the correct value? Which of the definitions would you consider good style? Why?

(The operator (&&) is logical ”and”, the operator (||) is logical ’or’, and (==) tests if two values are equal. The first two are of type Bool -> Bool -> Bool; the third is of type Eq a => a -> a -> Bool.)

almostEqual (x1, y1) (x2, y2)

| (x1 == x2) && (y1 == y2) = True

| (x1 == y2) && (y1 == x2) = True

| otherwise = False

almostEqual (x1, y1) (x2, y2)

| (x1 == x2) = (y1 == y2)

| (x1 == y2) = (y1 == x2)

| otherwise = False

almostEqual pair1 pair2

= (pair1 == pair2) || (swap pair1 == pair2)

**where**

swap (x,y) = (y,x)

almostEqual pair1 pair2

= (pair1 == pair2) || (swap pair1 == swap pair2)

**where**

swap (x,y) = (y,x)

almostEqual (x1, y1) (x2, y2)

= **if** (x1 == x2)

**then**

**if** (y1 == y2)

**then** True

**else** False

**else**

**if** (x1 == y2)

**then**

**if** (x2 == y1)

**then** True

**else** False

**else** False

1. Define a function isLower :: Char -> Bool which returns True if a given character is a lower case letter. You can use the fact that characters are ordered, and for all lower case letters ch we have ′a′ ≤ ch and ch ≤ ′z′. Alternatively, you can use the fact that ['a'..'z'] evaluates to a list containing all lower case letters.
2. Write a function mangle :: String -> String which removes the first letter of a word and attaches it at the end. If the string is empty, mangle should simply return an empty string:

mangle "Hello"   ⇒   "elloH"

mangle "I"   ⇒   "I"

mangle ""   ⇒   ""

1. Implement division on Int, divide :: Int -> Int -> Int using the list functions described in this section. Hint: first, write a function that returns all the multiples of a given number up to a specific limit.

divide 5 10   ⇒   2

divide 5 8   ⇒   1

divide 3 10   ⇒   3

1. Define the function length :: [a] -> Int. It is quite similar to sum and product in the way it traverses its input list. Since length is defined in the Prelude, don't forget to hide it by adding the line

**import** Prelude **hiding** (length)

to your module.

1. What are the values of the following expressions and what is wrong with the ones that give errors?

1:[2,3,4]

1:2:3:4:[]

[1,2,3]:[4..7]

[1,2,3] ++ [4..7]

1:['a','b']

"abc"++"cd"

"a":"bCc"

"a" ++ "bCc"

'a':'b'

'a':"b"

[1,4,7] ++ 4:[5:[]]

[True,True:[]]

True:[True,False]

1. Write a recursive function fact to compute the factorial of a given positive number (ignore the case of 0 for this exercise).

fact n = 1 \* 2 \* ... \* n

Why is the function fact a partial function? Add an appropriate error case to the function definition.

1. In the previous chapter, we introduced the ellipsis list notation in Haskell, which allows us to write

[m..n]

as shorthand for the list

[m, m+1, m+2, ..., n]

for numbers m and n, with n greater or equal m. Write a recursive function enumFromTo which produces such a list given m and n, such that

enumFromTo m n = [m..n]

As enumFromTo is a Prelude function, you have to add the line

**import** Prelude **hiding** (enumFromTo)

to your program.

1. Write a recursive function countOdds which calculates the number of odd elements in a list of Int values:

countOdds [1, 6, 9, 14, 16, 22] = 2

Hint: You can use the Prelude function odd :: Int -> Bool, which tests whether a number is odd.

1. Write a recursive function removeOdd that, given a list of integers, removes all odd numbers from the list, e.g.,

removeOdd [1, 4, 5, 7, 10] = [4, 10]

1. Challenge: At the end of the last screencast, demonstrating the implementation of closestPoint :: Point -> [Point] -> Point, we mentioned that the final implementation is less efficient than one might hope, as it uses the distance functions twice —instead of once— per recursive step. Improve the implementation to avoid that inefficiency.
2. mplement a function colouredFTree :: Float -> Int -> Colour -> Line -> Picture that elaborates on fractalTree by accepting the colour of the tree as an additional argument.
3. Vary colouredFTree by using the fade function, which we discussed in the context of spiral rays, to incrementally alter the colour in each recursive step.
4. Vary colouredFTree further by implementing and using factor as demonstrated in the last screencast.