DRRTY Programming Language Reference Manual

Dylan Bamirny, Richard Lopez, Rania Alshafie, Trinity Sazo

db3381, rl3020, rta2114, ts3185

Contents

1 Introduction

The DRRTY programming language is an imperative and declarative Python-like scripting language used to build modular HTML components. The language aims to integrate HTML and backend logic in one to create simple web pages with basic HTML components supplied by the language. DRRTY enables programmers to build more complex HTML modules similar to the functionality of the React Javascript library. The syntax follows Python syntax closely with the addition of a few operators to distinguish between HTML code and functional code. With HTML being a generally easy language to learn and understand, the language maximizes customization of HTML components in a more functional manner.

2 Syntax

2.1 Comments

Comments are notes put in by the user that is ignored by the compiler. We signify them to be any text placed between a /* and */

```
1    /* This is a comment */
2    /* They can also extend to
3    be multiple lines */
```

2.2 Identifiers

2.2.1 Variable Declarations

There are specific ways to declare a valid variable name to store information in DRRTY. Variable names can begin with a lowercase letter, followed by a series of alphabetic or numeric characters. They cannot begin with a number or capital letters, but can include underscores. They cannot include special characters and cannot be a DRRTY reserved keyword.

```
1    x = 4;
2    r1 = "Hello" ;
3    listOfNames= ['Dylan', 'Richard', 'Rania', 'Trinity'] ;
```

2.2.2 Function Declarations

Function declarations are signified by the keyword "def" , followed by the function name and any parameters that the function might take placed in parentheses. The beginning and end of a function's code to execute is bound by two curly braces.

```
def showHeader(head){
  render(head);
}
```

2.3 Keywords

A list of reserved keywords in DRRTY:

```
AND       ELSEIF    RETURN      HTML
OR        DEF       INT         LIST
IF        FOR       STRING      DICT
ELSE      RENDER    BOOL        LENGTH
```

2.4 Braces

2.4.1 Curly Braces

Double curly braces will be used to pass in variable values or code from our DRT code into the HTML snippets of code.

```
    myName =   DRRTY
    return( <> <h1> {{myName}} </h1> </> )
```

2.4.2 Kites

A special character used to signify the beginning and end of HTML snippet within our DRRTY code.

```
    <> <header> Hello World! </header> </>
```

2.5 Semicolons

Semicolons are used for sequencing and separating expressions, and signifying the end of a line in a multi-line program

```
x = 1;
y = 2 ;
x + y = 2;
```

3 Types

3.1 int

int represents an integer value that contains a sequence of one or more digits from 0-9. An example of a declaration is as follows:

```
a = 1;
b = 2;
c = 3;
```

3.2 bool

The boolean data type can be one of two values: true or false. When evaluating a boolean expression, the result will also be a boolean.

```
finished = true ;
```

3.3 str

A string is a sequence of alphanumeric characters enclosed in either single quotes or double quotes.

```
helloWorld = "Hello there!"
```

3.4 HTML

A sequence of valid HTML tags, elements, and other HTML content that is enclosed in kites, our special character used to indicate HTML type.

```
myHeader = <> <header> This is my webpage! </header> </>
```

3.5 list

List implementation in DRRTY is mutable, ordered, and dynamic. Lists allow users to store data of the same type within an iterable container. Lists can be initialized in a few ways, and have different ways and functions to access its elements. While and for loops are often used to iterate through them. When indexing these lists, a KeyError will occur if a key is not present in the list and a user attempts to retrieve it using an index out of bound. The start and end of a list are bound by square brackets [], and elements are separated by commas

```
/* The first way to initialize a list */
myList = []
```

```
3
4       /* The second way to initialize a list */
5       myList = [1,2,3,4,5]
6
7       /* Add an element to the list by indexing*/
8       myList[5] = 6
9
10      /* Access elements by indexing */
11      return myList[3] /* This would return the value of 4 */
```

3.6 dict

Dictionaries are used in Drrty to organize data into key-value pairs. Every key inserted into the dictionary must be unique and must map to a specified value, which does not have to be unique. It is important to note that Drrty dictionaries are not ordered. Dictionaries can be initialized in two ways, and have different ways of adding and accessing elements in the dictionary.A KeyError will occur if a key is not present in the dictionary and a user attempts to retrieve a value using the nonexistent key.

```
1  /* The first way to initialize a dictionary*/
2      myDict = {}
3
4  /* The second way */
5      myDict = { 1 : "one", 2: "two", 3: "three"}
6
7  /* Add an element (key and value) to a dictionary */
8      myDict["greeting"] = "Hello there!"
9
10 /*Retrieve an element from a dictionary */
11     output = myDict["greeting"]
```

4 Operators

4.1 Assignment

The assignment operator, signified by an equal sign (=), is used to store information or different values of any type into a variable. Using the variable naming conventions outlined end in the aforementioned "Identifiers" section (2.2), the assignment operator is used to set any variable name equal to a value of any data type.

```
1       x = 5;
2       greeting = "Hello!";
3       sum = 10 + 40 ;
4       list1 = ["My", "Name", "Is"] ;
```

4.2 Arithmetic

The arithmetic operators used remain consistent with the standard use of arithmetic operators used in any other constant. We implement the 4 main operators: addition, subtraction, multiplication, and division.

```
x = 10; y= 5;
z = x + y /* 15 */
z = x - y /* 10 */
z = x * y /* 50 */
z = x / y /* 2 */
```

4.3 Logic

Logic operators are implemented as comparative checks between different values of variables, and return booleans of whether these checks are true or false. The ones implemented are greater than, less than, greater than/equal to, less than/equal to, eqaul, and not equal. Below are examples of all of them in use.

```
 x = 10; y= 5;
z = x > y /* true */
z = x < y /* false */
z = x >= y /* true */
z = x <= y /* false */
z = x == y /* false */
z = x != y /* true */
```

5 Control Flow

5.1 if/elseif/else

Conditional statements in DRRTY are implemented through if/elseif/else checks. If the first conditional statement is evaluated to a boolean true, then the code that follows is completed. If false, the "elseif" block is completed. If all are false, the "else" block is executed. Each block of code corresponding to the conditional statements are bound by curly braces.

```
x=0
if myList.length() > 0{
    x= x+1;
}elseif myList.length() == 0 {
    x = x-1;
} else {
    return x }
```

5.2 for

For loops are used to iterate through a data structure, such as a list, and execute a block of code on each individual element of that structure. The start and end of a for loop is signified by curly braces.

```
    myList = [1,2,3];
    for item in myList{
        item= item + 2;
    }
```

5.3 while

While loops are used to execute a block of code until a a boolean statement that is defined in the parentheses following the keyword "while" no longer evaluates to true. The beginning and end of the while loop code will be signified by single curly braces.

```
    counter = 1;
    myList = [];
    while (counter < 10){
        myList.add(counter) ;
        counter++ ;
    }
```

6 Output

6.1 Return

Return is a keyword used to send output from the function to the main program, and signifies the complete execution of that function. It is analogous to the return keyword in Python and Java

```
 def double(num){
    newnum = num * 2 ;
    return newnum
 }
```

6.2 Render

Render is a DRRTY specific output function, that particularly takes HTML content and sends that code to the HTML client file. The content sent by the render function is HTML type.

```
 def fav():
```

```
    return(<>
        {makeHeader("Our favorite Colors!", 1)}
      </>)
render fav()
```

7 Standard Library

7.1 list functions

7.1.1 add

add is a static function that appends an item to the end of a list. It only accepts data types that match the type of elements contained in the list to which it is appending. It does not return anything.

```
    myList = ["1","2","3"];
    myList.add("4");
    /* would return ["1","2","3","4"] */
```

7.1.2 remove

remove is a static function that accepts a value or index and deletes the corresponding item from a list given its index or value. It only accepts items that exist in the list. It does not return anything.

```
    myList = ["1","2","3","4"];
    myList.remove("3");
    /* would return ["1", "2", "4"] */
```

7.2 length

length is a static function that returns the number of elements in a given list. The returned number is an **int**.

```
    myList = ["1", "2", "4"];
    myList.length() ;
    /* would return 3 */
```

7.3 dict functions

7.3.1 get

get is a static function that accepts a key and returns its corresponding value.A KeyError will occur if a key is not present in the dictionary and a user attempts to retrieve a value using the nonexistent key.

```
myDict["greeting"] = "Hello there!"
output = myDict.get("greeting")
/* Would return "Hello there!" */
```

7.3.2 pop

pop is a static function that accepts a key, removes it from the dictionary, and returns its corresponding value. If the key is not found, it will throw a KeyError.

```
myDict = { 1 : "one", 2: "two", 3: "three"}
deletedValue = myDict.pop(3) /* Would return "three" */
```

7.3.3 keys

keys is a static function that takes no parameters, and returns a list of all the keys in the dictionary

```
myDict = { 1 : "one", 2: "two", 3: "three"}
myKeys = myDict.keys() /* would return [1, 2, 3] */
```

7.3.4 values

values is a static function that takes no parameters, and returns a list of all of the values in the dictionary.

```
myDict = { 1 : "one", 2: "two", 3: "three"}
myValues = myDict.values() /* would return ["one", "two", "three"] */
```

7.3.5 length

length is a static function that returns the number of key/value pairs, and returns an **int**.

```
myDict = { 1 : "one", 2: "two", 3: "three"}
myDict.length()
/* would return 3 */
```

7.4 HTML functions

We are implementing built in HTML functions that will be able to create different HTML elements and set their values upon function call. Below are the different built in functions.

7.4.1 makeHeader

`makeHeader` is a function to create an HTML header. The function takes two parameters, a string and an integer. The string is text to be displayed within the header. The second input represents size of the header. This parameter is optional and of type `int`. The default value for the size is 1.

```
    headerText = "DRRTY";
    header = makeHeader(headerText, 1);
    return(header); /* returns "DRRTY" header in size h1 */
```

7.4.2 makeText

`makeText` is a function that returns an HTML body of text. The string is text to be displayed within the header. The second input represents size of the header. This parameter is optional and of type `int`. The default value for the size is 1.

```
    headerText = "DRRTY";
    text = makeText(headerText, 1);
    return(text); /* returns "DRRTY" text in size h1 */
```

7.4.3 makeList

`makeList` is a function that returns an HTML list. The function takes two parameters, a list parameter and optional boolean parameter. Elements of the list will be used, by default, to create an unordered HTML list. The second input is optional: if set to true, it will return an ordered HTML list instead.

```
    snacks = ["Chips", "Cookies", "Gummy Bears"];
    snackList = makeList(snacks);
    return(snackList); /* returns unordered HTML list */

    topDrinks = ["Matcha Latte", "Hot Chocolate", "Chai"];
    drinksRanked = makeList(topDrinks, True);
    return(drinksRanked); /* returns ordered HTML list */
```

7.4.4 makeTable

`makeTable` is a function that returns an HTML table. The function takes a 2D list. Each nested list represents a full column of the table, and elements of that nested listed are rows of the given column. Together, the columns form a complete table.

```
    names = ["Dylan", "Richard", "Rania", "Trinity"];
    colors = ["Purple", "Blue", "Orange", "Green"];

```

```
    favColors = [names, colors];
    colorTable = makeList(favColors);
    return(colorTable); /* returns HTML table */
```

8 Sample Program

```
fav_colors = {"Dylan" : "Purple",
              "Richard" : "Sky Blue",
              "Trinity" : "Green",
              "Rania" : "Orange"};

def header(title){
   return(<><h1>{{title}}</h1></>);
   }

def color_table(myDict){
       row = <></>
       for name, color in myDict{
       row += <>
                    <tr>
                        <td>{name}</td>
                        <td style="background-color: {color};"></td>
                    </tr>
                   </> ;
        }
    return(<> <table>
              <tr>
            <th>Name</th>
            <th>Favorite Color</th>
             </tr>
             {row}
            </table> </>);
}

def all(){
    return(<>
         {header("Our favorite Colors!")}
         {color_table(fav_colors)}
       </>);
}

render all();
```