Chapter 3

Sample Sheet

General information & code examples

Quorum is an evidence-based programming language, designed from the outset to be easily understood and picked up by beginners. One of the design decisions taken includes the full omit of brackets when defining scopes. Keywords in the language make use of a more natural mapping to the real world, such as "text" for strings, "number" for doubles and floats and "repeat" for loops. Conditional statements such as if-statement are always ended with the keyword "end" which specifies the end of scope.

Data types

```
integer a = 5
number b = 10.2
text c = "John"
boolean d = true
```

Type conversion:

```
text someText = "5.7"

number someNumber = cast (number, someText)
```

Simple operation with arrays and conditional statements

The following code creates an array a with some randomly placed elements. It then sorts the array and iterates through the array to create an output with all the elements.

```
use Libraries.Containers.Array
action Main
        text unordered = "fdebaac"
        Array<text> a = unordered:Split("")
        a:Sort()
        integer i = 0
```

```
        text out = ""
        repeat while i < a:GetSize()
                out = out + a:Get(i) + ";"
                i = i + 1
        end
        output out
end
```

Output is: `a;a;b;c;d;e;f;`

This is an example of an action using if- else statements

```
action checkIntervals(integer i)
    if i < 10
        output "it is less than 10"
    elseif i = 10 or i > 10 and i <= 15
        output "it is less than or equal to 15 but greater or equal to 10"
    else
        output "it is greater than 15"
    end
end
```

Classes & Inheritance

To demonstrate classes and inheritance in quorum, we use the example of the animal family felidae and a cat belonging to that family:

First the superclass felidae looks like this:

```
class felidae
    text name = "Sebastian"

    public action Paws() returns integer
        return 4
    end

    action Purr()
        output name + ": rhrhrhrhrhrhrhrhrhrhrhrh"
    end
end
```

We then create the cat subclass like this:

```
class cat is felidae
    action Meow
        output parent:felidae:name + ": meow"
    end
end
```

To show the code in action we then use a main action that looks like this:

```
action Main
    cat sampleCat
    sampleCat:Purr()
    sampleCat:Meow()
    output sampleCat:Paws()
end
```

Where we instantiate a cat and call both the action from the superclass and the subclass giving the output of:

```
Sebastian: rhrhrhrhrhrhrhrhrhrhrhrh
Sebastian: meow
4
```

Worth noting is that we need to specify that the action Paws is public before we can call it from outside the class since it returns something (actions that does not return something are public by default). Likewise, if we in main where to write something like:

```
output sampleCat:parent:felidae:name
```

In order to access the name property, it would give an error since the name is not public.