Reptile (.rt)
Language Reference Manual

Aileen Cano (ac4440), Aviva Weinbaum (aw3156), Lindsey Weiskopf (ltw2115), and Hariti Patel (hvp2105)

February 2021

Introduction

Reptile is a programming language that is intended to support libraries that streamline the process of creating simply-coded graphics. As more children are learning computer science at a younger age, there is a demand for simple programming languages that teach computer science principles in a digestible and visual manner. Languages typically labeled for beginners like Scratch and Swift Playgrounds teach kids to code by showing immediate visual results from code – whether that is a simple square or a complex environment built upon existing code blocks. Further, libraries like Turtle graphics add novelty to simple image-building operations by showing a turtle drawing the desired shape. The goal of Reptile is to build upon the success of these "beginner" programming languages to provide immediate gratification to the coders through graphics.

1. Lexical Conventions

1.1 Comments

The characters /\ introduce a comment for a single line. For multi-line comments, each line must begin with /\.

1.2 Identifiers (Names)

An identifier is a string of characters, both letters and digits, that can be used to name a variable, object, or function. The first character of the identifier must be alphabetic. Upper and lower case characters may be used along with underscores and digits to create the identifier.

1.3 Keywords

The following identifiers are reserved to be used as keywords, and cannot be used otherwise. The keywords are case sensitive.

Keywords	**Description**
if	Enters block if condition statement is satisfied
else	Enters block when if condition statement is not satisfied
for	Continues to repeat up until specified condition is not satisfied
while	Continues to repeat up until specified condition is not satisfied
true	Boolean literal for 1
false	Boolean literal for 0

function	Specifies that the following block is a function
return	Returns a value from a function
void	Used for functions when nothing is to be returned

1.4 Type Specifiers

The following are type specifiers with their descriptions and functions, if applicable. These type specifiers are case sensitive.

Type Specifiers	**Description**	**Functions**
int	Any signed integer	
float	Any floating point decimal	
boolean	Boolean value of “true” or “false”	
string	Standard string	**myString.length()**
RGB	List of 3 color values ranging from 0 to 255	
List	Standard array	**myList.length : length of myList**
Canvas	Two-dimensional array of pixels which Pointers act on	**myCanvas.x** : length of myCanvas **myCanvas.y** : height of myCanvas **myCanvas.close()** : stop editing canvas and generate SVG. There will be as many SVGs produced as there are .close() calls.
Pointer	Pen used to draw pixels and move around on Canvas	**myPointer.x** : x coordinate **myPointer.y** : y coordinate **myPointer.color(RGB rgb)** : set color for future markings **myPointer.pixel(int x, int y)** : mark pixel with specified color; if no color is specified, default is black [RGB (0, 0, 0)]; if no x and y are specified, mark pixel at current location of myPointer **myPointer.point(int d)** : point pen in direction d and when instructed to draw, follow the angle d specified
File	Encapsulated file pointer for outputting generated svg file	

1.5 Punctuators

A punctuator is a symbol which does not specify any specific operation to be executed, but instead, it has syntactic value to the compiler to format and parse the code.

Symbols	**Description**
;	Statement terminator
,	Separation of variables
{ }	Block of statements
( )	Condition, function declaration, and parameter specifier
[]	Used in list instantiation

1.6 Operators

1.6.1 Arithmetic/Logical Operators

Arithmetic/Logical Operators	**Description**
=	Assignment operator
+	Addition operator
-	Subtraction operator
*	Multiplication operator
/	Division operator
^	Exponentiation operator
%	Modulo operator
==	Returns true if values are equal, false otherwise
!=	Returns true if values are not equal, false otherwise
++	Increment the value of variable on the left by 1
--	Decrement the value of variable on the left by 1
<	Less than operator
>	Greater than operator
<=	Less than or equal to operator
>=	Greater than or equal to operator
&&	Logical AND operator

||	Logical OR operator
!	Logical NOT operator

1.6.2 Operator Precedence

The following operators are presented in the order of their precedence from highest to lowest.

Operator Symbols	**Description**
!	Logical NOT operator
^	Exponentiation operator
++ --	Increment operator, decrement operator
* / %	Multiplication operator, division operator, modulo operator
+ -	Addition operator, subtraction operator
< > <= >=	Less than operator, greater than operator, less than or equal to operator, greater than or equal to operator
== !=	Equality operator, inequality operator
&& ||	Logical AND operator, logical OR operator
=	Assignment operator

2. Syntax

2.1 Variable Declaration

Variables are defined using an identifier and an expression. Variable identifiers are strings for the name of the variable which are used to later manipulate the variable. Identifiers are any uppercase or lowercase character followed by any number of alphanumeric and underscore characters. If the variable identifier has not already been declared, it must have its type name before.

```
int a = 3;
```

2.2 Statements

2.2.1 Termination

Statements are sequenced and terminated using the semicolon. Each line must end with a semicolon in order for the next line to be taken as a separate statement, with the exception of control flow statements and loops, which utilize curly braces.

2.2.2 Control Flow

Conditional statements are if and else. Control flow is initialized using the "if" keyword, a boolean expression in parenthesis, and a code block enclosed in curly braces.

```
int foo = 3;
if (foo < 10) {
      /\ block of code
} else {
      /\ block of code
}
```

2.2.3 Loops

Reptile supports for and while loops.

2.2.3.1 For Loops

For loops require an initializer variable, a boolean conditional expression, and a variable expression, followed by a code block enclosed in curly braces to be executed during each iteration of the loop until the boolean expression is not true. An example of the syntax is as follows:

```
for (int i = 0; i < 5; i++) {
      /\ block of code
}
```

2.2.3.2 While Loops

While loops consist of the "while" keyword followed by a boolean expression in parenthesis that is evaluated each time the loop is executed. When the expression is false, the loop will stop executing. An example of the syntax is as follows:

```
int foo = 3; int bar = 4;
while (foo < 10) {
      /\ block of code
}
```

2.3 Functions

2.3.1 Function Declarations

Functions are declared using the “function” keyword, a return type, an identifier, and any number of arguments enclosed in parenthesis. The function body contains a block of code, and must return a value of the type specified using the keyword “return”. An example of the syntax is as follows:

```
function int foo(int a, int b) {
      /\ block of code
      return 3;
}
```

2.3.2 Function Calls

Functions are called with the function identifier and the number of arguments required. Because a value is returned, the outcome of a function call can be saved in a variable. An example of the syntax is as follows:

```
int bar = foo(5,10);
```

2.4 Arrays

Arrays are fixed-size, ordered data structures that hold a single data type.

2.4.1 Declaring

Arrays are instantiated with their type and a length. The array length function can be used to access the length of an array at any time.

```
int array[3] = [3,4,5];
int len = array.length;       /\ this variable has value 3
```

2.4.2 Accessing

Array elements can be accessed with their index using brackets and the array identifier. If the specified index is out of bounds of the array, an error will be thrown.

```
int second = array[1];        /\ this variable has value 4
```

3. Standard Library Functions

3.1 Canvas Manipulation

3.1.1 Canvas instantiation

The canvas is the drawing slate for all external libraries and pixel manipulation in Reptile. Canvas are initialized like any other object in the language, with the pixel parameters as the width and height of the canvas. The dimensions of the canvas are immutable after instantiation.

```
Canvas canvas = new Canvas(80, 100);
```

3.1.2 Canvas instance variables

The canvas object holds the length and width of the canvas as instance variables `x` and `y`.

3.1.3 Canvas terminate

The canvas must be closed after use through the following function. Closing the canvas will terminate the writing to file process.

```
canvas.close();
```

3.2 Pointer Manipulation

The pointer is critical to the instantiation of any drawing object created from Reptile. The pointer can “flip” or change the color of one pixel at a time.

3.2.1 Pointer instantiation

The pointer can be instantiated with the following code where canvas is the canvas object and the integer parameter represent the placement of the pointer starting from the top left corner of the canvas:

```
Pointer p = new Pointer(canvas, 0, 0);
```

3.2.2 Pointer instance variables

The pointer object carries the following instance variables.

Name	**Type**	**Meaning**
x	int	number of pixels from left of canvas
y	int	number of pixels from top of canvas
color	RGB	color of pixel pointer as an RGB value
angle	int	angle from first quadrant in degrees

3.2.3 Pointer functions

Function	**Return Type**	**Purpose**
p.pixel(int x, int y) or p.pixel()	none	“flips” the pixel at the current (x,y) or moves to

		parameter (x,y) and “flips”
p.color(RGB rgb)	none	changes the default color of the pointer
p.point(int a)	none	adds parameter a (angle change) to angle variable

3.3 File I/O

The user must specify to have their drawing written to an svg file. The canvas object must be passed into the file when opening. When the canvas is closed, the file is automatically closed and saved to the user. A file can be opened with the following function:

```
File f = new File(“image.svg”, canvas);
```

4. Semantics

4.1 Scope

4.1.1 Blocks

In Reptile, a block is defined by any segment of code defined within a set of curly braces. A block could be a class, a function, or segment of a control sequence. Curly braces can also be arbitrarily placed to define a block within a program.

4.1.2 Blocks and Scope

The scope of a variable is always available to and limited by its innermost curly braces. The only exception to this rule is instance variables defined in the standard library or belonging to objects created by the user.

4.2 Recursion

Recursion is extremely useful in graphics, especially when there are repeated patterns being drawn. Recursion can be used by calling a function within the function itself. See the gcd algorithm for an example.

5. Sample Code

gcd.rt
```
int gcd(int a, int b) {
      if (b==a) {
            return a;
      }
      else {
            return gcd(b, a%b);
```

```
     }
}
```

main.rt

```
/\ make a canvas
Canvas canvas = new Canvas(100,200);
Pointer ptr = new Pointer(canvas,0,0);
RGB blue = new RGB(0,0,255);

/\ set color for drawing
ptr.color(blue);

/\ point to the right and start drawing
ptr.point(90);
for (int i = 0; i < 50; i++) {
     ptr.pixel(ptr.x, ptr.y);
     ptr.x ++;
}
ptr.point(180);
for (int i = 0; i < 70; i++) {
     ptr.pixel();
     ptr.y ++;
}
for (int i = 0; i < 50; i++) {
     ptr.pixel();
     ptr.x --;
}
ptr.point(270);
for (int i = 0; i < 70; i++) {
     ptr.pixel();
     ptr.y --;
}

/\ diagonal line:
ptr.x = 50;
ptr.y = 0;
ptr.color(new RGB(255,0,0));
ptr.point(135);

for (int i = 0; i < canvas.x; i++) {
     ptr.pixel();
     ptr.x ++;
     ptr.y ++;
}
```

main.rt

```
/\ This program accomplishes the same thing
Canvas canvas = new Canvas(100,200);
Tortoise tortoise = new Tortoise(canvas,0,0);
tortoise.draw(50,90,new RGB(0,0,255));
tortoise.draw(70,180,new RGB(0,0,255));
tortoise.draw(50,270,new RGB(0,0,255));
tortoise.draw(70,0,new RGB(0,0,255));

tortoise.set(50,0);
tortoise.draw(50,135,new RGB(255,0,0));
```

Output: