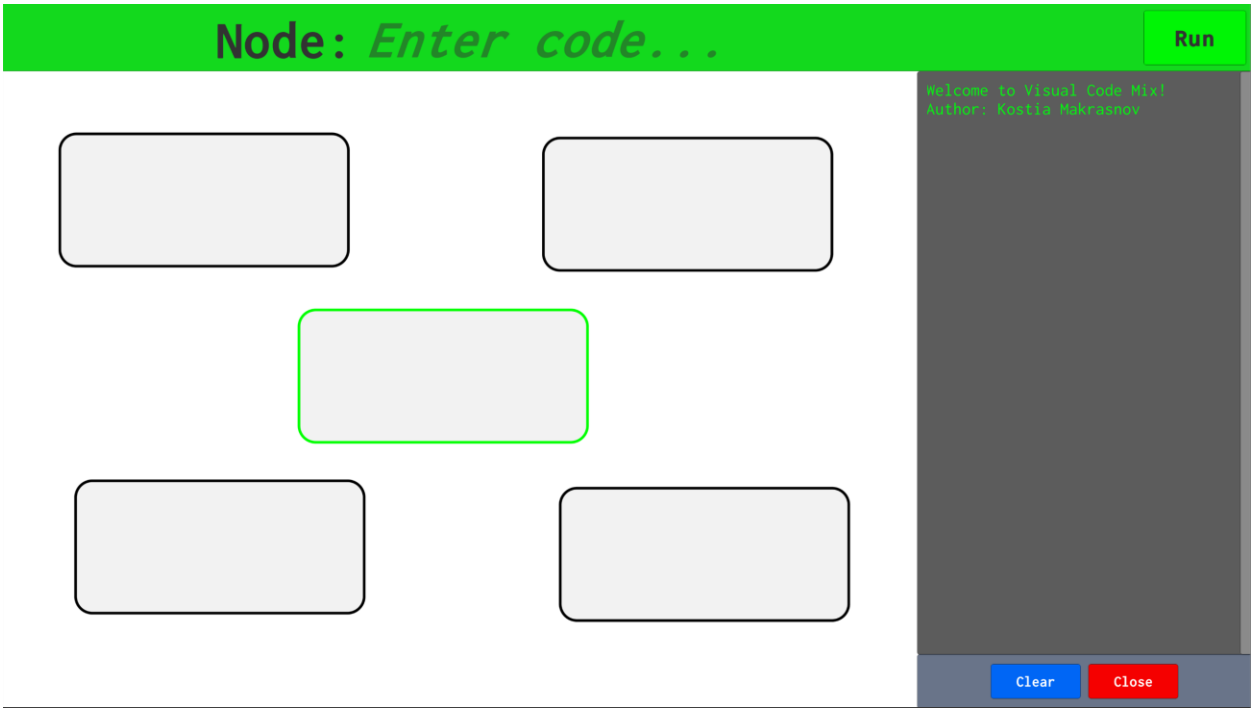
Visual Code Mix Documentation

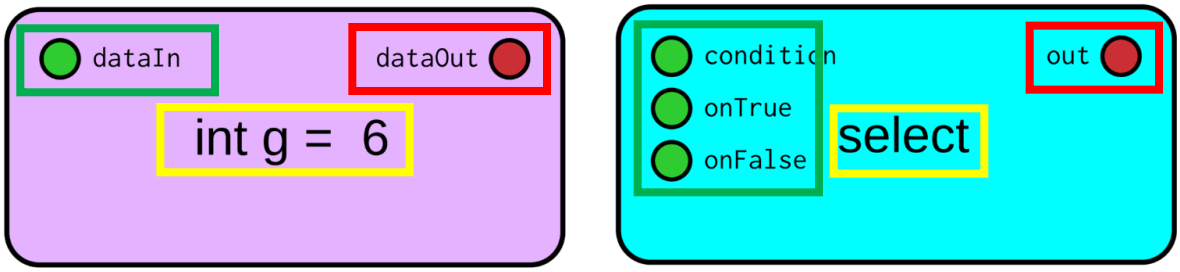
Because of the nature of many visual scripting languages the usage of visual code mix is different than many text-based scripting solutions. This means a greater amount of interaction from the mouse will be required and code flows not from top to down but rather through a set of nodes that resemble a graph.

Main UI Elements:



The basic components of the UI is the node canvas which takes up the majority of the screen and is where most of the code is located. The top of the screen is a large, single line input field that changes depending on the current selection in the node canvas. To run an assembled node graph, the user needs to click the top-right “Run” button. Output from any “print” nodes or errors in execution are listed on the output log on the right side of the screen. This output log can be either cleared or closed for a more cleaner program look. Note this output section will only reappear upon pressing the run button.

Node Anatomy:



Input Links: provide information to the node prior to the computation of output
Output Links: provide output information after node is computed to other connected nodes
Node Text: provide identifying information about node sub type and value (for data node)
Node Color: indicates the type of node

Node Interactions:

At any point in the development the following set of actions on the nodes and their children are available.

- **Create a node** by double clicking anywhere on the white canvas
- **Editing a node** by clicking once on the body of the node and then typing in the input field on the top of the screen
- **Connect output of one node to input of another** by first clicking one on the output circle of the source node (red), then double clicking on the input of another node (green). If done correctly a connection line should appear between two nodes
- **Delete a node** by selecting a node and then pressing the delete key

Node Editing:

While editing the nodes through the input field the following set of commands is available. It is important to press the *enter* key to progress from one type of node to another, all parameters should be separated with a space.

Empty Node:

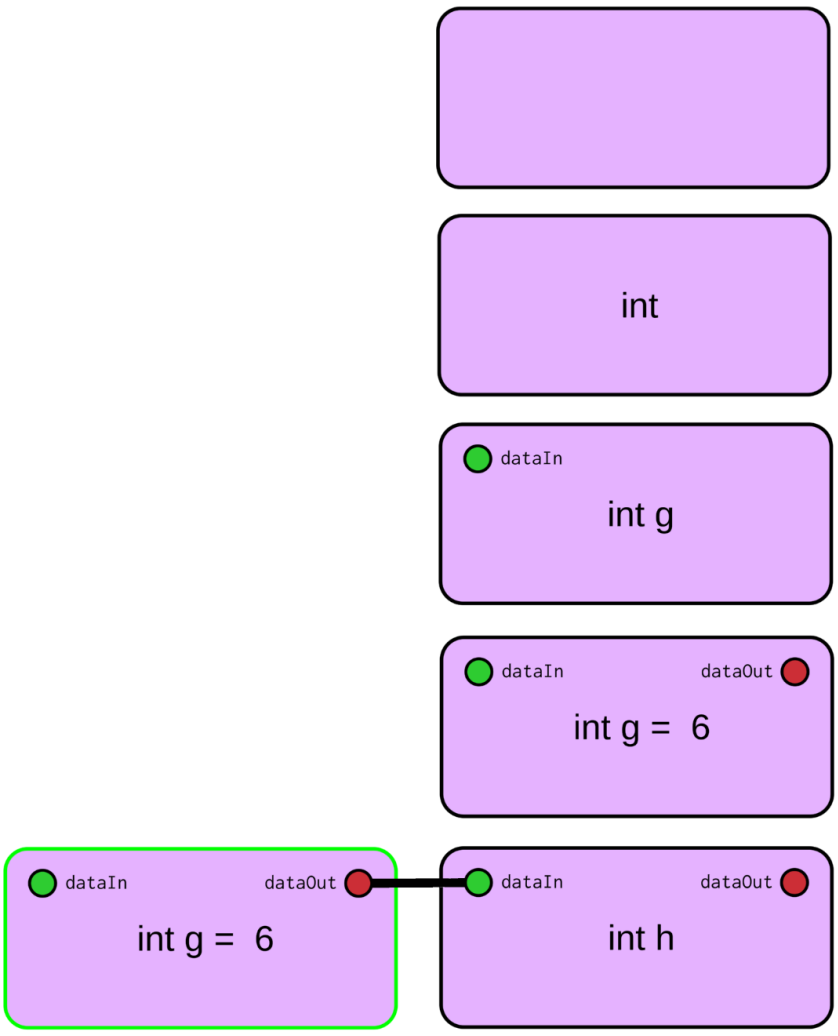
By clicking on an empty node, you are able to give it one parameter in the top input field, the node type.

Available types:

- **data** OR **1** (a variable node that allows to store a value)
- **function** OR **4** (any input → output type nodes that modify or use incoming messages)

Data Node:

Depending on the set of parameters that are provided, a data node can have different text states and link options. Essentially a data node is the primary input to the graph program. These nodes are what the program will start with and also where it can store or copy information during execution.



Empty Data Node
“”
Not useful for anything

Type Only Data Node
“int” (can do ‘’bool”, “double”, “string”)
Empty data node with data type declared

Named Data Node
“int g”
With data name you get the option to *assign* an output from elsewhere to here

Initialized Data Node
“int g 6”
Providing a third parameter enables the output link and sets an initial value for the data node

Assigned Data Node
“int h”
Linking output of another node makes the value and eventual output of the data node the same as the value of the input

Function Nodes:

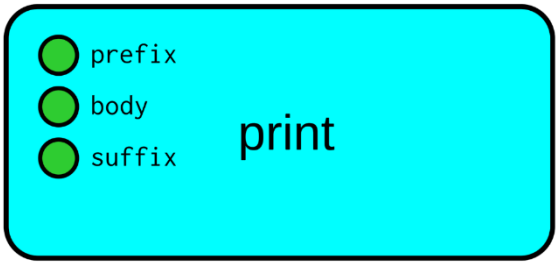
The following is a list of available function node types and an explanation of their input and output links.

Empty Function Node - “”

This is what every function node starts with. Until a proper name for the function is given such node has no input or output links as it does not really do anything.

Print Node - “print”

Used to show the value of node outputs to the output window on the right of the editor screen.

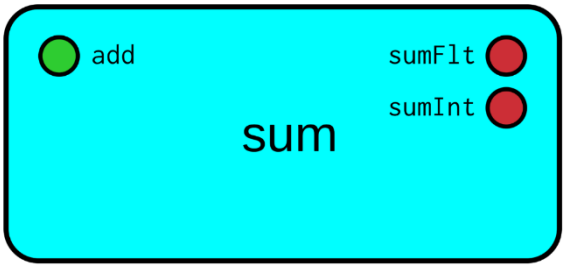


Inputs:

- prefix (optional): text or other value to put in front of body (only first linked connection is used)
- body: takes in all connections from other nodes and prints the value of the output links separated by a comma
- suffix (optional): text or other value to put at the end of the end of the body

Summation Node - “sum”

Used to add multiple numerical values together and provide the user with both a floating point and integer variant of the summation operation.



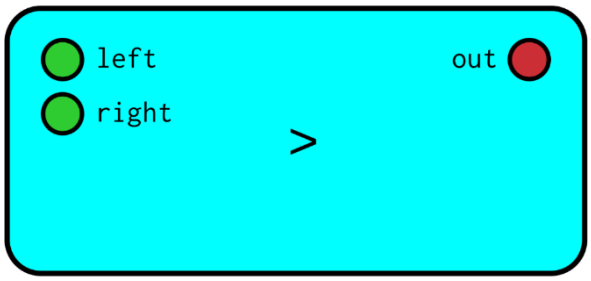
Input:

- add: one or more numerical value to be mathematically added together

Outputs:

- sumFlt: float variant of the summation result of all values connected to the input link
- sumInt: (sumFlt) but casted to an int, effectively the floor of sumFlt

isGreaterThan Node - ">"



Used to check if one value is bigger (numerically) than the other.

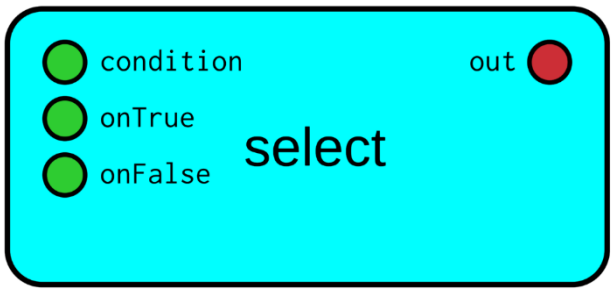
Inputs:

- left: the numerical value that will be placed at the left of the > operator
- right: numerical value that will be placed at the right side of the > operator

Output:

- out: a boolean value representing the outcome of the operator >. True if the left value is greater than the right one, False if not.

Selection Node - "select"



Used to imitate an if-else statement from a normal language such as C++.

Inputs:

- condition: a boolean value used to determine which input to pass on to the output
- onTrue: value that will be passed on in case the condition input is true
- onFalse: value that will be passed on in case the condition input is false

Output:

- out: the value that was selected based on the condition value provided

Example Problem and Visual Code Mix Solution

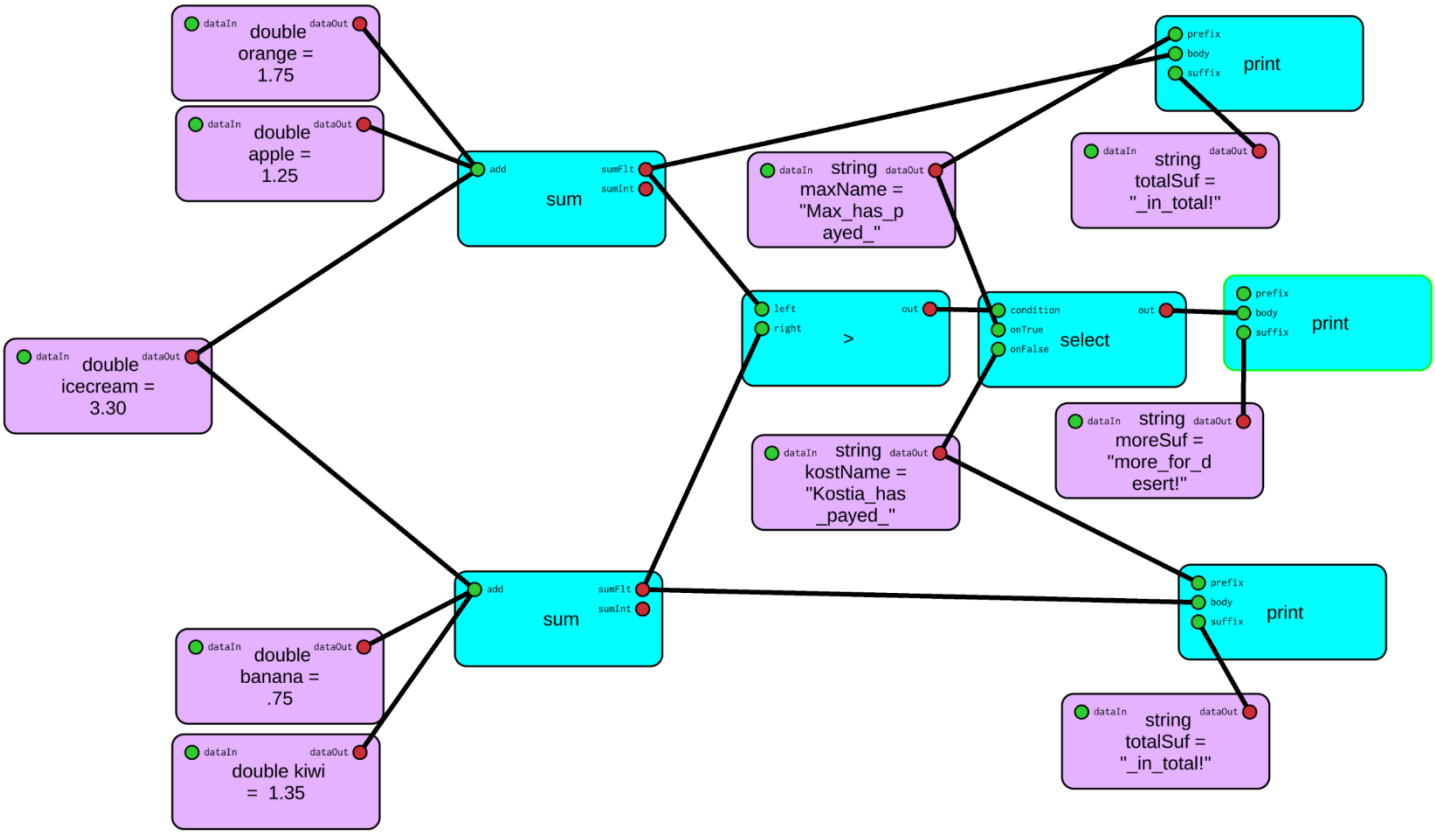
Problem:

Max and Kostia are both fruit lovers and go to the market to get their favorite fruits. The fruits prices at the market are marked as follows, orange - $1.75, apple - $1.25, banana - $.75 and kiwi - $1.35. Max likes oranges and apples while Kostia plans to take a banana and a kiwi. Both Max and Kostia also purchase chocolate ice cream for $3.30. Find who payed more for their deserts and also how much each person payed.

Solution:

[Next Page]

Node Canvas



Output of Above Node Graph:

```
Welcome to Visual Code Mix!
Author: Kostia Makrasnov

Kostia_has_payed_5.4_in_total!

Max_has_payed_6.3_in_total!

Max_has_payed_more_for_desert!
```

Limitations and Future Work:

Some known bugs exist when changing node designations (different functions the same link names). Also, upon one of the interconnected data nodes there can be linking problems upon next run of the program.

The language is not really Turing complete due to the absence of loops; this could be implemented with yet another function or restructuring how the execution of the node graph is handled. Python and C++ nodes turned out to be harder to complete than expected and thus do not exist as usable components of the language. As it currently stands the visual scripting language is a proof of concept, but with a few more days of development it could have become a great demo for a programming puzzle/learning game.