CaRdY

Katrina Zhao (kz2335), Kenya Plenty (kgp2111), Liseidy Bueno (lb3347), Pazit Schrecker (prs2143), Lindsey Wales (lbw2149)

What is CaRdY?

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.

Why CaRdY?

While it's possible to create a console-based card game with Java or Python, the programmer has to go through the process of designing separate classes for each component of the game. Decisions like what data structure to represent the rules with and how the different classes will interact with each other can be tedious and repetitive. With CaRdY, much of the grunt work is taken care of and the programmer can instead focus exclusively on programming the rules of the game as efficiently and

smoothly as possible. In short, CaRdY is the perfect language to help you create the console card game of your dreams.

What does CaRdY look like?

From a syntactic perspective, CaRdY borrows heavily from Python because of its general ease of readability. On top of this, CaRdY will incorporate Java's strong typing policies in order to make coding the game rules as simple as possible. These elements will also ensure that the programmer can safely and easily override the game scaffolding that CaRdY has provided them with. With these features in place, the programmer can easily customize the flow of the game to meet their specific needs.

Syntax:

Types:

str: array of ASCII characters

int: 32-bit signed integer

float: 32-bit signed floating point number that contains a decimal. Floats can also be scientific numbers with an "e" to indicate power of 10.

bool: 8-bit boolean variable

*CaRdY is a strongly typed language, so types need to be declared. Data types are set upon assignment.

Operators:

Assignment Operators:

= : assigns a value to a variable (x = 5)

+= : adds a value to a variable and then sets the variable to that result (x += 5 is the same as x = x + 5)

-= : subtracts a value from a variable then sets the variable to that result (x -= 5 is the same as x = x - 5)

***=** : multiplies a value to a variable then sets that variable to that result (x *= 5 is the same as x = x * 5)

/= : divides a value by a variable then sets that variable to that result (x /= 5 is the same as x = x / 5

//= : divides a value by a variable then rounds that result down to the nearest whole number and sets the variable to that results (x //= 5 is the same as x = x // 5)

%= : divides a value by a variable then sets the value to the remainder of the result (x %= 5 is the same as x = x % 5)

Comparison Operators:

== : equals (x == y)

!= : does not equal (x != y)

> : greater than (x > y)

< : less than (x < y)

>= : greathan or equal to (x <= y)

Logical Operators:

and: returns true if both statements are true (x>5 and x<10)

or: returns true if one statement is true (x>5 or x<3)

not: returns false if the result is true (not (x>5))

Identity Operators:

is: returns true if two variables are the same object in the same memory location (x is y)

is not: returns true if both variables are not the same object (x is not y)

Membership Operators:

in: returns true if a specified value is present in a sequence (x in y)

not in: returns true if a specified value is not present in a sequence (x not in y)

Collections:

Lists: lists store multiple objects in a single variable. Lists in CaRdY function similarly to lists in Python.

Sets: like lists, sets store multiple objects in a single variable, but are unindexed and unordered. Sets in CaRdY function in the same ways as Python sets.

Dictionaries: dictionaries store key/value pairs. Dictionaries in CaRdY are similar to dictionaries in Python.

Objects and Classes:

Card(type, color, number): Card() allows the user to create custom card objects to be added to a deck. The type attribute must be specified.

Player(score, hand_size, hand): Player() creates a player and keeps track of their cards.

Discarded: discarded is a list of discarded card objects.

Built-In Functions:

createDeckCustomized({'type': (color, number, num_copies), ...}): this function takes a dictionary as a parameter where the keys are card characteristics (e.g. type) and the value is a tuple containing a list of the secondary characteristics (e.g. colors) and the number of cards to create of that type (see code below for an example creating an Uno deck).

createDeck(num_copies, color, number, type): This function makes a list of card objects as the deck by making the specified number of copies of every combination of colors, numbers, and types from the lists specified beforehand (see sample code for example). This is useful for creating simpler decks.

add(): adds card objects to the list on which it's called (i.e. myDeck.add(myCard)).

createStandardDeck(): creates a regular deck of 52 playing cards.

shuffle(myDeck): takes in a deck or list of cards and shuffles it.

deal(number): deals a certain number of cards to the player's hand. Cards are removed from the deck when they're added to the player's hand.

deal(player): deals cards to fill the player's hand.

discard(): takes a card from the player's hand and puts it into the discard list.

draw(number): the player draws the specified number of cards from the deck to add to their hand. This deletes the card objects from the deck list and adds it to the player hand list.

display(): quick output for player to check their cards/scores, takes two types of inputs:

- **display(player.hand):** input is the player 's hand
- **display(player.score):** input is the player's score

input(): allows the user to input a variable from the keyboard. This variable can be read and used in the program.

print(): takes in a data type as parameter and prints it out in the console.

reset(): starts a new game of cards. Deletes current hands and player scores but doesn't delete the deck and players that have been created.

quit(): quits the game. Deletes all created decks, cards, hands, players. Ends a program completely and the user will have to rerun the program to play again.

Indentation and Brackets:

Indentations are only used for readability. Brackets ({ }) are used to denote blocks of code.

Comments:

Comments start with the hashtag symbol (#) on each line.

Semicolons

Similar to Python, semicolons are not necessary in CaRdY.

Reserved words

And, break, Card, class, color, del, Deck, else, False, for, if, in, is, player, return, True, type, while

Sample Code:

GCD Algorithm:

```
def gcd(x, y):
     if (x>y):
          int lo = y
     else
          int lo = x
     for i in range(1, lo+1):
          if((x%i==0) and (y%i==0):
               gcd = i
```

```
        return gcd
```

Creating a simple deck:

```
var colors = [“red”]
var types = [“diamonds”, “hearts”]
var numbers = [“A”, “2”, “3”]
var myDeck = createDeck(1, colors, numbers, types)
# the list myDeck now contains 6 card objects that all
have the attribute “red” for color. The deck has one
ace of diamonds, one ace of Hearts, one 2 of Diamonds,
one 2 of Hearts, one 3 of Diamonds, and one 3 of
Hearts.
```

Creating an Uno Deck:

```
var unoColors = [“yellow”, “blue”, “red”, “green”]
var myDeck2 = createDeckCustomized({‘Draw2’:
(unoColors, null, 2), ‘Draw4’: (unoColors, null, 2),
‘1’:(unoColors, null, 2), ‘2’:(unColors, null, 2)})
# the list myDeck2 now creates a list of 32 cards: 8
Draw 2 cards, 2 of each color; 8 Draw4 cards, 2 of
each color; 8 ‘1’ cards, 2 of each color; 8 ‘2’ cards,
2 of each color. Because the type here is the number
or type of card, the “number” attribute is entered as
null.
```