Development 8 - Exercises
Unit 1

Exercise 1:

Implement a function

```
let allNumber = (n:  number):  string
```

that returns a string containing all numbers from 0 to `n`. Separate the numbers with a white space.

Exercise 2:

Implement a function

```
let allNumberRev = (n:  number):  string
```

that returns a string containing all numbers from `n` to 0. Separate the numbers with a white space.

Exercise 3:

Implement a function

```
let allNumberRev = (lower:  number) => (upper:  number):  string
```

that returns a string containing all numbers between `lower` and `upper`. Separate the numbers with a white space.

Exercise 4:

Implement a function

```
let allNumberRev = (lower:  number) => (upper:  number):  string
```

that returns a string containing all numbers between `lower` and `upper`. Separate the numbers with a white space.

Exercise 5:

Implement a function

```
let allEvenRange = (lower:  number) => (upper:  number):  string
```

that returns a string containing all even numbers between `lower` and `upper`. Separate the numbers with a white space.

Exercise 6:

Implement a function

```
let drawLine = (length:  number):  string
```

that returns a string containing `length` asterisks.

Exercise 7:

Implement a function

```
let drawSymbols = (symbol:  string) => (length:  number):  string
```

that returns a string containing `length` repetitions of `symbol`.

Exercise 8:

Implement a function

```
let toBinary = (n:  number):  string
```

that returns a string containing the binary representation of the input number (it must be positive). The binary representation is obtained using the following procedure:

1. Add to the end of the string the remainder of the division between the current number and 2.
2. Repeat the previous step until the number is 0. In this case simply don't add anything.

Exercise 9:

Implement a function

```
let toBinary = (n:  number) => (base:  number):  string
```

that returns a string containing the representation of the input number in an arbitrary base (the number must be positive). The algorithm is the same as above except you must take the remainder of `n` divided by `base`.