Aardvark

Release 0.8.7

Dec 19, 2021

Contents

CHAPTER 1

Getting Started:

To get started with your first output do: `output('hello world!\n')`. There you go, you've officially ran your first program! Take note of the `\n`. If you (like me) get tired quickly of writing `\n` you can go ahead and use `#include anr` press enter, and then run `output('hello world!')`. See! works without the `\n`. We'll talk more about `#include` later, but for now thats all you need to know. Comments start with `//` and end with `\\`, but versions below `0.8.7` use `/` and `\`. Clearing the terminal is done by using `clear()`.

CHAPTER 2

A Note About Built In Functions:

Most (excluding `output()`, and `input()` ect...) functions do not output things on their own!! As such it is integral that if you want an output from (most) built in functions you output the result by assigning it to a variable, and outputting the variable, or calling the function from inside an output. Like this: `output(binf())`, or:

```
a = binf()
output(binf)
```

Now that we've cleared up this common missconception, proceed!

CHAPTER 3

Running Files:

To run files all you need to do is run `#include file-to-run` this imports all aspects of a file and runs them directly to the terminal.

CHAPTER 4

Math And String Concatenation:

Math works as usual (Multiplication: *, Division: /, Addition: +, Subtraction: -), and `ints()` are defined as `number(perameter)`. String concatenation is done by simply putting an addition sign between strings: `'string '+'concantonation!'`, `'string ' + 'concantonation!'` is also valid.

CHAPTER 5

Booleans

Booleans in Aardvark are `True` and `False`, they can both be assigned to a variable. (`true` and `false` for versions 0-0.4 and 0.8.7+)

```
myBoolean = True
if myBoolean == True {
  output('Nice!')
}
if myBoolean == False {
  output('Sad...')
}
```

CHAPTER 6

If Statements:

If statments work like most other langages; it runs a check to see if the conditions described in the first line (the one with the initial `if`) are met. If the conditions are true it will execute the given code inside the brackets. Syntax:

```
if condition(s) {
    result
}
```

CHAPTER 7

Recieving User Input:

To get user input use the `input()` function. Syntax: `input(prompt)`. This can be assigned to a variable like so: `myVar = input(prompt)`, as it is assigned to a variable you can check if (in this case) `myVar` is equal to something code-block:

```
myVar = input('How are you? ')
if myVar == 'good' {
    output('Nice!')
}
```

CHAPTER 8

While Loops:

While statements are so useful in all the languages, and also in Aardvark! While is used to check if something is true, and if it is true it will continue to execute the code within the while block until it is false (this can also work if something is always true).

:: n = 0 while n < 10 {

```
        output(n) n += 1

    }
```

CHAPTER 9

Functions:

Functions are defined by the `funct` keyword. Syntax:

```
funct myFunction(args) {
    do something
}
```

Function can be called by typing the function's name with parenthese at the end (if the function has arguments include the argument values too!) like so: `myFunction()`, or `myFunction(args)` if the function was defined with arguments. Function arguments are seperated by commas (`,`). Functions defined by the user run like any other function. To return from a function simply do `return data`, `data` can be equal to anything, a string, a number, function, or a variable ect…

CHAPTER 10

Directives:

All directives start with `#`, there are currently 3 directives, `#include`, `#ape`, and `#max-memory`. `#include` includes the specified module. Syntax:

```
#include file
```

`#max-memory` sets the program's maximum allowed memory. Syntax:

```
#max-memory number / For instance Sets the maximum memory to 50mb \
```

CHAPTER 11

File Handling:

Aardvark's file handling is very similar to Python's, as it is very straight forward: `open(file).read()` will read a file, `.write(data)` will write something to a file, `.append(data)` will apend something to the end of a file. You can also open files into variable like so: `a = open(file)`. You can get the ammount of space a file takes up in kilobytes by using `file_size()`, Syntax: `file_size(file)`.

CHAPTER 12

APE:

APE is Aardvark's package manager, it stands for Aardvark Packager Extension. You can install .adk files from the website. Go ahead, and type `#ape atest`, that will install atest.adk on to your computer. To run that file just `#include atest` (it has to be in the same directory as the file you are running it from or in your `scripts` folder). Extension/Packages can also be writen in python, these have to be in the same folder as `main.py`. To do this you have to put `#Aardvark.library` at the beginning of the file, and don't forget to do `from Aardvark import *` this allows you to use Aardvark's function and type creators, amoung other things: `Aardvark.function('function_name')`, `Aardvark.type('name')...` To learn more about this look into `main.py` and your `Language` folder.

CHAPTER 13

Visual Module:

To use the visual module first you have to include it (`#include visual`), after that you are good to go! The visual module allows you to display things to the screen in a sperate window, to first initiate the window you do: `visual(title, geometry)`, the window geometry is formated as follows: `NUMBERxNUMBER`. To make words apear to the screen use the label method: `label(text, foreground, background, xcord, ycord)`, for the list of colors see Tkinter's list of colors. To recieve input use: `entry(prompt, foreground, background, xplace, yplace).` All of these methods can be assigned to a variable. Last off, to make the window visible use: `show()`, make sure you do this, otherwise your window will be invisible. Example:

```
#include visual
visual('myWindow', '800x800')
label('hello world!', 'black', 'white', 80, 150)
entry('Entry! ', 'black', 'white', 100, 100)
show()
```

CHAPTER 14

Server Module

To start, lets include server using `#include server`, when thats done, you can start making your first Aardvark web server. Aardvark we servers work similar to python's flask (If you know what that is). Lets start by making a basic website in just 3 lines of code.

```
#include server
render_string('Hello World!')
run_server()
```

Run that and there you go, your first Aardvark web server. Now, lets learn something a little harder, rendering files. In `render_string()` you can add the second argument for the part of the site it will show up on. There is also `render_file()` which takes the same arguments as `render_string`, except that the first argument is the name of the file to render. Make a file called index.html and put some html code in it. And try this code:

```
#include server
render_file('index.html')
run_server()
```

Run the code and look at the output, your html file shows up in the browser. Now we will learn how to catch errors like 404. You can use `errorhandler()` to do that. `errorhandler()` takes to arguments, the error code, and the file to run if that error comes. Make a file called error.html and write an error message. Now run this code:

```
#include server
render_file('index.html')
errorhandler(404, 'error.html')
run_server()
```

And try going to a page that does not exist like `/abc.html` for example, your error message should come up.

CHAPTER 15

Exec And Running Other Langs:

In Aardvark you are able to run Python, and C++, this can be done by using the `exec()` function: `exec('code', 'language')`. Keep in mind this is for code snippets, NOT FULL PROGRAMS. Example:

```
exec('print("Hello world in python!")', 'py')
```

CHAPTER 16

Current Memory:

You can recieve the program's current memory usage by doing `currentMemUsage()`. This takes no arguments.

CHAPTER 17

Data Types:

CHAPTER 18

Tools Module:

To include the tools module in your program use: `#include tools`. This allows you to do things with random numbers, and factorials. First off to use factorials just do: `factorial(number)`. There are multiple function dealing with random things, first is `random()`, it returns a random number between 0 and 1. Next is `randomchoice()`, it chooses a random item from an iterable. Next is `randint()`, it return a random integer within the specified range.

CHAPTER 19

Database Module:

To have access to databeses you first have to include the db module, `#include db`. There are 2 functions in the db module: `addKey()`, and `loadData`. Lets start with addKey; db connects to Json, (it is recomended that you have some knowlege of Json before using this module) to add a key, and value to a Json file. The syntax for this is: `addkey(key, value, file)`, if nothing is passed into file it will default to `db.json` in the folder that Aardvark is kept in. The other function, `loadData()` returns the data of a specified Json file. Syntax: `loadData(file)`, if nothing is passed into the file parameter it will return the data of `db.json`.

CHAPTER 20

List:

List are created by using the `list()` function, or []. They can be assigned to variables. See `list.py` for more info!

CHAPTER 21

File System Module:

To include this module use `#include filesystem`. This adds 1 more function to your toolbelt: `newFile()`. The function allows you to create a new file. Syntax: `newFile(filename, text)`, if nothing is passed into the text parameter it will default to a blank file.

CHAPTER 22

NLP Module:

The NLP module consists of many functions useful in Natural Language Processing. Lets start with `#include nlp`. The first is `clean()`, it returns only the most important words in the given text. The next is `GetWordInfo()` it will return the information for any given word. The following code:

```
#include NLP
GetWordInfo('hello')
```

Will return a dictionary of all the synonyms, antonyms, and defintions of the given word. The next function is `ProcessList()`, this function takes 1 argument, a dictionary, and process it to remove any problems. The last function is `GetTopics`, which gets the topics of a conversation, it takes 1 manditory argument, a list of strings, and returns the main topics of the conversation.

CHAPTER 23

Timer module

The timer module has 4 functions that can be used by doing `#include timer`: `waitSeconds()` `waitMinutes()`, `waitHours()`, and `currentTime()`. All of the Wait functions take in 1 number perameter, the program will wait until that ammount of time has passed until proceeding with the program. Example:

```
#include timer
output('Hello,\n')
waitSeconds(8)
output('World!\n')
```

In the code above would output 'Hello,', wait 8 seconds, and then output 'World!'. The other function, `currentTime()`, takes in 0 perameters, and returns the current time.

CHAPTER 24

Regex Module:

See regex documentation for Python https://docs.python.org/3/library/re.html

CHAPTER 25

System Module:

The system module currently only has 2 functions: `blockStdout()` and `enableStdout()`, `blockStdout()` blocks the stdout, and `enableStdout()` reenables it.

CHAPTER 26

Closing:

Thats it for now! More features will be added in the future, and syntax will change so make sure to keep up to date with the docs! Thanks!