M21274 – MATHFUN
Discrete Mathematics and Functional Programming

Worksheet 1: Introduction to Functional Programming

Introduction

Hopefully, you have completed **Worksheet 0** by this time; if not please finish it before starting this one. This worksheet asks you to write more functions in the Haskell language.

At your own pace, work your way through this worksheet and ask questions during the practical session. If you are confused by what a particular operator or function does, look it up on Hoogle and find out what it does (also check out the examples).

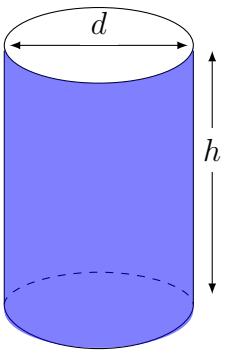
Once you have written and tested your solutions for the programming exercises, ask us during the **next** practical to check your work and give you a sign-off.

Worked examples

We have included two worked examples in every worksheet to help you with the exercises. Open the folder that you created as part of Worksheet 0 in your favourite editor. Then create a new file and save it with the name `Week1.hs`. Now read the questions below and follow the steps in their solutions.

Worked example 1

Question: Write a function `sideOfCylinder` that, given the diameter of the top of the cylinder and its height, returns the area on its side. For example, in the figure below, given the values `d` and `h`, `sideOfCylinder` should return the area coloured in blue. Your function must pass every test case in table 1.



Test	Output
`sideOfCylinder 2 10`	62.831856
`sideOfcylinder 8 7`	175.9292

Table 1: `sideOfCylinder` test cases

Solution: Here is the formula for the side of a cylinder: $circumference\_of\_top * height$. Let us approach this question methodically and write a helper function that calculates the circumference of a circle given its diameter.

The `circumferenceOfCircle` function defined below takes the diameter `d`, which is a `Float`, and returns `pi * d` (the circumference of type `Float`). Add these lines to `Week1.hs`:

```
1 circumferenceOfCircle :: Float -> Float
2 circumferenceOfCircle d = pi * d
```

Notice that the constant `pi` is already defined in Prelude (Haskell's standard module). Save your changes so that we can test `circumferenceOfCircle`.

Reload your script by writing the command below in GHCi, after the prompt `Prelude>` (in WinGHCi, you can reload with Ctrl + R):

```
:reload
```

Now test `circumferenceOfCircle` by evaluating the following expression in the shell. It should produce the output 15.707964.

```
circumferenceOfCircle 5
```

The `sideOfCylinder` takes two `Floats`: diameter `d` and height `h`. This function returns another `Float`, which is the result of the multiplication of `h` by the circumference of the top (calculated by passing `d` to `circumferenceOfCircle`). Add the highlighted to `Week1.hs`:

```
1 circumferenceOfCircle :: Float -> Float
2 circumferenceOfCircle d = pi * d
3
4 sideOfCylinder :: Float -> Float -> Float
5 sideOfCylinder d h = circumferenceOfCircle d * h
```

Notice that you do not need brackets around `circumferenceOfCircle d`. Save your script and reload it. Then check that it works correctly by running the test cases in table 1.

Worked example 2

Question: Write a function `all3CanDrink` that takes the age of 3 people and returns `True` if they are of legal drinking age (18+) otherwise `False`. Your function must satisfy the following test cases:

Test	Output
`all3CanDrink 19 21 18`	`True`
`all3CanDrink 19 25 17`	`False`

Solution: Let's begin with a simpler function that takes one person's age (as an `Int`) and returns `True` if they can drink, otherwise it returns `False`. Notice that the return type of this function is `Bool` (short for Boolean):

```
7 canDrink :: Int -> Bool
8 canDrink age = age >= 18
```

Test this function with different numbers before proceeding. We can use the "and" operator (`&&`) to write for `all3CanDrink`. Append the following to `Week1.hs`:

```
7 canDrink :: Int -> Bool
8 canDrink age = age >= 18
9
10 all3CanDrink :: Int -> Int -> Int -> Bool
11 all3CanDrink a b c = canDrink a && canDrink b && canDrink c
```

Save and reload your script. Then try the test cases provided in the table above. For example, see whether the following expression evaluates to `True`:

```
all3CanDrink 19 21 18
```

Programming exercises

For each of the questions in this section, write a function at the end of `Week1.hs`. Make sure to **test each function using the provided table** before moving onto the next question.

1. Write a function with the following type that multiplies its argument by 10:

```
timesTen :: Int -> Int
```

Test	Output
`timesTen 5`	`50`
`timesTen 60`	`600`

2. Write a function which gives the sum of three integers:

```
sumThree :: Int -> Int -> Int -> Int
```

Test	Output
`sumThree 12 9 20`	`41`

3. Using the constant `pi` and the power operator `^`, write a function that gives the area of a circle given its radius:

```
areaOfCircle :: Float -> Float
```

Test	Output
`areaOfCircle 4`	`50.265484`

4. Write a function that returns the volume of a cylinder given its height and the radius of its top. It must use `areaOfCircle` which is the answer to the previous question.

```
volumeOfCylinder :: Float -> Float -> Float
```

Test	Output
`volumeOfCylinder 10 9`	`2544.6902`

5. Write a function that takes four floats representing the coordinates x_1, y_1, x_2, y_2 of two points, and gives the distance between the points:

```
distance :: Float -> Float -> Float -> Float -> Float
```

Hint: You need the `sqrt` (square root) function and the formula shown below.

$$distance = \sqrt{(y_1 - y_2)^2 + (x_1 - x_2)^2}$$

Test	Output
`distance 9 15 5 12`	`5.0`
`distance 1.5 1.75 (-1.25) (-1.50)`	`4.2573466`

6. Use the /= operator (not equal to) to write a function that returns `True` if and only if all three of its arguments are different from each other:

```
threeDifferent :: Int -> Int -> Int -> Bool
```

Test	Output
`threeDifferent 10 11 12`	`True`
`threeDifferent 10 12 12`	`False`

Hint: You may need to use logical operators (and, or) for this question. Find out the syntax by referring to the documentation page on Boolean operators.

7. Using the `mod` function, write a function that tests if one integer is divisible by another:

```
divisibleBy :: Int -> Int -> Bool
```

Test	Output
`divisibleBy 10 2`	`True`
`divisibleBy 10 3`	`False`

8. Using `divisibleBy` from the previous question, write a function which determines if its argument is an even number:

```
isEven :: Int -> Bool
```

Test	Output
`isEven 10`	`True`
`isEven 11`	`False`

9. Write a function with the signature below that returns the average of three integers.

```
averageThree :: Int -> Int -> Int -> Float
```

Test	Output
`averageThree 68 72 56`	`65.333336`
`averageThree 10 11 12`	`11.0`

Hint: The / operator (division) requires both operands to be fractional. You need to coerce your `Int` types using the `fromIntegral` function before doing a division. For example, the function below uses `fromIntegral` to give a fractional value from the `percent` variable (which is an `Int`) so that it can be divided by 100. You don't need to use `fromIntegral` on 100 because it can actually be used as a fractional value, even though it looks like an integer.

```
applyDiscount :: Float -> Int -> Float
applyDiscount price percent = price * (1 - fromIntegral percent / 100)
```

10. Use a conditional expression (and not the built-in function `abs`) to write a function that returns the absolute value of an integer (i.e. gives a non-negative value):

```
absolute :: Int -> Int
```

Note: Evaluating the following expression will cause an error. This is because "-" is interpreted as an infix (not a prefix) subtraction. See the test cases in the table that show how to test `absolute` with negative numbers.

```
absolute -3
```

Test	Output
`absolute (-10)`	10
`absolute 0`	0
`absolute 10`	10

Hint: Here is an example on how to use the `if then else` expression:

```
passOrFail :: Int -> String
passOrFail mark = if mark >= 40 then "Pass" else "Fail"
```