Exercise 1

First we have to assign the values to the variable names.

```
x <- 6
y <- -2.5
```

Now we can start with our calculations.

(a) We can use `abs()` to calculate the absolute value of `y`. The result will then be used for the exponentiation.

```
abs(y)^3

# [1] 15.625
```

(b) As we learned in the lecture `e` is not an available constant in R. However, we can use the `exp` function.

```
exp(x)

# [1] 403.4288
```

(c) No further explanations...

```
(x + y) * 5

# [1] 17.5
```

(d) Here, we have to use parantheses to define the precedence of the operations.

```
1 / (x + y)

# [1] 0.2857143
```

(e) Can be written in R just like that.

```
sin(1.5 * pi)

# [1] -1
```

(f) There is no function in R to calculate the x-th root in R. Instead we have to use exponentiation.

```
65^(1/x)

# [1] 2.005175
```

(g) The ceiling functions always round up and is available as `ceiling()` in R.

```
ceiling(19 / x )

# [1] 4
```

(h) Using the floor function `floor()`.

```
floor(-17 / y)

# [1] 6
```

This is actually the definition of integer division. So we could equally well use the `%/%` operator.

```
-17 %/% y

# [1] 6
```

(i) The modulo operator gives back the remainder of an integer division.

```
17 %% x

# [1] 5
```

Please note: R also has a function `trunc()`, which removes the fractional part of any number. For positive values this is equal to `floor()` whereas for negative values it is equal to `ceiling()`

```
floor(2.8)

# [1] 2

trunc(2.8)

# [1] 2

ceiling(-2.8)

# [1] -2

trunc(-2.8)

# [1] -2
```

Exercise 2

(a) The `:` operator can be also used to create decreasing series of numbers.

```
10:1

#  [1] 10  9  8  7  6  5  4  3  2  1
```

(b) For an increment different than 1 we have to use the `seq()` function.

```
seq(1, 19, 3)

# [1]  1  4  7 10 13 16 19
```

(c) The numbers are all powers of 2. As we learned, shorter vectors are recycled in operations, which we can use as follows:

```
2^(0:10)

#  [1]    1    2    4    8   16   32   64  128  256
# [10]  512 1024
```

(d) If we give `rep()` a vector of length >1 for the `times` argument it will repeat each element and not the whole vector.

```
rep(1:5, times = 1:5)

#  [1] 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5
```

(e) Just as the one in (c) but this time we have to add 1.

```
2^(0:10) + 1

#  [1]    2    3    5    9   17   33   65  129  257
# [10]  513 1025
```

(f) This time we repeat the whole vector. Please compare this to (d).

```
rep(c(0, 1), times = 5)

#  [1] 0 1 0 1 0 1 0 1 0 1
```