Salutations, I'm Jesco and welcome to GameDevMadeEasy, where we break down the process of game development into easy-to-follow tutorials.

The topic for today is creating an addon for World of Warcraft with Lua. Getting started with creating a Lua addon for World of Warcraft can be a bit challenging due to the lack of official Lua documentation on the Blizzard website or forums, which means you'll need to search the internet to find the necessary information.

A couple of helpful resources for finding information on available functions is [<https://wowwiki-archive.fandom.com/wiki/World_of_Warcraft_API>](https://wowpedia.fandom.com/wiki/World_of_Warcraft_API) and

<https://wowpedia.fandom.com/wiki/World_of_Warcraft_API>

Today, we are going to create a couple simple addons. The first of which will print the food types that your hunter’s pet can eat.

Let's start by understanding the structure of a WoW addon. At its core, a WoW addon will consist of a single Toc file and one or many Lua files.

A Toc file needs to have a title, author and version. It also needs to have the name of the lua file with the lua extension associated with this addon.

A Toc file serves as a blueprint for WoW to identify the addon, specify dependencies, define the version number, determine the load order of Lua files, and include metadata and optional settings.

Now let's delve into the structure of a Lua file.

At the beginning of the file, it's recommended to define variables you intend to use, unless they have a function scope rather than being scoped to the class. In programming, you have different levels of scope. Global scope, class scope and function scope. Global scope means that it is accessible everywhere. Class scope means that it is accessible anywhere in the class. And function scope means that it is only accessible in that function.

It is time for the fun part, writing the code.

Create a folder called AddonTutorial, I would recommend putting this inside of the addon folder in your WoW directory. I would also recommend opening this folder in VS Code.

Create a file called AddonTutorial.toc

Remember to name the toc file the same as the folder it resides in, including the file extension (.toc).

The first item we will add to this file is.  
  
## Title: AddonTutorial

This tells WoW that the name of this addon will be called AddonTutorial.

Next,  
  
## Author: GameDevMadeEasy

This tells WoW who made the addon, feel free to change this to your name.

The following item is,

## Version: 0.0.1

This tells WoW what the version number is for the addon. It is important for tracking changes / updates and is heavily utilized by platforms that host WoW addons.

The next item is,

## SavedVariablesPerCharacter:

We will utilize this feature in a later tutorial. For now, understand that it serves to store data that can be accessed at a later time.

The final metadata item is,

## Notes: Tutorial on creating addons

This serves to give a description of the addon.

The final item for this file will be,   
HunterPetFood.lua

This links the lua file to this toc file and makes it readily available for use. It also means that if we have multiple lua files, that this will be the one that is loaded first.

Now we can create our lua file with the name of HunterPetFood.lua.

What we want to do now is look at the handy website that has all of the available functions that Blizzard has so graciously created for us to use.

If we do a search for Pet, we can see that there are quite a few functions related to the pet that we can use. The one we want is called GetPetFoodTypes.

And if we click on that function, we can see that it will return a list of the food types the player's pet can eat. We can also see that it will return multiple strings and not a table.

This information is vital as it gives us a clue on how we should write our code.

Moving back to the code editor, we can write

local petFoodList = { GetPetFoodTypes() };

The code initializes a local variable named "petFoodList" and assigns it the result of the "GetPetFoodTypes()" function call. By enclosing the function call in curly braces, the returned food types are stored as elements in a Lua table. The "local" keyword indicates that the variable is scoped locally and can only be accessed within its block of code or function.

On line 2, we want to create two local variables index and foodType.

These variables will be used in the next line.

For line 3, we will be creating a for loop. If you have any experience with other programming languages, you might notice that the loop is kind of familiar with some differences.

We will write for index, foodType in pairs(petFoodList) do

On line 4, we will add spaces until we get to the I in index from line 3. And we will write print(foodtype)

On line 5, we will type end.

Like JavaScript, Python and Ruby; Lua does not generally require a semi-colon at the end of a line and instead is very much like Python and Ruby where you must take care to properly indent and tab lines.

Save this and load WoW, if you have a hunter that has a pet, then this should work in Wrath of the Lich King Classic and Retail WoW.

Not bad, but it only displays it in the chat when you first log in and doesn’t update when you change pets. We can do much better.

In the AddonTutorial.toc file, let’s add another line under HunterPetFood.lua. We are going to put HunterPetFoodExpanded.lua here.

We are also going to create another Lua file in the Addontutorial folder with the name of HunterPetFoodExpanded.lua.

Before we continue, I want to note that the Lua files in the toc file are loaded from top to bottom with our current addon structure. Hunter Pet Food will be loaded and then Hunter Pet Food Expanded will be loaded. They are also treated as two separate entities within the same addon, which means you can have unrelated items function within your addon at the same time. You are also able to create a global object which will feed functions or tables to your other Lua files. More on that later.

The vision for this version of the Hunter Pet Food display is simple. It will display the text at the top of screen in large but easily readable font and every second, it will change colors. We also want this version of the display to update when changing pets or when you don’t have a pet active.

This means that we will be able to utilize some aspects of the previous code, but we will have to expand upon it.

The first thing we are going to do is create a new frame. This is supplied to us from the WoW API. It has 3 required arguments and 1 optional argument. Frame Type, which is the type of frame that will be created. Such as a Frame or Button. Frame Name, which gives a name to the newly created frame. Parent Frame, the frame object that will be used as the created frame’s parent. Inherits Frame, which are frames created using XML.

Write local display Frame equals Create Frame open parenthesis, quote, Frame, end quote, comma, open Quote, Pet Food Display Frame, end quote, comma, UI Parent, close parenthesis.

This will create our frame, but now, we need to set the size and location of this frame.

Display Frame colon Set Size, open parenthesis, 400 comma, 30, close parenthesis.

Display Frame colon Set Point, open parenthesis, open quote, TOP, end quote, comma, 0, comma, 0, close parenthesis.

This will give the text a large font size, relative to your display’s resolution in-game and place it at the top of the screen.

The next thing we want to do is create text that will live inside of the frame. The WoW API provides the Create Font String function for handling this.

This function has three arguments. Name, which is the name for the global variable that points to the newly created font string. This can be set to nil and not have a global variable created. Draw Layer, which is the layer in which the font should be drawn, which could include ARTWORK or OVERLAY. Template Name is the last argument, and it is the name of the virtual template previously defined with a layout Frame. If it is set to nil, the font does not inherit any properties.

Write local display Text equals display Frame colon Create Font String, open parenthesis, nil, comma, open quote, overlay, end quote, comma, open quote game font normal large, end quote, close parenthesis.

Display Text colon Set All Points, open parenthesis, close parenthesis.

Display Text colon Set Justify H, open parenthesis, open quote, Center, close quote, close parenthesis.

Set All Points adjusts the frame’s dimensions and position to match those of the parent frame. This means that the text will be stretched to fill the entire available space made from the display Frame.

Set Justify H adjusts the text to be centered horizontally.

We now need to ensure that our display frame will display on screen.

Write display Frame colon Show with an open and closing parenthesis.

Now we can create some functions. Remember, the is crucial in Lua, if you get this wrong while typing this code out, you will get errors.

The first function will be a function version of the original code which will pass the foodType to the displayText.

Local function UpdatePetFood, open and close parenthesis. This will denote that we are creating a locally scoped function that can only be used in this lua file.

On the next line, have spaces until you reach the c in local.

Write local petFoodList equals open curly bracket, GetPetFoodtypes, open and close the parenthesis and close the curly bracket.

This is the same as with the original version. We are getting a Lua table of the pet food types.

Now write local petFoodDisplay equals begin and end quote.

We are initializing an empty string value to this variable.

The following line is local index comma foodType. We are initializing the index and foodtype variables for use.

Now for the for loop. For index comma foodtype in pairs, open parenthesis, petFoodList, close parenthesis, do.

On the next line, we want the text to start right under the r in for. Write petFoodDisplay equals foodtype.

The following line, write end and it should coincide with the f in for.

This next line, we want the text to begin right where the e in end lands. Write, displaytext colon set text, open parenthesis pet food display, close parenthesis.

This will set the text for the display text to be the value of the pet food display text value.

We can now close this function with the word end and it should align with the l in local for the function on line 9.

One of the requirements we had for this version is to randomize the text color. We will create a function to handle this.

Write local function Randomize text Color with an open and close parenthesis.

The next line will have the spaces like before and we will write local r, comma, g, comma, b equals math dot random with an open and closing parenthesis, comma, math dot random with an open and closing parenthesis, comma, math dot random with an open and closing parenthesis.

We are defining our red, green and blue variables to be random values between 0 and 1.

Follow that up with displayText colon Set Text color, open parenthesis, r, comma, g, comma, b and a closing parenthesis. This will set those values to the r, g, b values associated with the Set Text Color function available to us.

Write end to close this function off.

This next function will take that Randomized Text Color and give it the ability to periodically change the color. In our case, we want it to update every second.

Local function Periodically Randomize Color with an open and closing parenthesis.

On the next line with the correct spacing, we are just going to write Randomize Text Color with an open and closing parenthesis. This calls our previously created function to be run from this function.

To create a timer, we will again leverage a function made available to us from the WoW Lua API, this function is C\_Timer.After.

The C\_Timer.After is a repeating timer that calls itself recursively. A recursive function is just a function that calls itself. C\_Timer.After requires 2 arguments. The first being the delay in seconds before the timer is triggered and the second argument is the function that will be called when the timer is triggered.

Write, C\_Timer.After, open parenthesis, one, comma, Periodically Randomize Color and then close parenthesis. On the following line, write end to close the function.

All functions have been created and we are almost done. The last portion is to create, register and utilize events to ensure that we have the functionality we require. Which is making sure that it updates when we change pets or don’t have a pet active.

Write local frame equals Create Frame, open parenthesis, begin quote, Frame, end quote and close the parenthesis.

We want to create another frame just for storing our events.

We will utilize four WoW specific events. Unit pet, pet bar update, pet stable update and pet UI update.

Write frame colon register event, open parenthesis, begin quote, Unit underscore pet, end quote and closing parenthesis.  
Following the same structure for the next call, our text will be Pet underscore bar underscore update.  
Number three is pet underscore stable underscore update.

Number four is pet underscore UI underscore update.

Now for the final six lines of code to complete this addon.

Write frame colon set script, open parenthesis, begin quote, On Event, end quote, comma function, open parenthesis, self, comma, event, comma unit and a closing parenthesis.

Using proper formatting to be inside of a function, write if event double equals begin quote, Unit underscore pet, end quote or event double equals begin quote, pet underscore bar underscore update, end quote, or event double equals begin quote, pet underscore stable underscore update, end quote or event double equals begin quote, pet underscore UI underscore update, end quote then.

On the next line with spaces for formatting, write Update Pet Food with an opening and closing parenthesis. Then follow it up with another line that has Periodically randomize Color with an opening and closing parenthesis.

Write end on this line to close the if statement.  
For the final line, write end with a closing parenthesis.

This takes our registered events and anytime the game registers that an event has been fired, it will check if our designated events have fired and run our two functions.

Congratulations, you have created a WoW addon! As always, there is a link to the completed code in the description of this video. If you want more complex WoW Addon tutorials, leave a like and comment below. Now, if you don’t mind, I’m gonna go bash my head into the keyboard from my never ending quest to learn and improve my skills further.