CaRdY

Language Reference Manual

Pazit Schrecker ⊙ Liseidy Bueno ⊙ Lindsey Wales
Kenya Plenty ⊙ Katrina Zhao

Contents

Introduction

CaRdY is an innovative new programming language designed to make the implementation of text-based card games as seamless and easy as possible. Because CaRdY aims to take all the hassle out of coding your own card game, the language provides built-in algorithms to control the flow of the game (ie shuffling, drawing/dealing, displaying round results, switching turns, etc.) and a pre-built and customizable 52 card Deck class. It is important to note that CaRdY is meant to be used in coding one player (human vs. computer) card games, but the language is not intended to facilitate the creation of multiplayer card games that can be played across consoles or hosted on a server of some kind.

Types

Primitive Data Types

int	4 byte signed integer
char	1 byte ASCII character
bool	1 byte, 0 or 1
float	4 byte floating point number

Non-Primitives

str	Sequence of ASCII characters enclosed in quotation marks

Collections

List	Stores multiple objects of the same type in a single variable. Lists can be initialized empty or with variables. The size is not fixed.
Set	Stores an unordered, unindexed collection of multiple objects in a single variable. Sets can be initialized empty or with variables. The size is not fixed.
Dictionary	Stores multiple key/value pairs in a single variable. Dictionaries can be initialized empty or with variables. The size is not fixed.

Operators

Assignment Operators

=	Assigns a value to a variable (x=5)
+	Addition
-	Subtraction

*	Multiplication
/	Division
//	Truncated division
%	Modulus

Comparison Operators

All comparison operators evaluate to bool

==	Equals (x==5)
!=	Does not equal (x!=5)
>	Greater than (x>y)
<	Less than (x<y)
>=	Greater than or equal to (x>=y)
<=	Less than or equal to (x<=5)

Logical Operators

All logical operators evaluate to bool

and	True if the statements on either side are both true.
or	True if at least one of the statements on either side is true.
not	False if the expression being evaluated is true.

Identity Operators

All identify operators evaluate to bool

is	True if the variables on either side are the same object (in the same location in memory)

is not	True if the variables on either side are not the same object (not in the same location in memory)

Membership Operators

All membership operators evaluate to bool

in	True if the value on the LHS appears in the sequence on the RHS
not in	True if the value on the LHS does not appear in the sequence on the RHS.

Statements

Expressions

An Expression is a line of code composed of one or more of the following:

- A built-in data type (See the built-in types section of Chapter)
- An identifier of a variable (See the identifiers section of Chapter)
- The associated key-value of a dictionary
- A list element
- A set element
- An operand followed by an operator followed by another operand
- The **not** operator followed by a **bool** value

Expression Examples

4 + 8; /* This expression evaluates to the int value 12 */

“hello” + “ World”; /* This expression evaluates to the str “hello World” */

not true; /* This expression evaluates to the bool value false */

Return Statements

A return statement exits from the current function. It may either be followed by an expression or exist on its own. When there is no expression following the return statement it simply exits the function:

return; /* function exits */

When there is an expression following the return statement, the function call returns that value:

return 9 + 10; /* function exists and returns 19 */

Declaration and Initialization

In order to use an identifier in your code it must first be declared. The declaration must include the type of the value associated with the identifier. Before use in the code, the value of the identifier must also be initialized, or set. This value must match the declared type of the identifier. For integer, string, float, boolean, and character types, the Initialization must either happen at the same time as declaration.

This is an example of an integer identifier **myInt** being declared and initialized in one line (valid):

int myInt = 3;

This is an example of an integer identifier **myOtherInt** being declared and initialized in one line (invalid):

int myInt = 3.14;

This case is invalid because an int type cannot be set to a value of type **float**.

This is an example of an integer identifier **myString** being declared and initialized in one line:

str myString = "I am a String";

The syntax to declare a dictionary is the following:

myDict = { ... };

Where the brackets can contain a list of key/value pairs separated by a comma, or it can be empty to initialize an empty dictionary. Key value pairs a denoted by the key followed by a colon and then the value.

This is an example of a dictionary **myDict** with integer keys and String values being declared and initialized:

myDict = {1: "spades", 2: "diamonds"};

The syntax to initialize a list is the following:

type myList = [...];

Where the brackets can contain a list of literals separated by a comma, or it can be empty to initialize an empty list.

This is an example of a list **myList** with of character types:

char myList = [‘a’, ‘b’, ‘c’];

The syntax to declare a set is the following:

type mySet = {...};

Where the brackets can contain a list of literals separated by a comma, or it can be empty to initialize an empty set.

This is an example of a set **mySet** with of character types:

char mySet = {‘a’, ‘b’, ‘c’};

Control Flow

Conditional Statements

There are two valid forms of the conditional statement

if (expression1) { statement$_1$ }

if (expression$_1$) { statement$_1$ }

else if (expression$_2$) { statement$_2$ }

.

.

.

else if (expression$_{n-1}$) { statement$_{n-1}$ }

else { statement$_n$ }

In both forms, **expression**$_1$ is evaluated first. Each **expression**$_x$ must evaluate to a bool. If the bool value of **expression**$_1$ is 1, **statement**$_1$. If **expression**$_1$ evaluates to 0 in the second case, **expression**$_2$ is evaluated. If this is 1, **statement**$_2$ is evaluated. The expressions are evaluated until an expression evaluates to 1, or the else clause is reached. If the else clause is reached, **statement**$_n$ is executed.

There may be 0 or more **else if** clauses and these must follow the **if** clause. The **else** clause is never required and there may be at most one **else** clause. The **else** clause must follow an **if** clause (or a number of **else if** clauses, as long as the first clause is an **if**).

Looping

The CaRdY language has two types of loops, while loops and for loops.

1. **for loops**

   A **for** loop executes a statement a predetermined number of times, specified by a counter that is initialized, a condition upon which to terminate, and a change rule for the counter. The syntax for a for loop is:

   ```
   for (initialize; terminate; change) {
           statement;
   ```

}

1.1. **initialize** sets the value of the counter

1.2. **terminate** specifies when to exit the loop. **terminate** must be an expression that evaluates to a bool value. The loop continues so long as the value of **terminate** is 1 and stops when the value of **terminate** is 0.

1.3. **change** specifies a rule dictating how the counter value should be changed after each execution of the loop. In order to avoid an infinite loop, the **change** rule should eventually lead to the termination condition. The **change** rule may alter the counter either by decrementing or incrementing it using the standard binary operators.

The following example is a valid for-loop that prints the numbers from 0 to 8:

```
for (int count = 0; count < 9; count = count + 1) {
        print(count);
}
```

In this case, the initialized variable is **count**, with a value of 0. When the loop body statement is executed, **count** is printed and its value is then incremented by 1 according to the change rule. Once the value of **count** has been incremented is equal to 9, the print statement in the body of the loop stops executing.

2. **while loops**

The syntax of a while loop is:

```
while (continuation_expression) {
        Statement;
}
```

When the value of the **continuation_expression** is 1, the **statement** inside the loop is executed. This occurs until the **continuation_expression** no longer evaluates to 1.

The following example is a while loop that prints the numbers from 0 to 8:

```
int count = 0;
while (count < 9) {
        print(count);
        count = count + 1;
}
```

In the above example, the **continuation_expression** is “count < 9”.

Note that unlike a for loop, integer value **count** must be declared and initialized prior to the start of the while loop.

Classes

Classes are user defined data types. Classes in CaRdY act as blueprints for the objects they define. They have functions and data as attributes.

Class definitions are as follows:

```
class exampleClass {
    main(){
        expression;
        fun exampleFunction(int x, int y){
            print(x + y);
        }
      }
  }
```

Objects are instances of classes. They are instantiated as follows:

```
exampleObject = exampleClass();
```

Once an object has been instantiated, functions within it can be invoked as follows:

```
exampleObject.exampleFunction(1, 2);
```

Functions

User-defined Functions

Functions are defined using the keyword ‘fun’ followed by the function name, return type if applicable, and any parameters in parentheses. The body of the function is enclosed in curly brackets. Functions are assumed to be void if no return type is specified. For example:

```
fun int exampleFunction(int x, int y){
        print(x + y);
}
```

Function Calls

Functions are called on objects. The syntax is as follows:

```
int x = exampleClass.exampleFunction(1, 2);
```

Each class must contain a main function, which indicates the start of runnable code and has a void return type. Syntax is as follows:

```
class exampleClass {
    main(){
            /*function body*/
            }
    }
}
```

Built-in Functions

Refer to section CaRdY Standard Library section to see the built-in functions.

CaRdY Standard Library

CaRdY Types

Card(type, color, number)	Allows the user to create a custom card object to be added to a deck. The type attribute must be specified.
Player(score, hand_size, hand)	Creates a player object and keeps track of the cards in the player's hand.
Discarded()	Retrieves a list of discarded card objects.

Built-In Functions

createDeckCustomized({'type': [color, number, num_copies],..f.})	Takes a dictionary as a parameter where the keys are card characteristics (e.g. type) and the value is a list of the secondary characteristics (e.g. colors) and the number of cards to create of that type.
createDeck(num_copies, color, number, type)	Creates a deck (a list of card objects) with the specified number of copies of every combination of the specified attributes.
add(myCard)	Adds card object to the list on which it's called.
createStandardDeck()	Creates a standard deck of 52 playing cards.
shuffle(myDeck)	Takes in a deck or list of cards and shuffles it.
deal(player)	Deals cards to fill the specified player's hand.
discard(myCard)	Removes a card from the player's hand and adds it to the discard list.
draw(number)	Moves the specified number of

	cards from the deck to the player's hand.
display(player.hand)	Prints cards in player's hand.
display(player.score)	Prints player's score.
input()	Allows the user to input a variable from the keyboard to be read/used in the program.
print(val)	Takes in data as a parameter and prints it in the console.
reset()	Starts a new card game, deleting current hands and player scores but not the deck or existing players.
quit()	Quits the game, deleting all created decks, cards, hands, and players.

Lexical Conventions

CaRdY contains the following tokens: identifiers (names), keywords, expression operators, comparison operators, identity operators Constants, tabs, newlines, and whitespace are ignored.

1. **Comments**

   The characters **/*** introduce a comment. A comment must terminate the with characters ***/**

   /* Everything between the start and end of a comment is ignored***/**

   Comments may span many lines or a single line, but the above syntax must be used in either case.

2. **Keywords**

   The following identifiers are reserved as keywords and may not be used otherwise (i.e., as the name of a variable or class):

   - for
   - while
   - if
   - else
   - return
   - int
   - char
   - bool
   - float
   - str
   - and
   - or
   - not
   - in
   - is
   - set
   - dict
   - list
   - class
   - fun

3. **Punctuation**

    - Parentheses: all **if**, **else if**, **while**, and **for** conditional expressions must be enclosed in (parentheses)
    - All logical blocks, including the body statements of **if**, **else if**, and **else** statements, and **while** and **for** loops must be enclosed in { curly braces }
    - Semicolons are used to separate the conditions in a for loop as well as to denote the end of a line.

4. **Identifiers (Names)**

    An Identifier is a value or expression that has a name associated with it. a name associated with an expression of some value. All characters must be lowercase alphanumeric characters and the first character must be an alphabetical character.