Final script to parse

From real R package - Bonne lecture !!!!

Script description

Rational

When comparing gene expression matrices from different experiments, gene symbols from each matrix often differs and finding a correspondance is sometimes tedious.

This package allow thourough identification of unique NCBI.Gene.ID (also names EntrezID) for each genes which allow the user to identify corresponding genes of two gene expression matrices.

User entry

- **GENES**: a list of genes symbols: BRCA1, TP53, NEF4, PEN2... is given as first entry.
- **HGNC**: a database file containing 4 columns.
  - **Approved symbols** : current notation for gene symbols (ex: BRCA1)
  - **Previous symbols**: old notation for gene symbols ()
  - **Synonyms**: list of symbol synonyms
  - **NCBI.Gene.ID**: universal gene identifyer number (less subject to updates)

##

	Approved.symbol	Previous.symbols	Synonyms	NCBI.Gene.ID
2756	BRCA1		RNF53, BRCC1, PPP1R53, FANCS	672
41984	TP53		p53, LFS1	7157
28823	PRPH	NEF4	PRPH1	5630
28780	PROB1	C5orf65		389333
28996	PSENEN		PEN2	55851

- **c**: number of cores to use for the computation.

How it works:

- The script starts with the correspondance between GENE and HGNC$Approved_symbols.
- If some genes match, it create a variable call **matched** containg the corresponding NCBI.Gene.ID and NA for other genes.
- Then it tries to find the ids of genes that were **not found** (subset of **matched** containing NA values) in the next HGNC column (previous symbols .. etc).
- Updates **matched** NA values with new match found and repeat previous step
- Finally when all resources have been checked (approved symbols, previous symbols, synonyms and other..) it returns the output describe above.

Script outputs

The goal is to compare successively the first user script entry (R variable: genes) to columns 1,2, and 3 of the database to find a match and retreive the 4th column value Entrez id.

- **aliases** is a 2 column table containing:
  - **Symbols** (GENEs changed with approved symbol)
  - **entrezID** corresponding entrez id if found or NA if no match were found for the gene symbol

Output for this example query BRCA1, TP53, NEF4, PEN2, and a mispelled.symbol is:

Symbols	entrezID
BRCA1	672
TP53	7157
PRPH	5630
PSENEN	55851
misspelled.symbol	NA

- As you can see Symbols corresponds to Approved symbols and entrezID to NCBI.Gene.ID.

	Approved.symbol	Previous.symbols	Synonyms	NCBI.Gene.ID
2756	BRCA1		RNF53, BRCC1, PPP1R53, FANCS	672
41984	TP53		p53, LFS1	7157
28823	PRPH	NEF4	PRPH1	5630
28780	PROB1	C5orf65		389333
28996	PSENEN		PEN2	55851

- BRCA1 and TP53 matched on approved symbols
- NEF4 matched on previous symbols
- PEN2 matched on synonyms
- Also the "misspelled.symbol" that was not found are kept unchange and show NA for entrezID.

Script decomposition and parsing

1. Function statement: function()

Script: line 1

```
convert_symbols <- function(symbols, HGNC, c = 1) { ....
```

How to

- A function should extract the name of the function.
- A function should extract the 3 argument of the function.
  - **Symbols**
  - **HGNC**
  - **c**

Resulting structures

NOTE:

- A good way to parse the script could be to check if all parenthesis and brackets [({})] to know if the command is complete.
- Knowing this keep in mind that, function statement should be evaluated even if { is not closed..

2. Checks HGNC file structure: if(), ifelse(), stop()

For the script to run, a properly formated HGNC file must be provided, if not, the script stops.

Script: line 2 to 13

```
expected <- c("Approved.symbol", "Previous.symbols", "Synonyms", "NCBI.Gene.ID")
  if (dim(HGNC)[2] != 4) {
    stop("HGNC correspondence table was not load properly...")
  } else if (!identical(colnames(HGNC), expected)) {
    stop("HGNC correspondence table was not load properly...")
  }
```

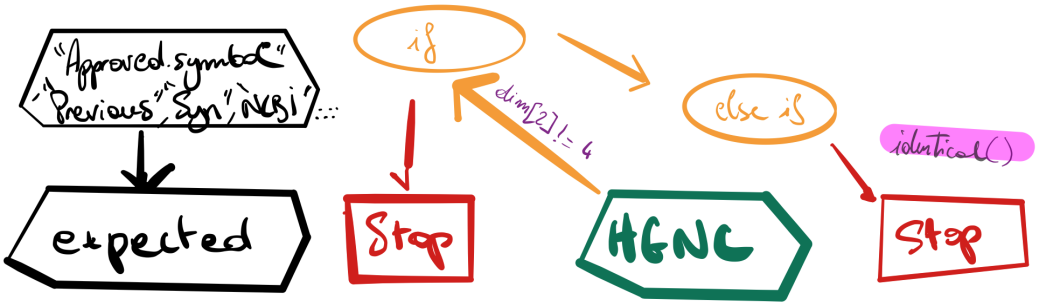
How to

- A function should recognise if() else() and also stop() functions, more generally all functions...
- A simple function should parse the text input c(“”,“”,“”)
- For the if and else statement, nodes should be created.
- In this case
  - if
  - else if

You will definitly need a color code:

- input
- test (if else ifelse)
- R commands
- exits (stop, return}

Resulting structures



Notes:

- With this color code you easily see the variables and the instances.
- Green input, red outputs (with type stop), you see the yellow if statements. On the first edge, from **HGNC** to the **if** node you have the if condition colored in purple as it is a R command.
- I think creating a if node as shown here is necessary, just mind that if multiple if conditions are present, you should disociate them .. something like if.line19 vs if.l25 to avoid having only one if node to which all the condition link to.
- Another interesting thing here is the undelined **indentical()** statement on the edge that arise from the **ifelse** node. Identical statement takes 2 arguments identical(a,b) returning T or F. I don't think we can link a and b to identical (link to link) so either string format or some unique id reference.
- For now lets ignore message(), print(), cat() options, put a good idea could be to let the user choose whether or not he wants to display these functions.. Similarly user could also chose a range of lines he don't want to analyse (complex or unrelated code).

3a. Accessors

Script: line 15 (part HGNC$NCBI.Gene.ID and HGNC$Approved.symbol)

```
HGNC$NCBI.Gene.ID
HGNC$Approved.symbol
```

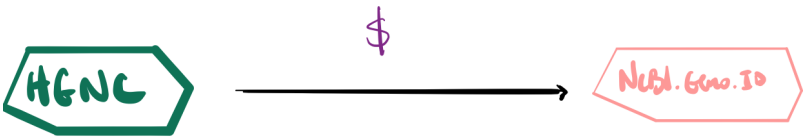
How to

- You can see that we call $ **NCBI.Gene.ID** on **HGNC** (*HGNC$NCBI.Gene.ID*). This is actually saying that we want a sub component of HGNC called NCBI.Gene.ID. So we need a function that looks for these **parent $ child** and creates a parent, child and direct connection between the two. Same goes for @. There can be many of them in one command Structure@variable$column.

We can add a new color in the code for accessors, and indices (see 4)

- $ - @ - and indices (sub items)

Resulting structures



Notes

- We find our colors again, green purple and the new one pink for subitems.

3b. Indices: Index tags

Lets say we have variable

- a = c(6,7,8)

We can use indices to acces parts of the variable.

- a[2] is equal to 7.

Script: line 15 (part HGNC$NCBI.Gene.ID[...])

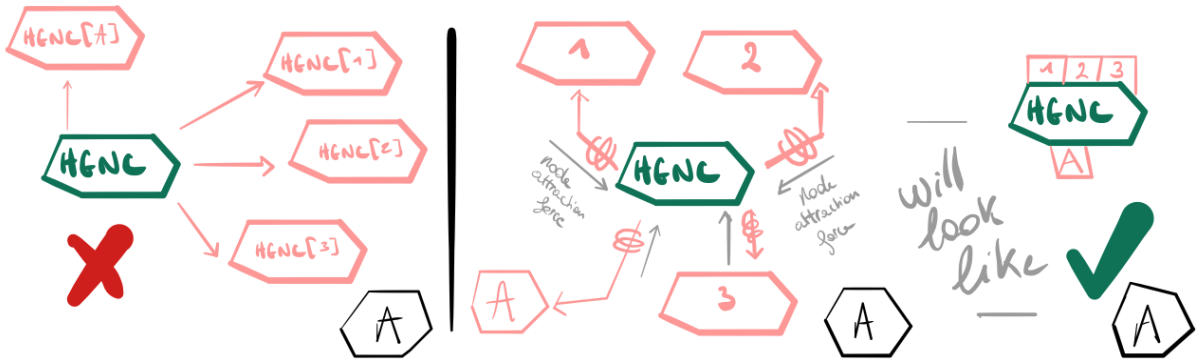
```
HGNC$NCBI.Gene.ID[somethinghere]
```

How to

Ok, so now it gets real. This is tricky.

- as you can see, we are subsetting HGNC[...] so this is a first challenge 2 solution for the resulting structures,
- Either create a new node **HGNC[...]** that will be comming from **HGNC** node. Not very practical because indexing is something that is really common so you can easily imagine that the graph will be crowded with nodes that are almost identical ...
- Or, create a new child node of **HGNC** with only the indice inside. This solution can take advantage of the visualization tool to strengthen the link between nodes. This leads to a visual trick where HGNC and its child node are very close to each other.

Resulting structures



Notes

- This is to verify on the cytoscape app. Anyway even if individual links cannot be tighten, extracting the id seems a good idea.
- The indices nodes are additionnal to the variable nodes... see unrelated A variable that is indexed from HGNC

3b. match ()

This part is about finding the correspondance between the 'user-provided' symbols and the 1st columns of HGNC (approved symbols) Match is a popular function in R where indexes are returned.

For example:

- a= c(1); b= c(3,4,2,1); result = match(a,b)

Result will have a value of 4. (1 is the 4th number in the second argument). Then we can subset **b** with **results** as an index:

- b[result]

Script: line 15 (part match())

```
matched <- HGNC$NCBI.Gene.ID[match(tolower(genes), tolower(HGNC$Approved.symbol))]
```

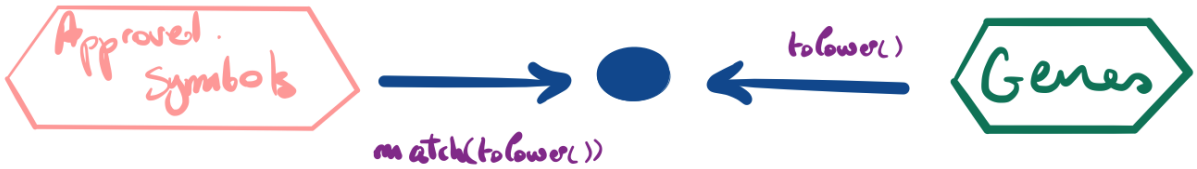
How to

- A function should work like your if_handler function and create a new node corresponding to IDs resulting from match. This could then be used as an index tag (described above)

Aaaaaand a new color code:

- match() statements

Resulting structures



Notes

- Here additionnal function are nested in the match function - match(tolower()).. (purple code).
- The blue node representing match node should be named and unique for further use (match for instance)

4. Object initialisation and return

This is where objects gets created (dataframe initialisation, matrix initialization, etc) and where the script/function ends (return). In the script there are several places were the scripts can return the **aliases** variable and stop and most of them rely on **if statements** .

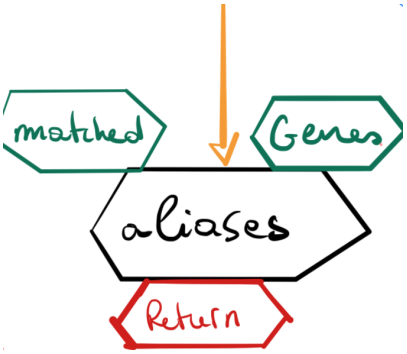
Script: lines 306 to 308

```
aliases <- data.frame(Symbols = genes, entrezID = as.numeric(matched))
return(aliases)
```

How to

- indices tags with specific color codes seems appropriate. (green to initialize and red to exit script)

Resulting structures



Notes

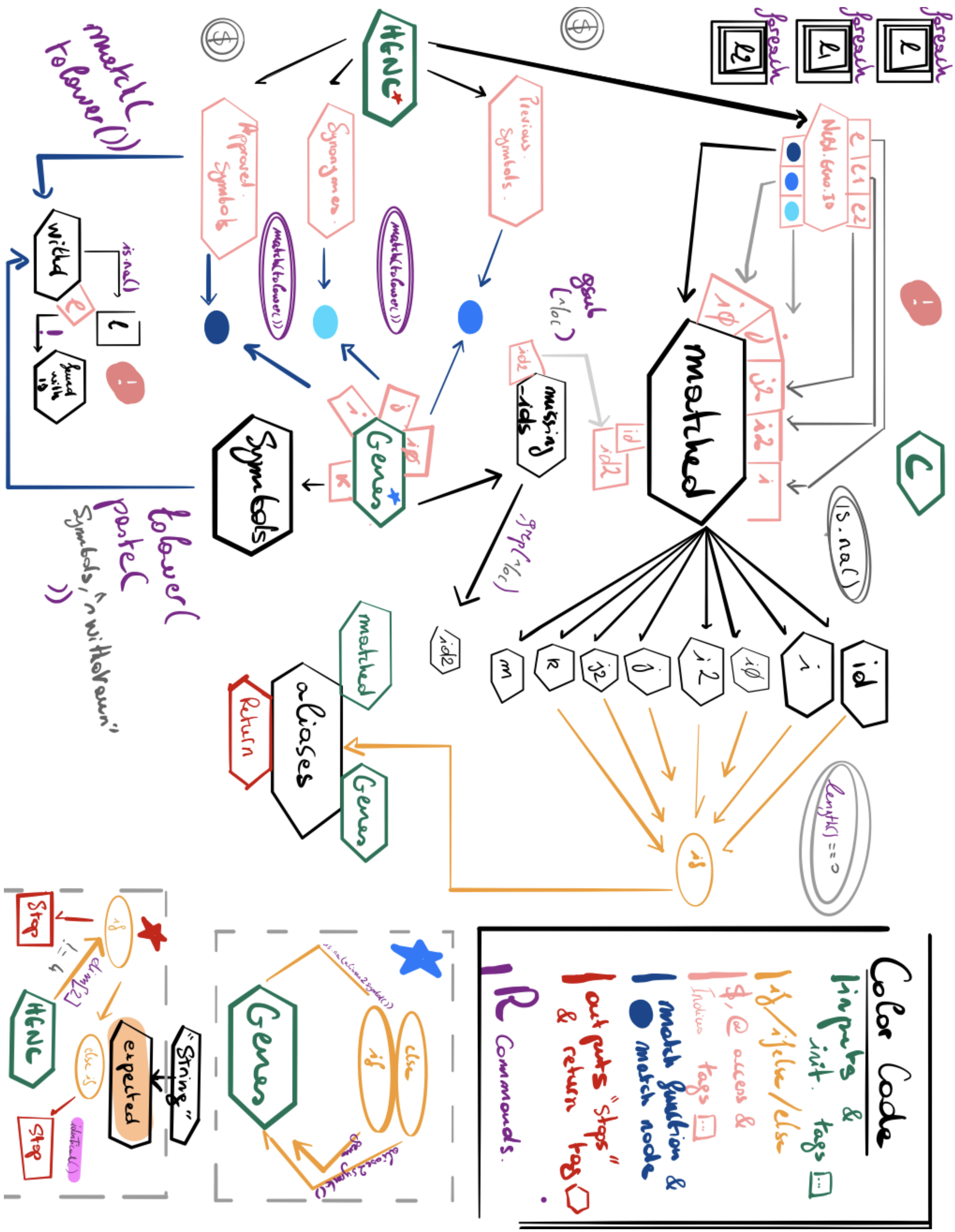
- aliases is initialized with the two green tagged variables **matched** and **genes**
- aliases has a **return** exit tag

Conclusion

- Here you have the main function to handle a regular Rscript.
- If we apply these 4 points to all the script we obtain the following graph.
- We spoke a very little of blackboxes but they are represented anyway in the final graph to have an idea of how the play a role. They correspond to the 3 foreach (parallel loops in the code 45-80, 120-162, 202-244) resulting in the creation of the variables l, l1, l2 (upper left of the global script, used as indices for NCBI.gene.ID)

You should start to catalog complex functions to avoid parsing and send to blackbox. We just need the function name and the output.

- There are some if(message('blabla')) chunk that can be ignored as well, but this meens that you hav to look into the if statement to see wether to display it or not..


Color Code
inputs & init. tags
if/ifelse/else
$, @ access & Indices tags
match function & match node
outputs "stops" & return tags
R Commands
HGNC
Previous Symbols
Synonymes
Approved Symbols
match(tolower())
Genes
Symbols
matched
missing_ids
aliases
Return
withd
found with id
foreach
tolower(paste(Symbols, "withdrawn"))
Stop
expected
"String"