A Comparison of Python, JavaScript and Lua Scripting Language Features

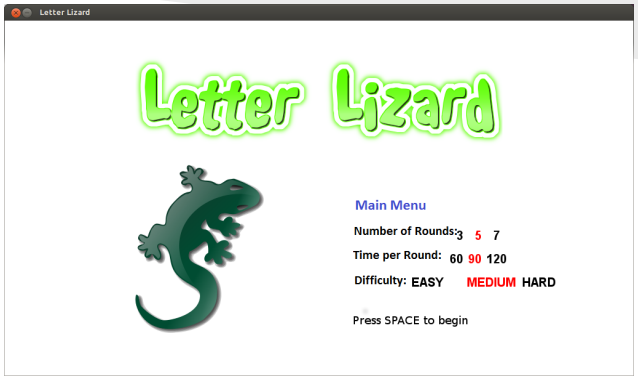
CS798 Scripting Languages Project Final Presentation
March 31, 2014

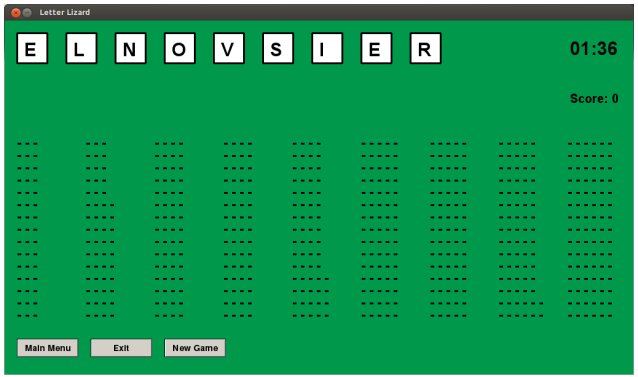
Afiya Nusrat, Alexander Pokluda and Michael Wexler

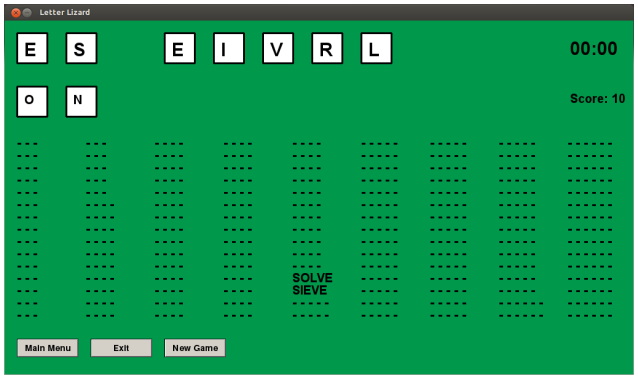
Outline

- Introduction
- Letter Lizard Implementations
  - Python Implementation
  - JavaScript Implementation and Demo
  - Lua Implementation
- Scripting Language Feature Comparison
  - Lexical Structure
  - Data Structures
  - Object Oriented Programming Features
  - Language Specific Features
- Conclusion

Python Implementation







Python Implementation Design

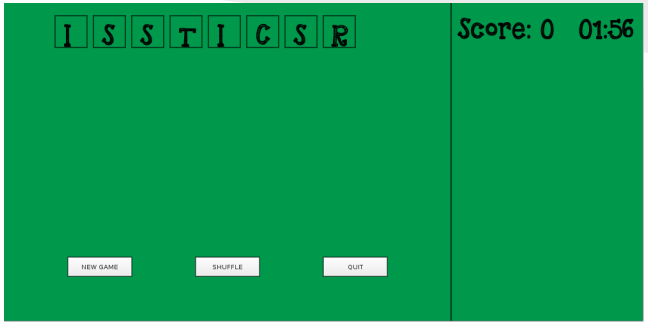
- Utilized PyGame
- Three modules:
  - letter_lizard.py
  - config.py
  - game.py
- At the core of the game is the Game Loop:
  - while (True):
    - Process Events
    - Update Game State
    - Redraw Screen

JavaScript Implementation

JavaScript Implementation Design

- Object-Oriented Design based on callbacks with classes for manipulating interface and game state:
  - **`Tile`**: Represents a letter in the set of letters shown to the player
  - **`Scramble`**: Manages set of letters, shuffles them
  - **`Builder`**: Handles key presses and moves tiles to form words
  - **`Word`**: Represents a word to be found
  - **`Game`**: Manages words to be found, generates hints
- Free functions for showing the splash screen, game screen, etc.

Lua Implementation



Lua Implementation Design

- Game Engine 
- Utilises ‘callbacks’
- Game functionality structured within callbacks
- Update is called continuously and takes in parameter ‘dt’ - utilized in the game for updating gamestate
- Game drawing done in love.draw

```
218 function love.load()
219     This function is called exactly once at the beginning of the game.
220 end
221
222 function love.update()
223     Callback function used to update the state of the game every frame.
224 end
225
226 function love.draw()
227     Callback function used to draw on the screen every frame
228 end
229
230 function love.keypressed()
231     Callback function triggered when a key is pressed.
232 end
233
234 function love.mousepressed()
235     Callback function triggered when a mouse button is pressed.
236 end
237
238 function love.quit()
239     Callback function triggered when the game is closed.
240 end
```

Lexical Structure

Python

- Designed to be highly readable
- Uses English words instead of punctuation
- Uses whitespace indentation rather than curly braces or keywords to delimit blocks

JavaScript

- Free-format
- Automatic semicolon insertion: some statements that are well formed when a newline is parsed will be considered complete
- Curly braces are used to delimit blocks

Lua

- Free-format
- Newlines used to delimit statements
- Keywords used to delimit blocks

Lexical Structure Comparison

	Python	JavaScript	Lua
Strengths	Good indentation is enforced by language making code easy to read	Curly brace delimited blocks means code can be “minimized” for the Web	Statements and blocks delimited by newlines and keywords help prevent errors

Weaknesses	Tabs and spaces can easily be mixed which can lead to bugs	Automatic semicolon insertion can lead to errors, eg:
		`a = b + c`
		`(d + e).foo()`
			Code can be difficult to read unless indentation conventions are followed

Data Structures: Python

- Sequence Types
  - List: mutable sequence of items of arbitrary types

    ```
    self.letters_guessed = []
    ```

  - Tuple: immutable sequence of items of arbitrary types

    ```
    red = (255, 0, 0)
    ```

  - Range: immutable sequence commonly used for looping

    ```
    for i in range(num):
    ```

- Set: mutable containers of items of arbitrary type

  ```
  self.words_guessed_correct = set([])
  ```

- Dictionaries: mutable mappings from keys to values

  ```
  lengths_to_words = {}
  ```

- Can create Lists, Sets, Dictionaries inline with comprehensions

Data Structures: JavaScript

- Fundamental data type: Object
  - Dynamic, unordered collection of properties (name-value pairs), similar to Python dictionary with String keys

```
this.words = {};
for (var i = 0; i < game.words.length; ++i) {
      var word = game.words[i];
      this.words[word] = new Word(word);
}
```

  - Can be used to simulate sets of strings (by ignoring values)
  - Language support enables objects to be used as array

Data Structures: JavaScript

- Array: Special type of JavaScript object with integer keys and automatic `length` property

```
shuffle: function() {
	// We need to skip tiles that have been moved to the builder
	var mytiles = [];
	for (var i = 0; i < this.tiles.length; ++i) {
		if (this.tiles[i]) {
			mytiles.push(i);
	}	}
	mytiles = shuffle(mytiles);
	var tilescpy = this.tiles.slice(0);
	for (var i = 0, j = 0; i < this.tiles.length; ++i) {
		// Move tile at position j to position i
		if (this.tiles[i]) {
			var tile = tilescpy[mytiles[j++]];
			tile.moveTo(this.x + 10 + 60 * i, this.y + 10, true);
			tile.scramblePos = i;
			this.tiles[i] = tile;
}	}	}
```

Data Structures: Lua

- Fundamental data type: Table
  - Similar to JavaScript Objects, but can be indexed with any value of the language (except nil)
  - The only data structuring mechanism in Lua
  - Tables are associative arrays
  - Can be used to implement arrays, sets, records and other data structures
  - Ease of creation:

```
games_letters = {}
games_letters = str_to_table(games.easy[1].letters)
games_words = {}
games_words = games.easy[1].words
```

Data Structures: Lua

- Array: Like JavaScript, special support is provided for tables containing values with integer property names
- Array length operator # returns the largest index in the array table

```
function print_array(arr)
    for i = 1, (#arr) do
        print(arr[i])
    end
end
```

Data Structures: Comparison

- Although JavaScript and Lua do not offer as many data structures as Python, most can be easily simulated (but requires extra work)
- JavaScript is the most limiting because object property names must be strings (or integers)
  - This makes implementing a generic set difficult
- JavaScript objects, Lua tables, and arrays were sufficient for our implementations

OOP Feature Comparison

Python

- Class mechanisms based on C++ and Modula-3
- Provides standard features of OOP including multiple inheritance and method overloading
- Private members provided through *name mangling*

JavaScript

- Prototype-based inheritance
- Can emulate many features of “classical” OOP
- Classes can be dynamically extended and support “duck typing”

Lua

- Colon operator adds hidden *self* parameter to function calls
- No notion of classes, but prototype-based inheritance can be implemented using *metatables*
- Metatables are like JavaScript prototype, but more powerful

Conclusion

- We compared the features offered by the scripting languages Python, JavaScript and Lua
- The basis of our comparison was the implementation of the Letter Lizard game in each language

Conclusion

	Python	JavaScript	Lua
Main Strengths	- “Batteries included” philosophy makes most common tasks trivial	- Closures and the callback-based design led to clean and modular code	- Tables provide a flexible and efficient multi-purpose data structures - The dynamic nature reduces the amount of code
Main Weaknesses	- Lack of an explicit variable declaration statement results in “broken” lexical scoping - Performance issues	- Lack of a general-purpose data structure makes some tasks challenging	- Very low-level; you have to code many basic functions yourself - Lack of built-in object oriented support increases difficulty in implementing some features

Questions?