QWEB: Language Reference Manual

Xabier Peralta, Tester
xp2159@columbia.edu

Ramisa Murshed, Language Guru
rm3508@barnard.edu

Kamrul Hossain, Systems Architect
kh2857@columbia.edu

Tamanna Hussain, Project Manager
th2704@barnard.edu

February 22, 2021

Contents

1 Introduction

QWEB is an object-oriented, pseudocode-style website language inspired by block-based visual programming languages such as Scratch and Snap, as well as interactive visualization languages such as Processing.js. The purpose of the language is to help both novel and experienced programmers design and develop websites to run in their browser.

Although popular web programming languages such as HTML and CSS are generally easy to learn and understand, QWEB fuses the two into one and produces a language that is more intuitive, maximizes human readability, and incorporates familiar programming constructs that are used in traditional high-level programming languages. By replacing repetitive aspects of HTML and CSS such as tags and selectors, QWEB makes use of control flow statements, variable declarations, and objects instead. In essence, QWEB is to HTML and CSS as Processing.js is to JavaScript.

2 Lexical Conventions

2.1 Comments

Comments are explanatory notes embedded within the code that are ignored by the compiler. QWEB has both single-line and multi-line comments. Single-line comments begin with a # symbol and multi-line comments begin and end with triple double quotes.

```
# This is a single-line comment

"""
This is a multi-line comment
"""
```

2.2 Identifiers

Valid identifiers, or names, consist of a sequence of alphanumeric characters or a single alphabetic character. Identifiers can contain an underscore, but they cannot be a QWEB keyword. They are also case sensitive; in other words, identifiers that differ in uppercase and lowercase letters are considered distinct.

```
SET x to 14 # x is a valid identifier because it is a single
    alphabetic character

SET 1a to 10 # 1a is an invalid identifier because it begins with a
    number
```

2.3 Operators

Relational operators are used to evaluate a boolean statement. QWEB uses the following reserved operators:

```
+ - < > <= >= != == * / = **
```

For example:

```
SET a to 1
SET b to 2

output a > b # false
output a < b # true
output a >= b # false

SET a to 2

output a <= b # true

SET A to [1,2,3]
SET B to [4,5,6]

output A == B # false
output A != B # true
```

2.4 Keywords

The list of identifiers reserved as keywords are below:

`and`	`or`	`not`
`IF`	`OTHERWISE IF`	`OTHERWISE`
`ENDIF`	`REPEAT`	`UNTIL`
`SET`	`FOR each`	`ENDFOR`
`function`	`length`	`object`
`true`	`false`	`output`
`display`	`pass`	`continue`
`int`	`str`	`bool`
`float`	`color`	`rect`
`circ`	`poly`	`tri`
`sqre`	`elps`	`point`
`output`	`object`	`line`

2.5 Indentation

Sequence control is indicated by writing one action after another on separate lines and with the same indent. For control flow statements, however, QWEB relies on brackets to indicate grouping. In other words, control flow keywords like `IF, OTHERWISE, OTHERWISE IF, REPEAT until`, and `FOR each` must be followed by a bracket with subsequent lines indented with a tab or four spaces and enclosed with a closing bracket.

```
# Example using indentation and brackets in a for each loop
FOR each key in data {
    FOR each val in key {
        table.append(val)
    }
}
ENDFOR
```

2.6 Literals

Literals represents strings or one of QWEB's primitive data types: int, char, float, and boolean.

2.6.1 String Literals

A string literal is a sequence of alphabetic characters enclosed in single or double quotation marks. Examples of string literals include:

```
"QWEB"
'Websites'
```

2.6.2 Int Literals

An integer literal is any sequence of integers between 0 and 9. Examples of integer literals include:

```
12
25
62
34
97
```

2.6.3 Char Literals

A character literal consists of a single letter from the alphabet. Examples of char literals include:

```
a
v
c
d
e
```

2.6.4 Boolean Literals

Boolean types are represented by the `True` and `False` keywords.

```
True
False
```

2.6.5 Float Literals

A float literal is a number with a whole number, optional decimal point, a fraction, and an exponent. Examples of float literals include:

```
10
11.5
5.25
6e+2
0.005
```

3 Data Types

3.1 Primitives

In QWEB, there are four primitive data types available: **`int`**, **`char`**, **`float`**, **`bool`**, and **`string`**.

3.1.1 `int`

`int` represents integers and consists of one or more characters in the range 0-9. Examples of declarations are:

```
int b
SET b to 23
SET int c to 45
```

3.1.2 `char`

`char` represents characters like symbols and letters (A-Z). In order to use `char`, the char element must be written in between single quotes and can be declared as the following:

```
char n
SET n to 'X'
SET char m to '$'
```

3.1.3 `float`

`float` represents floating point numbers and consists only of numbers that have a decimal point. Declaration is as follows:

```
SET float a to 10.5
float b
SET b to 50.0
```

3.1.4 `bool`

`bool` can only represent a boolean value of either true or false. An example of a declaration is:

```
SET bool a to false
bool b
SET b to true
```

3.1.5 `str`

`str` represents a sequence of characters stored in a character array. Declaration is as follows:

```
SET x to "language"
```

3.2 Types

QWEB also offers data types that are combinations of different data types and allows users to create and edit 2D structures. These data types are: `rect`,

circ, tri, sqre, ellipse, poly, point, line, and **size**.

3.2.1 `rect`

The `rect` datatype can be used to create rectangles. The syntax for this datatype consists of a comma-separated list of four integers representing the distance in CSS pixels from the sides of the image to the sides of the rectangle and the height and width. An example of this is:

```
    SET rect x to [20, 20, 150, 100]
```

The first two values of the parameters refers to the positioning of the rectangle within the page and the last two values refer to the length and width respectively.

3.2.2 `circ`

The `circ` datatype is used to create circles. The syntax for this datatype consists of a comma-separated list of three numbers representing the distance in CSS pixels from the edges of the image to the center of the circle and the radius must be given. An example of this syntax is:

```
     SET circ x to [50, 50, 40]
```

The first two values of the parameters refers to the positioning of the rectangle within the page and the last value represents the radius of the circle.

3.2.3 `tri`

The `tri` datatype is used to create triangles, which is essentially a plane connected by three points. As such, using tri datatype requires six arguments in which the first two refer to the x and y coordinates of the first point, the middle two refer to the x and y coordinates of the second point, and the final two refer to the last x and y coordinates of the third point. An example of this is:

```
    SET tri x to [50, 40, 100, 78, 54, 58]
```

3.2.4 `sqre`

The `sqre` datatype can be used to create squares. In order to create squares, a comma-separated list of three numbers representing the positioning of the upper left corner of the square and the length of each side must be used. An example of this is:

```
    SET sqre x to [20,20,16]
```

The first two values refer to the x and y coordinates of the upper left corner of the square and the last value represents the length of each side of the square.

3.2.5 `elps`

The `elps` dataype can be used to create an ellipse, which are circles with unequal width and height. In order to create ellipses, a comma-separated list of four numbers that represent the location, width, and height of the ellipse must be used. An example of this is:

```
SET elps x to [50, 50, 16, 19]
```

The first two values refers to the x and y position of the center of the ellipse and the last two refer to the width and height respectively.

3.2.6 `poly`

The `poly` datatype can be used to create polygons, which are shapes consisting of at least three straight sides that are not already featured in the QWEB library. The `poly` datatype consists of a comma-separated list of at least six integers that represent coordinates and points of the polygon. An example of this is:

```
SET poly x to [200, 10, 250, 190, 160, 210]
```

Each pair of arguments refers to the x and y coordinate of a point that connect to the other points given.

3.2.7 `point`

The `point` data type can be used to create a single pixel that pertains to a coordinate in space. To use the `point` data type, a comma-separated list of two integers is needed to represent the x and y coordinates of the point. An example of this is:

```
SET point x to [5, 6]
```

3.2.8 `line`

The `line` data type can be used to draw a line, which is a direct path between any two points. In order to construct a line, a list of four integers that represent the two endpoints of a line must be provided. An example of this is:

```
SET line x to [7,8,12,15]
```

The first two arguments refer to the x and y coordinates of the first endpoint and the last two refer to the x and y coordinates of the second endpoint.

3.3 Collections

In QWEB, there are two types of collections: `list` and `dict`.

3.3.1 `list`

A list is represented by a sequence of comma-separated elements enclosed in two square brackets, [ ]. The index of a list starts at zero and the position of each element can be accessed using their position along the list. Lists can only contain one data type of either primitives and objects. If a list consists of one of the primitive data types, all of the elements that are added must be of the same type (i.e., a list of only integers or a list of only floats). The methods available for a list in QWEB are `append`, `remove`, and `length`. An example of lists being used is shown below:

```
SET list to ["Apples", "Bananas", "Pears", "Oranges", "Grapes"]
```

In this example, a list is created to hold five string elements.

3.3.2 `dict`

Dict or dictionaries are represented by a series of element pairs that include different data types. This collection type can be used to store data in key value-pairs. An example of this is shown below:

```
SET data to:
    "Firstname": ["Jill", "Eve"],
    "Lastname": ["Smith", "Jackson"],
    "Age": [50, 94]
```

In this example, a dict called data is being created to hold some basic information about two different people.

3.4 Objects

In QWEB, objects acts as as container for different or similar data types and can be used to create more intricate structures.

4 Object-Oriented Programming

QWEB supports objected-oriented programming without inheritance and allows for multiple data types and functions to be nested within a single class. Classes have to be defined within the global scope. An example of creating an object is shown below:

```
object Orange{
    function constructor(length, width){
        SET this.length to length
        SET this.width to width
    }

    function area(){
        output this.length * this.width
    }
}
```

This is an example of how to instantiate an object in a program:

```
# prints the area of an Orange object
SET object Orange(12,4) to clementine
SET areaOfClementine to clementine.area()
display areaOfClementine
```

5 Statements and Expressions

5.1 Statements

QWEB supports the use of statements in order to iterate or perform repetitive actions. All statements are executed sequentially.

5.1.1 `if-else`

QWEB supports the use of conditional statements, including Pythonic `if-else` statements. In QWEB, *if* is denoted by an all-caps `IF`, *elif* is denoted by `OTHERWISE IF`, and *else* is denoted by `OTHERWISE`. Each statement is separated by curly braces to denote the end of the statement. If an `IF` statement evaluates to the boolean false, then `OTHERWISE IF` is evaluated. If this statement also evaluates to false, then `OTHERWISE` is executed.

```
IF (x == 5) {
        output "x is 5"
} OTHERWISE IF (x == 3) {
        output "x is 3"
} OTHERWISE {
        output "x is not 5 or 3"
}
ENDIF
```

5.1.2 `for`

The usage of `for` loops is also supported by QWEB. They serve a similar function as `for` loops in Python and are denoted by the keywords `FOR each`. These loops iterate through elements in a structure such as a list and allow for the repeated execution of a block of code, for which beginning and ending is indicated by curly braces. Iteration begins at 0.

```
FOR each element in list {
    output element
}
ENDFOR
```

5.1.3 `while`

QWEB also supports the use of Pythonic `while` loops, which are indicated in QWEB by the keywords `REPEAT until`. These loops repeatedly execute a block of code (contained in the curly braces) until the boolean statement that follows `REPEAT until` no longer evaluates to true.

```
SET int x to 0

REPEAT until (x > 6) {
    output "Hello World"
    SET x to x + 1
}
```

5.1.4 `output`

QWEB denotes `return` statements with the keyword `output`. This can be used in any function in QWEB and is not constrained to any particular type. Whenever an `output` statement is called, the program exits out of the current loop and returns the value denoted after the keyword `output`.

```
def incrementX (int x) {
    SET result to x + 1
    output result
}
```

5.2 Expressions and Operators

5.2.1 Assignment Operator

The assignment operator in QWEB is denoted by the keyword `SET...to`, which assigns a value to a variable. The value directly after the keyword `to` is assigned to the variable indicated between the keywords `SET` and `to`.

```
SET A to [1, 2, 3, 4]

SET x to "Hello World"
```

5.2.2 Arithmetic Operators

QWEB supports the use of the arithmetic operators + (to add or concatenate values), - (to subtract values), * (to multiply values), / (to divide values), and % (to perform modulo operations).

5.2.3 Logical Operators

QWEB also makes use of the logical operators `and`, `or`, and `not`, providing an identical function to that of its usage in Python.

6 Standard Library

6.1 List Functions

6.1.1 `append`

`append` is a static function that adds an item to the end of a list. It takes in a value or object compatible with the type of elements contained in the list it is appending to and does not return anything.

```
SET list to [1, 2, 3, 4]

list.append(5)
display list # [1, 2, 3, 4, 5]
```

6.1.2 `remove`

`remove` is a static function that removes an item from a list, given the value or the index of the item in the list. It takes in a value or object that exists in the list it is appending to and does not return anything.

```
SET list to [1, 2, 3, 4]

list.remove(3)
display list # [1, 2, 4]

list.remove(list[1])
display list # [1, 4]
```

6.1.3 `length`

`length` is a function that computes the length of a given list. It returns a value of type **int**.

```
SET list to [1, 2, 3, 4]

SET len to list.length()
display len # 4
```

6.2 HTML Functions

6.2.1 `createHeader`

`createHeader` is a function that returns an HTML header. It takes in two values: the first being a string that contains the text to be put in the header, and the second being an optional parameter of type `int` from 1 to 6, inclusive, that specifies which size the header should be. If the second parameter is omitted, then it defaults to `<h1>`.

```
SET headerText to "This is a header."

SET head to createHeader(headerText, 2)
display head #prints "This is a header." header in size h2
```

6.2.2 `createParagraph`

`createParagraph` is a function that returns an HTML paragraph. It takes in a string that contains the text to be put in the paragraph.

```
SET paragraphText to "This is a paragraph."

SET par to createParagraph(paragraphText)
display head #prints "This is a paragraph." paragraph
```

6.2.3 `createTable`

`createTable` is a function that returns an HTML table. It takes in a 2D array containing each value to be put into each cell of the table.

```
SET tableArrays to [[1, 2, 3, 4], [a, b, c, d]]

SET tab to createTable(tableArrays)
display tab # produces a 2D array in table form
```

6.2.4 `createUnorderedList`

`createUnorderedList` is a function that returns an HTML unordered list. It takes in a list that contains all of the values to be put into the unordered list.

```
SET ulList to [a, b, c, d]

SET ul to createUnorderedList(ulList)
display ul # prints out an unordered list of ulList elements
```

6.2.5 `createOrderedList`

`createOrderedList` is a function that returns an HTML ordered list. It takes in a list that contains all of the values to be put into the ordered list.

```
SET olList to [x, y, z]

SET ol to createOrderedList(olList)
display ol # prints out an ordered list of olList elements
```

7 Sample Program

The following program illustrates an example of how to use Object-Oriented programming in QWEB:

```
# define a ShoppingList object
object ShoppingList {
   # the constructor is defined with arguments
   function constructor(str item, int quantity, float price) {
      SET self.groceries to {}
      SET this.item to item
      SET this.quantity to quantity
      SET this.price to price
```

```
    }

    # functionality
    function addToList(str itm, int quantity, float price) {
        SET totalPrice to price * quantity
        SET groceries[itm] to totalPrice
    }

    function removeFromList(str itm) {
        FOR each fruit in groceries {
            IF itm == fruit {
                groceries.remove(itm)
            }
            ENDIF
        }
        ENDFOR
    }

    function display() {
        # uses the built-in createTable function to return the
            structure of the table
        SET table to createTable(len(groceries), 2)

        # uses a nested for loop to add the data to the cells in the
            table
        FOR each fruit in groceries {
            FOR each price in fruit {
                table.append(price)
            }
            ENDFOR
        }
        ENDFOR

        # returns the table to display on webpage
        output table
    }
}

# declare a ShoppingList object called myShoppingList
SET object ShoppingList(Oranges, 13, 1.33) to myShoppingList
myShoppingList.addToList(Apples, 5, 1.32)
myShoppingList.addToList(Tomatoes, 3, 1.89)

# print myShoppingList object
display myShoppingList
```

In contrast, the following program in HTML would require users to manually populate and remove elements from the table:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Shopping List</title>
</head>
<body>
    <table border = "1">
    <tr>
        <th>Item</th>
        <th>Price</th>
    </tr>
    <tr>
        <td>Oranges</td>
        <td>23.94</td>
    </tr>
    <tr>
        <td>Apples</td>
        <td>6.60</td>
    </tr>
    <tr>
        <td>Tomatoes</td>
        <td>5.67</td>
    </tr>
</table>
</body>
</html>
```