Importing and managing a large data set in *fizz*

Jean-Louis Villecroze

jlv@f1zz.org @CocoaGeek

September 1, 2018

Abstract

This article[1] looks at how *fizz* (version 0.3 and up) deals with importing a *large* set of data for inferences, and explores possible optimizations for faster loading and *statement* retrieval time. In it, we will be using the *Escherichia coli* (abbreviated as *E. coli*) genome retreived from the `ecogene.org` website. We will also build the *procedural knowledge* needed to find out how many genes contains the famous `GATTACA` sequence in the *E. coli* genome. The completed application can be found in the `etc/articles/e.coli` folder of the *fizz* 's distribution.

Prerequisite

A basic understanding of the concepts behind *fizz* is expected from the reader of this article. It is suggested to read the introductary article *Building a simple stock prices monitor with fizz* [2] first or at least read section two to four of the *user manual* for an overview of the language and runtime usage.

Importing the raw DNA sequences

The entirety of the *E. coli* genome can be downloaded from http://www.ecogene.org/ in the `FASTA` text format where the whole genomic DNA sequence is split over a large number of lines (of a known maximum length). We can use the `import.txt` *command*, to instruct *fizz* to process that file, and generate a *statement* for each line that it contains.

To get started, let's try the *command* by extracting the first 10 lines from the file. We will first use the `spy` *command* to get *fizz* to show us the generated *statements*:

```
$ ./fizz.x64
fizz 0.3.0-X (20180519.2228) [x64|8|w|l]

?- /spy(append,line.f)
spy : observing line.f
?- /import.txt("./etc/data/U00096.3.txt",line.f,1,10)
import.txt : 10 lines read in 0.000s.
spy : S line.f(0, "AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTGTCTGATAGCAGCTTCTG") := 1.00 (700.000000)
spy : S line.f(1, "AACTGGTTACCTGCCGTGAGTAAATTAAAATTTTATTGACTTAGGTCACTAAATACTTTAACCAATATAGGCATA") := 1.00 (700.000000)
spy : S line.f(2, "GCGCACAGACAGATAAAAATTACAGAGTACACAACATCCATGAAACGCATTAGCACCACCATTACCACCACCATC") := 1.00 (700.000000)
spy : S line.f(3, "ACCATTACCACAGGTAACGGTGCGGGCTGACGCGTACAGGAAACACAGAAAAAAGCCCGCACCTGACAGTGCGGG") := 1.00 (700.000000)
spy : S line.f(4, "CTTTTTTTTTCGACCAAAGGTAACGAGGTAACAACCATGCGAGTGTTGAAGTTCGGCGGTACATCAGTGGCAAAT") := 1.00 (700.000000)
spy : S line.f(5, "GCAGAACGTTTTCTGCGTGTTGCCGATATTCTGGAAAGCAATGCCAGGCAGGGGCAGGTGGCCACCGTCCTCTCT") := 1.00 (700.000000)
spy : S line.f(6, "GCCCCCGCCAAAATCACCAACCACCTGGTGGCGATGATTGAAAAAACCATTAGCGGCCAGGATGCTTTACCCAAT") := 1.00 (700.000000)
spy : S line.f(7, "ATCAGCGATGCCGAACGTATTTTTGCCGAACTTTTGACGGGACTCGCCGCCGCCCAGCCGGGGTTCCCGCTGGCG") := 1.00 (700.000000)
spy : S line.f(8, "CAATTGAAAACTTTCGTCGATCAGGAATTTGCCCAAATAAAACATGTCCTGCATGGCATTAGTTTGTTGGGGCAG") := 1.00 (700.000000)
spy : S line.f(9, "TGCCCGGATAGCATCAACGCTGCGCTGATTTGCCGTGGCGAGAAAATGTCGATCGCCATTATGGCCGGCGTATTA") := 1.00 (700.000000)
```

Since the `import.txt` primitive doesn't `assert` the *statements* it generates (but `declare` them), we need to create an *elemental* object which will be doing that for us. For that, we create a *fizz* file called `import.fizz` and set up a single *procedural knowledge* definition which we'll call `import.frag`. In it we will set up a single `prototype` which will be *triggered* by any `line.f` *statements*, and for now we will just output the line identifier to the console:

```
1 import.frag {
2
3     () :- @line.f(:i,:s), console.puts(:i);
4
5 }
```

[1] Thanks to Robert Wasmann (@retrospasm) for providing feedback and reviewing this document.

[2] available on the web site

We will now restart *fizz* and load the file we just created. Next use the `import.txt` *command* as we did just above:

```
$ ./fizz.x64 ./etc/articles/e.coli/import.fizz
fizz 0.3.0-X (20180519.2228) [x64|8|w|l]

load : loading ./etc/articles/e.coli/import.fizz ...
load : loaded  ./etc/articles/e.coli/import.fizz in 0.001s
?- /import.txt("./etc/data/U00096.3.txt",line.f,1,10)
import.txt : 10 lines read in 0.000s.
0
1
2
3
4
5
6
7
8
9
```

Now, since the *statements* that are being *declared* (broadcast in the *substrate*), and we would like them to be stored in memory, we need to modify the *procedural knowledge* to *assert* them using the `assert` primitive:

```
1 import.frag {
2
3     () :- @line.f(:i,:s), assert(frag(:i,:s)), console.puts(:i);
4
5 }
```

Re-running our code just as above won't yield any visible difference; however, if we run the console *command* `/stats` we can see that there are now ten *statements* in the *substrate*:

```
$ ./fizz.x64 ./etc/articles/e.coli/import.fizz
fizz 0.3.0-X (20180519.2228) [x64|8|w|l]

load : loading ./etc/articles/e.coli/import.fizz ...
load : loaded  ./etc/articles/e.coli/import.fizz in 0.001s
?- /import.txt("./etc/data/U00096.3.txt",line.f,1,10)
import.txt : 10 lines read in 0.000s.
0
1
2
3
4
5
6
7
8
9
?- /stats
stats : e:7 k:3 s:10 p:1 u:15.33 t:1 q:0 r:0 z:29
```

We can also call the `list` console *command* to check that a new *elemental* has been created by *fizz* to collect all the *statements* we asserted:

```
?- /list
list : 0cd38dac-93cd-dd42-d998-af71e107397b MRKCBFSolver        import
list : f2850094-0398-0942-45ad-0134a4f86744 MRKCLettered        frag
list : 2 elementals listed in 0.000s
```

In order to avoid having to re-import the whole genome each time, we're going to save the *elemental* to disk with the `save` console *command*. We will make sure to indicate that we only want to save the `frag` *elemental*, since that's the end result of the import:

```
?- /save("./etc/articles/e.coli/frag.fizz",frag)
save : completed in 0.001s.
```

The contents of `frag.fizz` we just created shouldn't be much of a surprise:

```
frag {

    (0, "AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTGTCTGATAGCAGCTTCTG");
    (1, "AACTGGTTACCTGCCGTGAGTAAATTAAAATTTTATTGACTTAGGTCACTAAATACTTTAACCAATATAGGCATA");
    (2, "GCGCACAGACAGATAAAAATTACAGAGTACACAACATCCATGAAACGCATTAGCACCACCATTACCACCACCATC");
    (3, "ACCATTACCACAGGTAACGGTGCGGGCTGACGCGTACAGGAAACACAGAAAAAAGCCCGCACCTGACAGTGCGGG");
    (4, "CTTTTTTTTTCGACCAAAGGTAACGAGGTAACAACCATGCGAGTGTTGAAGTTCGGCGGTACATCAGTGGCAAAT");
    (5, "GCAGAACGTTTTCTGCGTGTTGCCGATATTCTGGAAAGCAATGCCAGGCAGGGGCAGGTGGCCACCGTCCTCTCT");
    (6, "GCCCCCGCCAAAATCACCAACCACCTGGTGGCGATGATTGAAAAAACCATTAGCGGCCAGGATGCTTTACCCAAT");
    (7, "ATCAGCGATGCCGAACGTATTTTTGCCGAACTTTTGACGGGACTCGCCGCCGCCCAGCCGGGGTTCCCGCTGGCG");
    (8, "CAATTGAAAACTTTCGTCGATCAGGAATTTGCCCAAATAAAACATGTCCTGCATGGCATTAGTTTGTTGGGGCAG");
    (9, "TGCCCGGATAGCATCAACGCTGCGCTGATTTGCCGTGGCGAGAAAATGTCGATCGCCATTATGGCCGGCGTATTA");

}
```

Now, we could import the entirety of the *E. coli* genome this way; however, this file is rather large (61890 lines) and so the runtime cost of *asserting* each of the 61889 *statements* is prohibitively high. Instead, we are going to replace the *primitive* `assert` by `bundle` as it allows for *statements* to be *bundled* into a single *procedural knowledge* which will be injected on the *substrate*, thus creating a new *elemental* to handle it.

Let's modify `import.fizz` to use that `primitive` and instruct it to split all the *statements* into a bundle of 1024 *statements*:

```
import.frag {

    () :- @line.f(:i,:s), bundle(frag(:i,:s),1,{},1024), hush;

}
```

Note that at the end of the *prototype* we have added a call to the *primitive* `hush`. This will make sure that the completion of the inference will not result in thousands of `import.frag` *statements* being published, since this is unnecessary. This will give us slightly better performance. If we now restart *fizz* , and import the first 2048 lines, we will get 2048 *statements* split over 2 *elementals*:

```
$ ./fizz.x64 ./etc/articles/e.coli/import.fizz
fizz 0.3.0-X (20180519.2228) [x64|8|w|l]

load : loading ./etc/articles/e.coli/import.fizz ...
load : loaded  ./etc/articles/e.coli/import.fizz in 0.001s
?- /import.txt("./etc/data/U00096.3.txt",line.f,1,2048)
import.txt : 1000 lines parsed ...
import.txt : 2000 lines parsed ...
import.txt : 2048 lines read in 0.013s.
?- /list
list : 352b248c-3dd2-444d-2593-aaee3921226e MRKCBFSolver        import
list : 18525ad5-a1b1-8a41-cfac-c8c78c9ca790 MRKCLettered        frag
list : cb705a4d-3092-7942-658f-aeee4cba1bdc MRKCLettered        frag
list : 3 elementals listed in 0.000s
?- /stats
stats : e:8 k:4 s:2048 p:1 u:41.93 t:0 q:0 r:0 z:4096
```

If we were to import the whole file, then 61 *elementals* will be created on the *substrate* to handle them, which is a little excessive, so we are going to change the bundle size we provide to the `bundle` *primitive* to 3072, and we will then get 22 *elementals*:

```
$ ./fizz.x64 ./etc/articles/e.coli/import.fizz
fizz 0.3.0-X (20180519.2228) [x64|8|w|l]

load : loading ./etc/articles/e.coli/import.fizz ...
load : loaded  ./etc/articles/e.coli/import.fizz in 0.001s
?- /import.txt("./etc/data/U00096.3.txt",line.f,1)
import.txt : 1000 lines parsed ...
import.txt : 2000 lines parsed ...
...
import.txt : 61000 lines parsed ...
import.txt : 61889 lines read in 0.397s.
?- /stats
stats : e:20 k:16 s:43008 p:1 u:15.94 t:13 q:0 r:0 z:105989
?- /stats
stats : e:24 k:20 s:55296 p:1 u:18.42 t:15 q:0 r:0 z:119389
?- /stats
stats : e:27 k:23 s:61889 p:1 u:21.28 t:880 q:0 r:0 z:123778
?- /stats
stats : e:27 k:23 s:61889 p:1 u:22.83 t:0 q:0 r:0 z:123778
?- /list
list : b073f751-abd8-f14f-1299-b7b45d05a1c8 MRKCBFSolver        import
list : bf02b90a-30ce-2a41-e283-fbd96beccd7f MRKCLettered        frag
list : bb4759d0-69fa-b348-3abf-19dbb0a161bb MRKCLettered        frag
list : b0489927-a51d-924b-0d87-ed12179ca732 MRKCLettered        frag
list : e427c9e6-792c-5f4b-d081-1db31eaf72a8 MRKCLettered        frag
list : e0f3714c-0811-8c4a-0994-46c1fec1e470 MRKCLettered        frag
list : 23891829-5335-8b42-0e90-ba77cb5e6d7f MRKCLettered        frag
list : 6d9660c3-ce66-104c-8cbf-58366b5f9996 MRKCLettered        frag
list : 9046cd34-107e-2e48-9aae-4226b0e582ce MRKCLettered        frag
list : df490e35-0b0a-a94b-e788-0e3adba67a27 MRKCLettered        frag
list : ae2b2e9b-1ee5-2a46-a197-e2a3b671c468 MRKCLettered        frag
list : 12e7b881-04c2-fb4c-5a9f-51da1817343d MRKCLettered        frag
list : 1852d124-a409-0344-f3b1-831a3d2e38e7 MRKCLettered        frag
list : 68811f8d-0df5-bc42-70aa-1dd5fe8e74ec MRKCLettered        frag
list : bd536caa-54d4-424d-7cbf-3e810274044a MRKCLettered        frag
list : e1e56c90-d943-374e-f7a7-df6146cc558a MRKCLettered        frag
list : 0399e71b-68dc-1c4c-ab9d-248fdd3e6511 MRKCLettered        frag
list : 6df47184-0aa1-574d-ab8d-1632255610e1 MRKCLettered        frag
list : bdcb5bf3-9295-3846-fa9b-01c592dcdab2 MRKCLettered        frag
list : 8c0aa796-62b7-5a41-1cbe-b615ef83128f MRKCLettered        frag
list : e4514816-922b-ad45-20a8-72416bd5fa6f MRKCLettered        frag
list : 7ddd475e-37d2-8445-98a3-3e5f11f107d1 MRKCLettered        frag
list : 22 elementals listed in 0.007s
```

Finally, we will save the *statements* into a *fizz* file, so that we can later reload it::

```
?- /save("./etc/articles/e.coli/frags.fizz",frag)
save : completed in 0.429s.
```

Since it's a rather large file (5682972 bytes) for *fizz* to parse, don't expect the loading time to be stellar:

```
$ ./fizz.x64 ./etc/articles/e.coli/frags.fizz
fizz 0.3.0-X (20180519.2228) [x64|8|w|l]

load : loading ./etc/articles/e.coli/frags.fizz ...
load : loaded  ./etc/articles/e.coli/frags.fizz in 7.244s
?- /list
list : dea95f02-9b8b-9e4f-54b2-ccc3dd755e51 MRKCLettered        frag
list : 5a05e004-2de2-6642-87b0-fd08a661daa8 MRKCLettered        frag
list : 0f7e1975-b14e-3449-6488-fcbc538600e0 MRKCLettered        frag
list : 065b74c2-00a8-d74d-72bb-19e2cba18c0c MRKCLettered        frag
list : 42b6e269-62e5-5b4f-7582-9659cad1cc48 MRKCLettered        frag
list : 91859291-b806-c849-3eb8-a83a39bffddb MRKCLettered        frag
list : 0a470c7c-3b9a-f142-b789-5a4c1d651c17 MRKCLettered        frag
list : 97b6a24d-fd51-4649-aead-e0ee7f6b1f14 MRKCLettered        frag
list : 2693353f-d98d-0b4d-1cad-ef003067efd3 MRKCLettered        frag
list : 0fe0fcb0-fd1f-6e45-3793-849650ac01be MRKCLettered        frag
list : ec6802b4-3a87-c94a-b193-bfbc32fdb1a2 MRKCLettered        frag
list : 48f6371a-0b1b-e24d-faa7-a3133d58c63a MRKCLettered        frag
list : c4fac853-80e3-2240-5a86-66da3ed658ca MRKCLettered        frag
list : a79b17d2-e1d3-de42-10a9-d084983388fa MRKCLettered        frag
list : 2766422f-f4ee-5e4a-b096-71ebb37e2455 MRKCLettered        frag
list : fc76a5aa-4379-6447-6a81-8c7542a95da2 MRKCLettered        frag
list : 38a4a6e0-db1b-d941-be9a-dd6de8bc2218 MRKCLettered        frag
list : 0bcff970-6738-cb4c-31bd-644e3b266be8 MRKCLettered        frag
list : 1edcd74b-0513-e84a-5690-d7e378b60aff MRKCLettered        frag
list : 1c9e1128-2796-a44e-2cae-9acf1e5fa8e9 MRKCLettered        frag
list : d329fe7f-c742-4a44-0ab4-846e3ee59d9d MRKCLettered        frag
```

```
list : 21 elementals listed in 0.002s
?- /stats
stats : e:26 k:22 s:61889 p:0 u:25.83 t:2 q:0 r:0 z:0
```

As we are not actually transforming the *statements* we read from the text file, we can use a special mode of the `import.txt` *command* to bundle the *statements* over multiple *elemental* objects like we did above with the advantage of a much better runtime performance since we won't be executing any inference for each *statement*.

This is done by adding a *list* of flags (*symbols*) as the last *term* of the call. Here we will ask the `import.txt` *command* to bundle the *statements* by 3072, and to spawn a new *elemental* for each bundle:

```
$ ./fizz.x64
fizz 0.3.0-X (20180519.2228) [x64|8|w|l]

?- /import.txt("./etc/data/U00096.3.txt",frag,1,3072,[loop,bundle,spawn])
import.txt : 1000 lines parsed ...
import.txt : 2000 lines parsed ...
...
import.txt : 61000 lines parsed ...
import.txt : 61889 lines read in 0.291s.
?- /stats
stats : e:26 k:22 s:61889 p:0 u:11.28 t:0 q:0 r:0 z:0
?- /list
list : 52383e12-0034-954e-d48c-3e0d15952362 MRKCLettered        frag
list : 7c00f988-3b0d-5249-9a83-14230680cfcb MRKCLettered        frag
list : 65e83f8a-17d1-4c40-9ca8-28f2339a6639 MRKCLettered        frag
list : 97498a7a-7f32-df4f-cdb6-06d866c05a8a MRKCLettered        frag
list : d94d31d9-cf81-cf44-94a8-c4b21743675b MRKCLettered        frag
list : 46063da9-44ea-2042-20b6-aaaf753e08c0 MRKCLettered        frag
list : b4520421-c09a-384c-349c-b49727869a01 MRKCLettered        frag
list : d7335353-a3bd-c14f-9cb1-c596c6d6aee0 MRKCLettered        frag
list : 2169255d-6855-884e-d59d-7808066ca3fb MRKCLettered        frag
list : 378798f5-8771-ac4a-4bb4-1d4bfadccfd4 MRKCLettered        frag
list : d4916964-bfd4-4d42-c8b9-6bea18c8fc1a MRKCLettered        frag
list : ab9982e2-27c1-5b4e-a7b3-b74f6b5bcc0f MRKCLettered        frag
list : 24100611-4edf-474e-4288-9c14fbbda099 MRKCLettered        frag
list : 75f4a47b-e770-3745-3784-83a906f32a58 MRKCLettered        frag
list : 56b4c9d1-50a9-3544-f6ac-3c498bf43515 MRKCLettered        frag
list : c53725de-bf65-ad46-8795-9c0dbd44e15b MRKCLettered        frag
list : f65b0bdc-4490-e049-3698-dbfdb616299c MRKCLettered        frag
list : bdeb4d21-e025-9543-8d8d-e5725331518a MRKCLettered        frag
list : 2cbb5aef-d545-194f-40a2-9cb47515d82e MRKCLettered        frag
list : 426917ff-d5a1-d544-db87-25868c0b202a MRKCLettered        frag
list : c42bf3d3-77ba-c648-31af-c88128dedcb3 MRKCLettered        frag
list : 21 elementals listed in 0.001s
```

Importing the genes descriptions

The second data set we are going to import into *fizz* is a collection of all the identified genes for the *E. coli* genome. The set is in a `CSV` formatted document (with tabulation as separator), so we will be using the *command* `/import.csv`. Unlike the DNA data, we are going to have to do some transformation on each of the *statements* that will be extracted, which means it won't be able to make the import as fast as for the sequences. Hopefully there's only 4504 lines to be processed in that file.

Each of the lines from the `CSV` file contains the following 12 elements, some of which we will be ignoring:

`EG`	EcoGene Accession Number.
`ECK`	K-12 Gene Accession Number.
`Gene`	Primary Gene Name.
`Syn`	Alternate Gene Symbols.
`Type`	Genotype.
`Len`	Sequence length.
`Orientation`	Orientation (Clockwise, Counterclockwise).
`LeftEnd`	Genomic Address, left end.
`RightEnd`	Genomic Address, right end.
`Protein`	Protein description.
`Function`	Known function.
`Description`	Description.
`Comments`	Comments.

Let's start by adding to the `import.fizz` file we have already been using a new *procedural knowledge* definition with a single *prototype*, which when triggered by any *statements* published by the `import.csv` *command* will print to the gene's identifier to the console:

```
import.gene {

    () :-   @line.gene(:t1,:t2,:gene,:t4,:t5,:len,:t7,:leftend,:rightend,:t10,:t11,_,:comments),
            console.puts(:t1),
            hush;

}
```

We can then go ahead and test that by importing the first 10 lines from the file:

```
$ ./fizz.x64 ./etc/articles/e.coli/import.fizz
fizz 0.3.0-X (20180519.2228) [x64|8|w|l]

load : loading ./etc/articles/e.coli/import.fizz ...
load : loaded  ./etc/articles/e.coli/import.fizz in 0.007s
?- /import.csv("./etc/data/EcoData022718-235543.txt",line.gene,"\t",[],1,10)
import.csv : 10 lines read in 0.001s.
EG10001
EG10002
EG10003
EG10004
EG10005
EG10006
EG10007
EG10008
EG10009
EG10010
```

Because the `import.csv` *command* doesn't automatically convert *strings* into *symbols*, we are going to have to do so for all the *terms* that we wish to be handled as *symbols*. For that we will use the *primitive* `str.tosym` to convert the following *terms*: `EG`, `ECK`, `Type`, and `Orientation`:

```
import.gene {

    () :-   @line.gene(:t1,:t2,:gene,:t4,:t5,:len,:t7,:leftend,:rightend,:t10,:t11,_,:comments),
            str.tosym(:t1,:eg),
            str.tosym(:t2,:eck),
            str.tosym(:t5,:type),
            str.tosym(:t7,:ori),
            console.puts(:t1),
            hush;

}
```

Ideally, we would have like to also convert the gene identifier to a *symbol* (e.g. `hisM`). Unfortunately some of them contains unsuitable characters (e.g. `rhsE'`) so we will leave them as *strings*. Next, we will use *primitive*

`str.tokenize` to transform the fourth *term* (Alternate Gene Symbols) from a *string* to a *list* of *strings*, as that field uses a comma to separate the symbols:

```
import.gene {

    () :-   @line.gene(:t1,:t2,:gene,:t4,:t5,:len,:t7,:leftend,:rightend,:t10,:t11,_,:comments),
            str.tosym(:t1,:eg),
            str.tosym(:t2,:eck),
            str.tosym(:t5,:type),
            str.tosym(:t7,:ori),
            str.tokenize(:t4,",",:alts),
            console.puts(:t1,": ",:alts),
            hush;

}
```

If we run the import again, then we can make two observations which will improve the representation of the data:

```
?- /import.csv("./etc/data/EcoData022718-235543.txt",line.gene,"\t",[],1,10)
import.csv : 10 lines read in 0.001s.
EG10001: ["None"]
EG10002: ["chlJ"]
EG10003: ["None"]
EG10004: ["coaBC"]
EG10005: ["fpr", " fruF"]
EG10006: ["genF"]
EG10007: ["None"]
EG10008: ["hydH"]
EG10009: ["icd"]
EG10010: ["None"]
```

the first is that when the *string* is `"None"`, then we should be using an empty *list*. The second observation is that we should trim each of the *strings*, since we can see instances where extra space shows up (e.g. `" fruF"`). To that end, we're going to add two *procedural knowledge* definitions to our `import.fizz` file to perform that transformation. The first one, which we will call `clean.list`, handles the trimming of any *strings* in a *list*. It works recursively (like most things in *fizz*), and uses the *primitive* `str.trim` for the actual trimming of the *strings*:

```
clean.list {

    ([],[])                ^:- true;
    ([:e?[is.string]],[:f]) ^:- str.trim(:e,:f);
    ([:h|:t],[:h2|:t2])     :- #clean.list(:t,:t2), str.trim(:h,:h2);

}
```

As a reminder, the caret that you see on the first two *prototypes* indicates that if the *entry point* does unify, the solver shouldn't try to use any of the following prototypes. Now, the second bit of *procedural knowledge* we are going to add is the one we will be directly calling in `import.gene`. It simply either matchs `["None"]` to an empty *list*, or calls `clean.list`:

```
import.gene.clean {

    (["None"],[])  ^:- true;
    (:l,:cl)        :- #clean.list(:l,:cl);

}
```

We can now add the transformation of the fourth *term* to `import.gene`:

```
import.gene {

    () :-   @line.gene(:t1,:t2,:gene,:t4,:t5,:len,:t7,:leftend,:rightend,:t10,:t11,_,:comments),
            str.tosym(:t1,:eg),
            str.tosym(:t2,:eck),
            str.tosym(:t5,:type),
            str.tosym(:t7,:ori),
            str.tokenize(:t4,",",:t4.1), #import.gene.clean(:t4.1,:syn),
            console.puts(:t1,": ",:syn),
            hush;

}
```

If we now perform the test import again, then we get a much better result:

```
?- /import.csv("./etc/data/EcoData022718-235543.txt",line.gene,"\t",[],1,10)
import.csv : 10 lines read in 0.001s.
EG10001: []
EG10003: []
EG10002: ["chlJ"]
EG10004: ["coaBC"]
EG10005: ["fpr", "fruF"]
EG10007: []
EG10006: ["genF"]
EG10008: ["hydH"]
EG10010: []
EG10009: ["icd"]
```

Now, we will perform the same transformation for the 10th and 11th *terms*, but before that we're going to make a small addition to `import.gene.clean` to handle the fact that the *string* can have a value of `"Null"`, which we will handle like we did for `"None"`:

```
import.gene.clean {

    (["None"],[])  ^:- true;
    (["Null"],[])  ^:- true;
    (:l,:cl)        :- #clean.list(:l,:cl);

}

import.gene {

    () :-   @line.gene(:t1,:t2,:gene,:t4,:t5,:len,:t7,:leftend,:rightend,:t10,:t11,_,:comments),
            str.tosym(:t1,:eg),
            str.tosym(:t2,:eck),
            str.tosym(:t5,:type),
            str.tosym(:t7,:ori),
            str.tokenize(:t4,",",:t4.1), #import.gene.clean(:t4.1,:syn),
            str.tokenize(:t10,";",:t10.1), #import.gene.clean(:t10.1,:protein),
            str.tokenize(:t11,";",:t11.1), #import.gene.clean(:t11.1,:function),
            console.puts(:t1,": ",:function),
            hush;

}
```

Lastly, we just need to assert a statement for each gene we are importing. To speed things up we're going to use the `bundle` primitive like we did earlier with a bundle size of 2048:

```
import.gene {

    () :-   @line.gene(:t1,:t2,:gene,:t4,:t5,:len,:t7,:leftend,:rightend,:t10,:t11,_,:comments),
            str.tosym(:t1,:eg),
            str.tosym(:t2,:eck),
            str.tosym(:t5,:type),
            str.tosym(:t7,:ori),
            str.tokenize(:t4,",",:t4.1), #import.gene.clean(:t4.1,:syn),
            str.tokenize(:t10,";",:t10.1), #import.gene.clean(:t10.1,:protein),
            str.tokenize(:t11,";",:t11.1), #import.gene.clean(:t11.1,:function),
```

```
11              bundle(gene(:eg,:eck,:gene,:syn,:type,:len,:ori,:leftend,:rightend,:protein,:function,:comments),1,{},2048),
12              hush;
13
14  }
```

We are now ready to import the whole content of the file. Note that depending on the *runtime* settings, and your system's performance, the number of *elementals* that will be spawned may be different and they may not all have exactly 2048 *statements* in it. We will then save the *statements* into a *fizz* file:

```
?- /import.csv("./etc/data/EcoData022718-235543.txt",line.gene,"\t",[],1)
import.csv : 1000 lines parsed ...
import.csv : 2000 lines parsed ...
import.csv : 3000 lines parsed ...
import.csv : 4000 lines parsed ...
import.csv : 4504 lines read in 0.273s.
?- /list
list : f5ef9cec-8462-1e49-b08d-70889857cc03 MRKCBFSolver        clean.list
list : 40946431-403d-af4b-ddb2-297f0d869d40 MRKCLettered        gene
list : f001366a-680a-c540-b0a4-c7582d6d078d MRKCLettered        gene
list : fc5cfd50-1270-a540-3a80-d677f8e50eb7 MRKCLettered        gene
list : a65b3e1e-11ab-d845-44a1-f580bc4a37d0 MRKCLettered        gene
list : a958a4b2-17c1-f542-1892-4c7a748109a1 MRKCBFSolver        import.gene
list : 16269115-186e-c34b-178d-3ee7bcef64e6 MRKCBFSolver        import.gene.clean
list : a519801f-0745-e143-ab98-6cae94920398 MRKCBFSolver        import.frag
list : 8 elementals listed in 0.000s
?- /stats
stats : e:13 k:9 s:4504 p:8 u:23.02 t:18 q:37875 r:37875 z:4504
?- /save("./etc/articles/e.coli/genes.fizz",gene)
save : completed in 0.114s.
```

Optimizing loading time

Now that we have transformed both raw data files into the form of *factual knowledge* representation that *fizz* can easily manipulate, we are going to look at how to speed-up the loading time. Let's first assess the loading time by loading both files when starting *fizz* :

```
$ ./fizz.x64 ./etc/articles/e.coli/genes.fizz ./etc/articles/e.coli/frags.fizz
fizz 0.3.0-X (20180519.2228) [x64|8|w|l]

load : loading ./etc/articles/e.coli/genes.fizz ...
load : loading ./etc/articles/e.coli/frags.fizz ...
load : loaded  ./etc/articles/e.coli/genes.fizz in 2.238s
load : loaded  ./etc/articles/e.coli/frags.fizz in 5.493s
load : loading completed in 5.493s
?- /stats
stats : e:30 k:26 s:66393 p:0 u:33.16 t:9 q:0 r:0 z:0
```

To speed-up the loading time we are going to take advantage of a native feature of *fizz* - the *runtime* will try to load each specified file *concurrently*. This means we need to split any large file into multiple files which can then be loaded concurrently (*fizz* is setup to use up to half of the cores it is enabled on to parallelize the loading).

We are going to start by the `frags.fizz` file since it contains over 60000 *statement* definitions spreads over 21 *factual knowledges*. Using the *primitive* `fzz.lst` (which can only be called as an *offload*), we are going to obtain the `GUID` of all these *elementals* and group them, so that we can then save each group into a separate file. To do that, we first need to add to the `import.fizz` (which we have been working on for a while now) some new *procedural knowledges* that will allow us to break a large list into small sub-lists.

For that, we first declare a *procedural knowledge* which we will call `lst.split` which splits a list into two based on an arbitrary number of elements to be included in the first *list*:

```
lst.split {

    ([],_,[],[])              ^:- true;
    ([:e],_,[:e],[])          ^:- true;
    ([:h|:r],1,[:h],:r)       ^:- true;
    ([:h|:r],:c,[:h|:l1],:l2)   :- sub(:c,1,:c1), #lst.split(:r,:c1,:l1,:l2);

}
```

We will then use it in another *procedural knowledge* (which we call `lst.break`) as follows:

```
lst.break {

    ([],_,[])           ^:- true;
    ([:e],_,[[:e]])     ^:- true;
    (:l,:n,[:l2|:l3])    :- #lst.split(:l,:n,:l2,:r), #lst.break(:r,:n,:l3);

}
```

We'll quickly check that they both work as expected:

```
?- #lst.break([a,b,c,d,e,f],2,:l)
-> ( [[a, b], [c, d], [e, f]] ) := 1.00 (0.011) 1
?- #lst.break([a,b,c,d,e,f,g],2,:l)
-> ( [[a, b], [c, d], [e, f], [g]] ) := 1.00 (0.012) 1
```

By combining `lst.break`, and `fzz.lst`, we are able to split all the *elementals* (using their GUIDs) into sub-lists:

```
?- &fzz.lst(frag,:l), #lst.break(:l,3,:l2)
-> ( ["93c52b11-d765-9741-89b3-6869f6eac854", "6b14f85e-aad1-d148-efb6-58f4f89ecf93", "eedf5b25-45b4-6c47-ff88-04a934f6ebc2",
     "f89e5abc-40e4-b140-f190-e2ad51c274b5", "23656e64-c0d0-aa48-22af-d5a3730fe5d7", "050d91ad-d2eb-cf4b-94bd-5b7da5d64526",
     "e9b1603e-32f6-6a48-0c99-2e38a0d6362e", "2b285f0d-ee95-8f47-e3a3-69eca43ef8a9", "995f1eeb-5795-5b4d-2bb2-dee45381126b",
     "3a3c92c2-3995-6f4e-7a83-92b33018f682", "eec06c04-2081-db4c-ed90-a84636a8a6cb", "3328c60c-a842-8945-63be-ab0762eeb262",
     "dd0503c8-ac9f-1f4a-a086-6541f62d5781", "51156d15-40a3-9349-7d85-c83852f2f1eb", "62952974-2a70-b647-1ca5-346c4ff67679",
     "8c5e42b1-21dc-8e47-4ba3-480f22cb26cf", "7332b836-ec54-ea45-4aac-90fadea47203", "9dc15b29-4cd3-b144-8493-5c3ca70c1cee",
     "6b0b3dd4-76a2-2746-1f89-af9245803418", "30331e58-ae61-ff45-7496-65c29acef854", "5c0c524b-cd92-9c42-23a3-01955a487c84"]
     , [["93c52b11-d765-9741-89b3-6869f6eac854", "6b14f85e-aad1-d148-efb6-58f4f89ecf93", "eedf5b25-45b4-6c47-ff88-04
     a934f6ebc2"], ["f89e5abc-40e4-b140-f190-e2ad51c274b5", "23656e64-c0d0-aa48-22af-d5a3730fe5d7", "050d91ad-d2eb-cf4b-94bd
     -5b7da5d64526"], ["e9b1603e-32f6-6a48-0c99-2e38a0d6362e", "2b285f0d-ee95-8f47-e3a3-69eca43ef8a9", "995f1eeb-5795-5b4d-2
     bb2-dee45381126b"], ["3a3c92c2-3995-6f4e-7a83-92b33018f682", "eec06c04-2081-db4c-ed90-a84636a8a6cb", "3328c60c-a842
     -8945-63be-ab0762eeb262"], ["dd0503c8-ac9f-1f4a-a086-6541f62d5781", "51156d15-40a3-9349-7d85-c83852f2f1eb", "62952974-2
     a70-b647-1ca5-346c4ff67679"], ["8c5e42b1-21dc-8e47-4ba3-480f22cb26cf", "7332b836-ec54-ea45-4aac-90fadea47203", "9dc15b29
     -4cd3-b144-8493-5c3ca70c1cee"], ["6b0b3dd4-76a2-2746-1f89-af9245803418", "30331e58-ae61-ff45-7496-65c29acef854", "5
     c0c524b-cd92-9c42-23a3-01955a487c84"]] ) := 1.00 (0.018) 1
```

From there, we will get every sub-list, make up an appropriate filename for it, and use the `console.exec` *primitive* to execute the *command* `save` which when given a *list* of GUIDs will save the identified *elementals* into the same file:

```
?- &fzz.lst(frag,:l), #lst.break(:l,3,:l2), lst.item(:l2,:i,:e), str.cat("./etc/articles/e.coli/frag-",:i,".fizz",:n), console
     .exec(save(:n,:e))
-> ( ["9dc15b29-4cd3-b144-8493-5c3ca70c1cee", "30331e58-ae61-ff45-7496-65c29acef854", "eedf5b25-45b4-6c47-ff88-04a934f6ebc2",
     "3328c60c-a842-8945-63be-ab0762eeb262", "6b14f85e-aad1-d148-efb6-58f4f89ecf93", "62952974-2a70-b647-1ca5-346c4ff67679",
     "dd0503c8-ac9f-1f4a-a086-6541f62d5781", "5c0c524b-cd92-9c42-23a3-01955a487c84", "8c5e42b1-21dc-8e47-4ba3-480f22cb26cf",
     "7332b836-ec54-ea45-4aac-90fadea47203", "2b285f0d-ee95-8f47-e3a3-69eca43ef8a9", "3a3c92c2-3995-6f4e-7a83-92b33018f682",
     "6b0b3dd4-76a2-2746-1f89-af9245803418", "23656e64-c0d0-aa48-22af-d5a3730fe5d7", "e9b1603e-32f6-6a48-0c99-2e38a0d6362e",
     "51156d15-40a3-9349-7d85-c83852f2f1eb", "f89e5abc-40e4-b140-f190-e2ad51c274b5", "93c52b11-d765-9741-89b3-6869f6eac854",
     "995f1eeb-5795-5b4d-2bb2-dee45381126b", "eec06c04-2081-db4c-ed90-a84636a8a6cb", "050d91ad-d2eb-cf4b-94bd-5b7da5d64526"]
     , [["9dc15b29-4cd3-b144-8493-5c3ca70c1cee", "30331e58-ae61-ff45-7496-65c29acef854", "eedf5b25-45b4-6c47-ff88-04
     a934f6ebc2"], ["3328c60c-a842-8945-63be-ab0762eeb262", "6b14f85e-aad1-d148-efb6-58f4f89ecf93", "62952974-2a70-b647-1ca5
     -346c4ff67679"], ["dd0503c8-ac9f-1f4a-a086-6541f62d5781", "5c0c524b-cd92-9c42-23a3-01955a487c84", "8c5e42b1-21dc-8e47-4
     ba3-480f22cb26cf"], ["7332b836-ec54-ea45-4aac-90fadea47203", "2b285f0d-ee95-8f47-e3a3-69eca43ef8a9", "3a3c92c2-3995-6f4e
     -7a83-92b33018f682"], ["6b0b3dd4-76a2-2746-1f89-af9245803418", "23656e64-c0d0-aa48-22af-d5a3730fe5d7", "e9b1603e-32f6-6
     a48-0c99-2e38a0d6362e"], ["51156d15-40a3-9349-7d85-c83852f2f1eb", "f89e5abc-40e4-b140-f190-e2ad51c274b5", "93c52b11-d765
     -9741-89b3-6869f6eac854"], ["995f1eeb-5795-5b4d-2bb2-dee45381126b", "eec06c04-2081-db4c-ed90-a84636a8a6cb", "050d91ad-
```

```
    d2eb-cf4b-94bd-5b7da5d64526"]] , 0 , ["9dc15b29-4cd3-b144-8493-5c3ca70c1cee", "30331e58-ae61-ff45-7496-65c29acef854", "
    eedf5b25-45b4-6c47-ff88-04a934f6ebc2"] , "./etc/articles/e.coli/frag-0.fizz" ) := 1.00 (0.018) 1
-> ( ["9dc15b29-4cd3-b144-8493-5c3ca70c1cee", "30331e58-ae61-ff45-7496-65c29acef854", "eedf5b25-45b4-6c47-ff88-04a934f6ebc2",
    "3328c60c-a842-8945-63be-ab0762eeb262", "6b14f85e-aad1-d148-efb6-58f4f89ecf93", "62952974-2a70-b647-1ca5-346c4ff67679",
    "dd0503c8-ac9f-1f4a-a086-6541f62d5781", "5c0c524b-cd92-9c42-23a3-01955a487c84", "8c5e42b1-21dc-8e47-4ba3-480f22cb26cf",
    "7332b836-ec54-ea45-4aac-90fadea47203", "2b285f0d-ee95-8f47-e3a3-69eca43ef8a9", "3a3c92c2-3995-6f4e-7a83-92b33018f682",
    "6b0b3dd4-76a2-2746-1f89-af9245803418", "23656e64-c0d0-aa48-22af-d5a3730fe5d7", "e9b1603e-32f6-6a48-0c99-2e38a0d6362e",
    "51156d15-40a3-9349-7d85-c83852f2f1eb", "f89e5abc-40e4-b140-f190-e2ad51c274b5", "93c52b11-d765-9741-89b3-6869f6eac854",
    "995f1eeb-5795-5b4d-2bb2-dee45381126b", "eec06c04-2081-db4c-ed90-a84636a8a6cb", "050d91ad-d2eb-cf4b-94bd-5b7da5d64526"]
    , [["9dc15b29-4cd3-b144-8493-5c3ca70c1cee", "30331e58-ae61-ff45-7496-65c29acef854", "eedf5b25-45b4-6c47-ff88-04
    a934f6ebc2"], ["3328c60c-a842-8945-63be-ab0762eeb262", "6b14f85e-aad1-d148-efb6-58f4f89ecf93", "62952974-2a70-b647-1ca5
    -346c4ff67679"], ["dd0503c8-ac9f-1f4a-a086-6541f62d5781", "5c0c524b-cd92-9c42-23a3-01955a487c84", "8c5e42b1-21dc-8e47-4
    ba3-480f22cb26cf"], ["7332b836-ec54-ea45-4aac-90fadea47203", "2b285f0d-ee95-8f47-e3a3-69eca43ef8a9", "3a3c92c2-3995-6f4e
    -7a83-92b33018f682"], ["6b0b3dd4-76a2-2746-1f89-af9245803418", "23656e64-c0d0-aa48-22af-d5a3730fe5d7", "e9b1603e-32f6-6
    a48-0c99-2e38a0d6362e"], ["51156d15-40a3-9349-7d85-c83852f2f1eb", "f89e5abc-40e4-b140-f190-e2ad51c274b5", "93c52b11-d765
    -9741-89b3-6869f6eac854"], ["995f1eeb-5795-5b4d-2bb2-dee45381126b", "eec06c04-2081-db4c-ed90-a84636a8a6cb", "050d91ad-
    d2eb-cf4b-94bd-5b7da5d64526"]] , 1 , ["3328c60c-a842-8945-63be-ab0762eeb262", "6b14f85e-aad1-d148-efb6-58f4f89ecf93",
    "62952974-2a70-b647-1ca5-346c4ff67679"] , "./etc/articles/e.coli/frag-1.fizz" ) := 1.00 (0.018) 2
-> ( ["9dc15b29-4cd3-b144-8493-5c3ca70c1cee", "30331e58-ae61-ff45-7496-65c29acef854", "eedf5b25-45b4-6c47-ff88-04a934f6ebc2",
    "3328c60c-a842-8945-63be-ab0762eeb262", "6b14f85e-aad1-d148-efb6-58f4f89ecf93", "62952974-2a70-b647-1ca5-346c4ff67679",
    "dd0503c8-ac9f-1f4a-a086-6541f62d5781", "5c0c524b-cd92-9c42-23a3-01955a487c84", "8c5e42b1-21dc-8e47-4ba3-480f22cb26cf",
    "7332b836-ec54-ea45-4aac-90fadea47203", "2b285f0d-ee95-8f47-e3a3-69eca43ef8a9", "3a3c92c2-3995-6f4e-7a83-92b33018f682",
    "6b0b3dd4-76a2-2746-1f89-af9245803418", "23656e64-c0d0-aa48-22af-d5a3730fe5d7", "e9b1603e-32f6-6a48-0c99-2e38a0d6362e",
    "51156d15-40a3-9349-7d85-c83852f2f1eb", "f89e5abc-40e4-b140-f190-e2ad51c274b5", "93c52b11-d765-9741-89b3-6869f6eac854",
    "995f1eeb-5795-5b4d-2bb2-dee45381126b", "eec06c04-2081-db4c-ed90-a84636a8a6cb", "050d91ad-d2eb-cf4b-94bd-5b7da5d64526"]
    , [["9dc15b29-4cd3-b144-8493-5c3ca70c1cee", "30331e58-ae61-ff45-7496-65c29acef854", "eedf5b25-45b4-6c47-ff88-04
    a934f6ebc2"], ["3328c60c-a842-8945-63be-ab0762eeb262", "6b14f85e-aad1-d148-efb6-58f4f89ecf93", "62952974-2a70-b647-1ca5
    -346c4ff67679"], ["dd0503c8-ac9f-1f4a-a086-6541f62d5781", "5c0c524b-cd92-9c42-23a3-01955a487c84", "8c5e42b1-21dc-8e47-4
    ba3-480f22cb26cf"], ["7332b836-ec54-ea45-4aac-90fadea47203", "2b285f0d-ee95-8f47-e3a3-69eca43ef8a9", "3a3c92c2-3995-6f4e
    -7a83-92b33018f682"], ["6b0b3dd4-76a2-2746-1f89-af9245803418", "23656e64-c0d0-aa48-22af-d5a3730fe5d7", "e9b1603e-32f6-6
    a48-0c99-2e38a0d6362e"], ["51156d15-40a3-9349-7d85-c83852f2f1eb", "f89e5abc-40e4-b140-f190-e2ad51c274b5", "93c52b11-d765
    -9741-89b3-6869f6eac854"], ["995f1eeb-5795-5b4d-2bb2-dee45381126b", "eec06c04-2081-db4c-ed90-a84636a8a6cb", "050d91ad-
    d2eb-cf4b-94bd-5b7da5d64526"]] , 2 , ["dd0503c8-ac9f-1f4a-a086-6541f62d5781", "5c0c524b-cd92-9c42-23a3-01955a487c84", "8
    c5e42b1-21dc-8e47-4ba3-480f22cb26cf"] , "./etc/articles/e.coli/frag-2.fizz" ) := 1.00 (0.018) 3
-> ( ["9dc15b29-4cd3-b144-8493-5c3ca70c1cee", "30331e58-ae61-ff45-7496-65c29acef854", "eedf5b25-45b4-6c47-ff88-04a934f6ebc2",
    "3328c60c-a842-8945-63be-ab0762eeb262", "6b14f85e-aad1-d148-efb6-58f4f89ecf93", "62952974-2a70-b647-1ca5-346c4ff67679",
    "dd0503c8-ac9f-1f4a-a086-6541f62d5781", "5c0c524b-cd92-9c42-23a3-01955a487c84", "8c5e42b1-21dc-8e47-4ba3-480f22cb26cf",
    "7332b836-ec54-ea45-4aac-90fadea47203", "2b285f0d-ee95-8f47-e3a3-69eca43ef8a9", "3a3c92c2-3995-6f4e-7a83-92b33018f682",
    "6b0b3dd4-76a2-2746-1f89-af9245803418", "23656e64-c0d0-aa48-22af-d5a3730fe5d7", "e9b1603e-32f6-6a48-0c99-2e38a0d6362e",
    "51156d15-40a3-9349-7d85-c83852f2f1eb", "f89e5abc-40e4-b140-f190-e2ad51c274b5", "93c52b11-d765-9741-89b3-6869f6eac854",
    "995f1eeb-5795-5b4d-2bb2-dee45381126b", "eec06c04-2081-db4c-ed90-a84636a8a6cb", "050d91ad-d2eb-cf4b-94bd-5b7da5d64526"]
    , [["9dc15b29-4cd3-b144-8493-5c3ca70c1cee", "30331e58-ae61-ff45-7496-65c29acef854", "eedf5b25-45b4-6c47-ff88-04
    a934f6ebc2"], ["3328c60c-a842-8945-63be-ab0762eeb262", "6b14f85e-aad1-d148-efb6-58f4f89ecf93", "62952974-2a70-b647-1ca5
    -346c4ff67679"], ["dd0503c8-ac9f-1f4a-a086-6541f62d5781", "5c0c524b-cd92-9c42-23a3-01955a487c84", "8c5e42b1-21dc-8e47-4
    ba3-480f22cb26cf"], ["7332b836-ec54-ea45-4aac-90fadea47203", "2b285f0d-ee95-8f47-e3a3-69eca43ef8a9", "3a3c92c2-3995-6f4e
    -7a83-92b33018f682"], ["6b0b3dd4-76a2-2746-1f89-af9245803418", "23656e64-c0d0-aa48-22af-d5a3730fe5d7", "e9b1603e-32f6-6
    a48-0c99-2e38a0d6362e"], ["51156d15-40a3-9349-7d85-c83852f2f1eb", "f89e5abc-40e4-b140-f190-e2ad51c274b5", "93c52b11-d765
    -9741-89b3-6869f6eac854"], ["995f1eeb-5795-5b4d-2bb2-dee45381126b", "eec06c04-2081-db4c-ed90-a84636a8a6cb", "050d91ad-
    d2eb-cf4b-94bd-5b7da5d64526"]] , 3 , ["7332b836-ec54-ea45-4aac-90fadea47203", "2b285f0d-ee95-8f47-e3a3-69eca43ef8a9", "3
    a3c92c2-3995-6f4e-7a83-92b33018f682"] , "./etc/articles/e.coli/frag-3.fizz" ) := 1.00 (0.018) 4
-> ( ["9dc15b29-4cd3-b144-8493-5c3ca70c1cee", "30331e58-ae61-ff45-7496-65c29acef854", "eedf5b25-45b4-6c47-ff88-04a934f6ebc2",
    "3328c60c-a842-8945-63be-ab0762eeb262", "6b14f85e-aad1-d148-efb6-58f4f89ecf93", "62952974-2a70-b647-1ca5-346c4ff67679",
    "dd0503c8-ac9f-1f4a-a086-6541f62d5781", "5c0c524b-cd92-9c42-23a3-01955a487c84", "8c5e42b1-21dc-8e47-4ba3-480f22cb26cf",
    "7332b836-ec54-ea45-4aac-90fadea47203", "2b285f0d-ee95-8f47-e3a3-69eca43ef8a9", "3a3c92c2-3995-6f4e-7a83-92b33018f682",
    "6b0b3dd4-76a2-2746-1f89-af9245803418", "23656e64-c0d0-aa48-22af-d5a3730fe5d7", "e9b1603e-32f6-6a48-0c99-2e38a0d6362e",
    "51156d15-40a3-9349-7d85-c83852f2f1eb", "f89e5abc-40e4-b140-f190-e2ad51c274b5", "93c52b11-d765-9741-89b3-6869f6eac854",
    "995f1eeb-5795-5b4d-2bb2-dee45381126b", "eec06c04-2081-db4c-ed90-a84636a8a6cb", "050d91ad-d2eb-cf4b-94bd-5b7da5d64526"]
    , [["9dc15b29-4cd3-b144-8493-5c3ca70c1cee", "30331e58-ae61-ff45-7496-65c29acef854", "eedf5b25-45b4-6c47-ff88-04
    a934f6ebc2"], ["3328c60c-a842-8945-63be-ab0762eeb262", "6b14f85e-aad1-d148-efb6-58f4f89ecf93", "62952974-2a70-b647-1ca5
    -346c4ff67679"], ["dd0503c8-ac9f-1f4a-a086-6541f62d5781", "5c0c524b-cd92-9c42-23a3-01955a487c84", "8c5e42b1-21dc-8e47-4
    ba3-480f22cb26cf"], ["7332b836-ec54-ea45-4aac-90fadea47203", "2b285f0d-ee95-8f47-e3a3-69eca43ef8a9", "3a3c92c2-3995-6f4e
    -7a83-92b33018f682"], ["6b0b3dd4-76a2-2746-1f89-af9245803418", "23656e64-c0d0-aa48-22af-d5a3730fe5d7", "e9b1603e-32f6-6
    a48-0c99-2e38a0d6362e"], ["51156d15-40a3-9349-7d85-c83852f2f1eb", "f89e5abc-40e4-b140-f190-e2ad51c274b5", "93c52b11-d765
    -9741-89b3-6869f6eac854"], ["995f1eeb-5795-5b4d-2bb2-dee45381126b", "eec06c04-2081-db4c-ed90-a84636a8a6cb", "050d91ad-
    d2eb-cf4b-94bd-5b7da5d64526"]] , 4 , ["6b0b3dd4-76a2-2746-1f89-af9245803418", "23656e64-c0d0-aa48-22af-d5a3730fe5d7", "
    e9b1603e-32f6-6a48-0c99-2e38a0d6362e"] , "./etc/articles/e.coli/frag-4.fizz" ) := 1.00 (0.018) 5
-> ( ["9dc15b29-4cd3-b144-8493-5c3ca70c1cee", "30331e58-ae61-ff45-7496-65c29acef854", "eedf5b25-45b4-6c47-ff88-04a934f6ebc2",
    "3328c60c-a842-8945-63be-ab0762eeb262", "6b14f85e-aad1-d148-efb6-58f4f89ecf93", "62952974-2a70-b647-1ca5-346c4ff67679",
    "dd0503c8-ac9f-1f4a-a086-6541f62d5781", "5c0c524b-cd92-9c42-23a3-01955a487c84", "8c5e42b1-21dc-8e47-4ba3-480f22cb26cf",
    "7332b836-ec54-ea45-4aac-90fadea47203", "2b285f0d-ee95-8f47-e3a3-69eca43ef8a9", "3a3c92c2-3995-6f4e-7a83-92b33018f682",
    "6b0b3dd4-76a2-2746-1f89-af9245803418", "23656e64-c0d0-aa48-22af-d5a3730fe5d7", "e9b1603e-32f6-6a48-0c99-2e38a0d6362e",
    "51156d15-40a3-9349-7d85-c83852f2f1eb", "f89e5abc-40e4-b140-f190-e2ad51c274b5", "93c52b11-d765-9741-89b3-6869f6eac854",
    "995f1eeb-5795-5b4d-2bb2-dee45381126b", "eec06c04-2081-db4c-ed90-a84636a8a6cb", "050d91ad-d2eb-cf4b-94bd-5b7da5d64526"]
    , [["9dc15b29-4cd3-b144-8493-5c3ca70c1cee", "30331e58-ae61-ff45-7496-65c29acef854", "eedf5b25-45b4-6c47-ff88-04
    a934f6ebc2"], ["3328c60c-a842-8945-63be-ab0762eeb262", "6b14f85e-aad1-d148-efb6-58f4f89ecf93", "62952974-2a70-b647-1ca5
    -346c4ff67679"], ["dd0503c8-ac9f-1f4a-a086-6541f62d5781", "5c0c524b-cd92-9c42-23a3-01955a487c84", "8c5e42b1-21dc-8e47-4
    ba3-480f22cb26cf"], ["7332b836-ec54-ea45-4aac-90fadea47203", "2b285f0d-ee95-8f47-e3a3-69eca43ef8a9", "3a3c92c2-3995-6f4e
    -7a83-92b33018f682"], ["6b0b3dd4-76a2-2746-1f89-af9245803418", "23656e64-c0d0-aa48-22af-d5a3730fe5d7", "e9b1603e-32f6-6
    a48-0c99-2e38a0d6362e"], ["51156d15-40a3-9349-7d85-c83852f2f1eb", "f89e5abc-40e4-b140-f190-e2ad51c274b5", "93c52b11-d765
    -9741-89b3-6869f6eac854"], ["995f1eeb-5795-5b4d-2bb2-dee45381126b", "eec06c04-2081-db4c-ed90-a84636a8a6cb", "050d91ad-
    d2eb-cf4b-94bd-5b7da5d64526"]] , 5 , ["51156d15-40a3-9349-7d85-c83852f2f1eb", "f89e5abc-40e4-b140-f190-e2ad51c274b5",
```

```
     "93c52b11-d765-9741-89b3-6869f6eac854"] , "./etc/articles/e.coli/frag-5.fizz" ) := 1.00 (0.019) 6
-> ( ["9dc15b29-4cd3-b144-8493-5c3ca70c1cee", "30331e58-ae61-ff45-7496-65c29acef854", "eedf5b25-45b4-6c47-ff88-04a934f6ebc2",
     "3328c60c-a842-8945-63be-ab0762eeb262", "6b14f85e-aad1-d148-efb6-58f4f89ecf93", "62952974-2a70-b647-1ca5-346c4ff67679",
     "dd0503c8-ac9f-1f4a-a086-6541f62d5781", "5c0c524b-cd92-9c42-23a3-01955a487c84", "8c5e42b1-21dc-8e47-4ba3-480f22cb26cf",
     "7332b836-ec54-ea45-4aac-90fadea47203", "2b285f0d-ee95-8f47-e3a3-69eca43ef8a9", "3a3c92c2-3995-6f4e-7a83-92b33018f682",
     "6b0b3dd4-76a2-2746-1f89-af9245803418", "23656e64-c0d0-aa48-22af-d5a3730fe5d7", "e9b1603e-32f6-6a48-0c99-2e38a0d6362e",
     "51156d15-40a3-9349-7d85-c83852f2f1eb", "f89e5abc-40e4-b140-f190-e2ad51c274b5", "93c52b11-d765-9741-89b3-6869f6eac854",
     "995f1eeb-5795-5b4d-2bb2-dee45381126b", "eec06c04-2081-db4c-ed90-a84636a8a6cb", "050d91ad-d2eb-cf4b-94bd-5b7da5d64526"]
     , [["9dc15b29-4cd3-b144-8493-5c3ca70c1cee", "30331e58-ae61-ff45-7496-65c29acef854", "eedf5b25-45b4-6c47-ff88-04
     a934f6ebc2"], ["3328c60c-a842-8945-63be-ab0762eeb262", "6b14f85e-aad1-d148-efb6-58f4f89ecf93", "62952974-2a70-b647-1ca5
     -346c4ff67679"], ["dd0503c8-ac9f-1f4a-a086-6541f62d5781", "5c0c524b-cd92-9c42-23a3-01955a487c84", "8c5e42b1-21dc-8e47-4
     ba3-480f22cb26cf"], ["7332b836-ec54-ea45-4aac-90fadea47203", "2b285f0d-ee95-8f47-e3a3-69eca43ef8a9", "3a3c92c2-3995-6f4e
     -7a83-92b33018f682"], ["6b0b3dd4-76a2-2746-1f89-af9245803418", "23656e64-c0d0-aa48-22af-d5a3730fe5d7", "e9b1603e-32f6-6
     a48-0c99-2e38a0d6362e"], ["51156d15-40a3-9349-7d85-c83852f2f1eb", "f89e5abc-40e4-b140-f190-e2ad51c274b5", "93c52b11-d765
     -9741-89b3-6869f6eac854"], ["995f1eeb-5795-5b4d-2bb2-dee45381126b", "eec06c04-2081-db4c-ed90-a84636a8a6cb", "050d91ad-
     d2eb-cf4b-94bd-5b7da5d64526"]] , 6 , ["995f1eeb-5795-5b4d-2bb2-dee45381126b", "eec06c04-2081-db4c-ed90-a84636a8a6cb",
     "050d91ad-d2eb-cf4b-94bd-5b7da5d64526"] , "./etc/articles/e.coli/frag-6.fizz" ) := 1.00 (0.019) 7
save : completed in 0.051s.
save : completed in 0.067s.
save : completed in 0.083s.
save : completed in 0.060s.
save : completed in 0.134s.
save : completed in 0.050s.
save : completed in 0.091s.
```

We can now restart *fizz* and this time use the seven files containing the DNA fragments instead of the single file:

```
$ ./fizz.x64 ./etc/articles/e.coli/genes.fizz ./etc/articles/e.coli/frag-* ./etc/articles/e.coli/import.fizz
fizz 0.3.0-X (20180519.2228) [x64|8|w|l]

load : loading ./etc/articles/e.coli/genes.fizz ...
load : loading ./etc/articles/e.coli/frag-1.fizz ...
load : loading ./etc/articles/e.coli/frag-2.fizz ...
load : loading ./etc/articles/e.coli/frag-0.fizz ...
load : loaded  ./etc/articles/e.coli/frag-2.fizz in 0.969s
load : loaded  ./etc/articles/e.coli/frag-0.fizz in 0.980s
load : loaded  ./etc/articles/e.coli/frag-1.fizz in 1.046s
load : loading ./etc/articles/e.coli/frag-3.fizz ...
load : loading ./etc/articles/e.coli/frag-4.fizz ...
load : loading ./etc/articles/e.coli/frag-5.fizz ...
load : loaded  ./etc/articles/e.coli/frag-3.fizz in 1.043s
load : loaded  ./etc/articles/e.coli/frag-4.fizz in 1.057s
load : loaded  ./etc/articles/e.coli/frag-5.fizz in 0.986s
load : loading ./etc/articles/e.coli/frag-6.fizz ...
load : loading ./etc/articles/e.coli/import.fizz ...
load : loaded  ./etc/articles/e.coli/import.fizz in 0.031s
load : loaded  ./etc/articles/e.coli/genes.fizz in 2.656s
load : loaded  ./etc/articles/e.coli/frag-6.fizz in 0.730s
load : loading completed in 2.948s
?- /stats
stats : e:36 k:32 s:66393 p:15 u:57.54 t:0 q:0 r:0 z:0
```

We now have a much better loading time.

Optimizing retrieval time

Once all the DNA fragments are loaded, let's see what sort of performance we can get when retrieving a particular fragment based on its identifier:

```
?- #frag(4528,:s)
-> ( "GCAGCGGCATTCTGCCGGTGATCAACACCGCCATCGCCCATAAAGATGCGGGCGTCGGCATGATTGGCGCGGGCA" ) := 1.00 (0.135) 1
?- #frag(60000,:s)
-> ( "ACAGGCAATTTTTCGGGGATACTGCTCCAGGTAATTATTCGGCTAGGAGTTAAGGCTGTCACACGGATTTGGATG" ) := 1.00 (0.118) 1
?- #frag(1,:s)
-> ( "AACTGGTTACCTGCCGTGAGTAAATTAAAATTTTATTGACTTAGGTCACTAAATACTTTAACCAATATAGGCATA" ) := 1.00 (0.118) 1
```

As we have spread all the DNA fragments over 21 *elementals*, when we query the *runtime* for a specific one, the query is sent concurrently to all. While this is faster than if we had all the sequences in a single

elemental, there's one simple thing we can do to improve the performance: indexing the *statements* in each *elemental* based on the first *term*. This will allow for a faster retrieval of any *statements* when one of the indexed *terms* is bound to a value in a *query*.

To do that, we are going to instruct each of the *elementals* to setup an index using the `poke` console *command*:

```
?- /poke(frag,index,0)
```

This gives a value of `0` (the position of the *term* to be use as the indexing key in the *statements*) to the `index` property of all the *elemental* objects labeled `frag`. We can now check if this has improved the query performance:

```
?- #frag(4528,:s)
-> ( "GCAGCGGCATTCTGCCGGTGATCAACACCGCCATCGCCCATAAAGATGCGGGCGTCGGCATGATTGGCGCGGGCA" ) := 1.00 (0.001) 1
?- #frag(60000,:s)
-> ( "ACAGGCAATTTTTCGGGGATACTGCTCCAGGTAATTATTCGGCTAGGAGTTAAGGCTGTCACACGGATTTGGATG" ) := 1.00 (0.001) 1
?- #frag(1,:s)
-> ( "AACTGGTTACCTGCCGTGAGTAAATTAAAATTTTATTGACTTAGGTCACTAAATACTTTAACCAATATAGGCATA" ) := 1.00 (0.001) 1
```

That's much better. Note that once an index has been created, the *elemental* will maintain it when *statements* are added or removed, so you do not have to `poke` at it after changes. Multiple indexes are also supported (using a *list* of indexes).

In order to avoid having to *poke* each time we load the DNA sequences, we are going to once again save all the `frag` *elementals* to file. We will use a different filename so as to keep the non-indexed version:

```
?- &fzz.lst(frag,:l), #lst.break(:l,3,:l2), lst.item(:l2,:i,:e), str.cat("./etc/articles/e.coli/frag-i-",:i,".fizz",:n),
     console.exec(save(:n,:e))
```

Alternatively, we could have edited each of the *fizz* files with a text editor and manually added the index property for each of the *knowledge* definitions - such as this:

```
1 frag {index = 0} {
2
3     (24576, "GAAACTGGCACGTCTTATCGAAATCAAAGCCAGCCGCGATGGTCGAGTGGCAGATTACGCCAAAGAATTTGGTCT");
4     (24577, "GGTCTATCTCGAAGGCCAACAGCCGTGGTCTCTACCGGTTGATATCGCCCTGCCTTGCGCCACCCAGAATGAACT");
5     (24578, "GGATGTTGACGCCGCGCATCAGCTTATCGCTAATGGCGTTAAAGCCGTCGCCGAAGGGGCAAATATGCCGACCAC");
6     (24579, "CATCGAAGCGACTGAACTGTTCCAGCAGGCAGGCGTACTATTTGCACCGGGTAAAGCGGCTAATGCTGGTGGCGT");
7     ...
8     ...
9 }
```

We will complete this section by indexing the genes data using the gene's name as the indexing *term* (the third *term* in each *statements*):

```
?- #gene(_,_,"feaR",_,_,_,:orient,:s,:e,_,_,_)
-> ( Counterclockwise , 1446378 , 1447283 ) := 1.00 (0.163) 1
?- /poke(gene,index,2)
?- #gene(_,_,"feaR",_,_,_,:orient,:s,:e,_,_,_)
-> ( Counterclockwise , 1446378 , 1447283 ) := 1.00 (0.001) 1
?- /save("./etc/articles/e.coli/genes-i.fizz",gene)
save : completed in 0.151s.
```

Finally, we also saved the newly modified *elementals* into a different *fizz* file.

Finding the famous GATTACA in the genes

To conclude this article, let's look at something a little more fun: finding all the genes whose DNA contains at least one occurance of the famous `GATTACA` string (from the 1997 sci-fi movie of the same name). To that end, we are going to have to write some *procedural knowledge* which, when given a gene's name, will retreive the complete DNA sequence, and check if it contains a given *substring*.

To start, create a new *fizz* file called `base.fizz`. We will first write a way for us to, given a gene's name, retrieve the offset and length (in base-pairs) of its DNA sequence, as well as the orientation of the sequence (as we will see later - it matters!):

```
1 gene.offset {
2
3     (:name,:offset,:length,:orient) :-  #gene(_,_,:name,_,_,_,:orient,:s,:e,_,_,_),
4                                         sub(:e,:s,:l2),
5                                         add(:l2,1,:length),
6                                         sub(:s,1,:offset);
7
8 }
```

The first thing we do, is to query (line 3) the `gene` *factual knowledge* for any *statements* matching the gene's name. Since we only care about a few of the *terms*, we use the wildcard *variable* for most of the *terms*. Once we have the start and end base-pairs, we compute the length of the sequence (line 4 and 5) by substracting the start and end offset, and then adding 1 to it. The result is unified with the variable `length` which we will return. We end (line 6) by substracting 1 from the start offset, as the base-pair offset starts at 0 for us instead of 1.

Let's give this a try:

```
$ ./fizz.x64 ./etc/articles/e.coli/genes-i.fizz ./etc/articles/e.coli/frag-i-*.fizz ./etc/articles/e.coli/base.fizz
fizz 0.3.0-X (20180519.2228) [x64|8|w|l]

load : loading ./etc/articles/e.coli/genes-i.fizz ...
load : loading ./etc/articles/e.coli/frag-i-1.fizz ...
load : loading ./etc/articles/e.coli/frag-i-0.fizz ...
load : loading ./etc/articles/e.coli/frag-i-2.fizz ...
load : loaded  ./etc/articles/e.coli/frag-i-0.fizz in 1.401s
load : loaded  ./etc/articles/e.coli/frag-i-2.fizz in 1.419s
load : loading ./etc/articles/e.coli/frag-i-3.fizz ...
load : loading ./etc/articles/e.coli/frag-i-4.fizz ...
load : loaded  ./etc/articles/e.coli/frag-i-1.fizz in 1.637s
load : loading ./etc/articles/e.coli/frag-i-5.fizz ...
load : loaded  ./etc/articles/e.coli/frag-i-5.fizz in 1.163s
load : loading ./etc/articles/e.coli/frag-i-6.fizz ...
load : loaded  ./etc/articles/e.coli/frag-i-4.fizz in 1.428s
load : loaded  ./etc/articles/e.coli/frag-i-3.fizz in 1.552s
load : loading ./etc/articles/e.coli/base.fizz ...
load : loaded  ./etc/articles/e.coli/base.fizz in 0.020s
load : loaded  ./etc/articles/e.coli/genes-i.fizz in 3.915s
load : loaded  ./etc/articles/e.coli/frag-i-6.fizz in 1.368s
load : loading completed in 4.373s
?- #gene.offset("feaR",:o,:l,:or)
-> ( 1446377 , 906 , Counterclockwise ) := 1.00 (0.001) 1
```

Next, we are going to assemble the complete gene sequence from an offset, and length, and this is going to be a little bit more tricky. First, we will need to find out which of the 60000+ fragments contains the start of the gene (based on the starting offset we retreived earlier). The following *procedural knowledge* implements this:

```
1 frag.offset.to.id {
2
3     (:offset,:id,:off) :- div.int(:offset,75,:id), mod(:offset,75,:off);
4
5 }
```

It relies on the fact that each of the fragments (except the last one) contains 75 characters (base-pairs) to compute the `ID` of the first sequence (using `div.int`) as well as the actual offset within that sequence using the `mod` *primitive*. If we combine that new *procedural knowledge* with the `frag` one, we can retrieve the very first fragment from which we will still have to extract the relevant part:

```
?- #frag.offset.to.id(1446377,:id,:o), #frag(:id,:s)
-> ( 19285 , 2 , "AGTTAGCGGAATTTACGTCGATACTCGCCTGGCGTCATCCCAAAGCGTTGCTTAAATACCGTTGAAAAATGACTC" ) := 1.00 (0.001) 1
```

Once we have the starting sequence, we are going to write the *procedural knowledge* that will assemble a complete sequence given the `ID` of the first fragment, the offset in that fragment and the total length of the sequence:

```
1  frag.get {
2
3      (:id,_,0,[])                ^:- true;
4
5      (:id,0,:len?[lte(0)],[])    ^:- true;
6
7      (:id,0,:len?[lte(75)],[:h]) ^:- #frag(:id,:frag),              // grab the sequence for the id
8                                      str.sub(:frag,0,:len,:h);      // extract the part of it we need
9
10     (:id,0,:len,[:h|:r])        ^:- #frag(:id,:h),                 // grab the sequence for the id
11                                     str.length(:h,:h.size),        // get the size of the part we just got
12                                     sub(:len,:h.size,:len.2),      // subtract it from the length still to get
13                                     add(:id,1,:id.2),              // compute the next sequence's id
14                                     #frag.get(:id.2,0,:len.2,:r);  // get the next sequence
15
16     (:id,:off,:len,[:h|:r])      :- #frag(:id,:frag),              // grab the sequence for the id
17                                     str.rest(:frag,:off,:h),       // extract the part of it we need
18                                     str.length(:h,:h.size),        // get the size of the part we just got
19                                     sub(:len,:h.size,:len.2),      // subtract it from the length still to get
20                                     add(:id,1,:id.2),              // compute the next sequence's id
21                                     #frag.get(:id.2,0,:len.2,:r);  // get the next sequence
22
23 }
```

It works by recursively constructing a *list* that contains all the fragments that compose a gene's sequence. To simplify further processing, we are going to transform that *list* of *strings* into a single *string* by concatenating all of its elements. We will also combine that operation with the call to `frag.offset.to.id`:

```
1  frag.fetch {
2
3      (:offset,:length,:s) :- #frag.offset.to.id(:offset,:i,:o),
4                              #frag.get(:i,:o,:length,:l),
5                              str.tokenize(:s,"",:l);
6
7
8  }
```

Let's try it with the offset and length we got earlier for the *feaR* gene:

```
?- #frag.fetch(1446377,906,:s)
-> ( "
     TTAGCGGAATTTACGTCGATACTCGCCTGGCGTCATCCCAAAGCGTTGCTTAAATACCGTTGAAAAATGACTCTGGTCAGAAAATCCCCAATGAAAGCCGATGCCTGCCAGTTTTTCATCA

TCTGCGGCATGGCGAATCGCATCTGCACAAAAATCGAGACGACGGTTACGAATATATTGCGCGACTACCAAACCTTTATCGGCAAACATTCGGTACAAACTACGTACTGACATACCTGTCTCTCCGG

CTATCCACTCCGGGGCGTAATATCTCTTCGCGAATATTATCGTCTATCAACGTAACCACTTTTTGAAACTGACGTTCACGACGAGGTTGAACAGATTCCCGCTGATGAAGTACCGGGCGCAGCAGACA

CACCATCGCCTGTAGCGCAGCTTCACTTTCTGTTTCAGAAAGTGCCGGATTATTCATGCTCTCCTGTAACAGGCGATGACTGAGTTGCACCATGGGTAAGTCAGCGTCCAGTCTTTCTGCGCAGATA

GGTTTTTGATGGGGAAAATATTGTTCCAGCAGAGTGCGTGGCAAAAGTAATGAAATCTGTTTAGAAGACTCCTGCCAGTAAAGCGAACAGGGGCGTGAGGCATCGAGTAACGTAATATCGCCAGCGC

CAATCTGCACCTGACGCTCATCCTGCTCCATTATTGCCTGACCACTAAGCTGAAAAACGGTGTAAAACCAGGCATCGTCGCTGCCTTTTACTTCCTGCCAGGTGCGGGATAAATTCACCCCGCTGGT
```

```
TGTCACGGTACTCAGCTTTAGTCCTTTGGCAAAATGCGTGTCCAGTACACCCGTGTAACGCTCAGTCAGCAGGCGTCCGGTAAAATTTCCGCATACCTGATTGATTTGGGAAAGCCATTGCTGAAAC

TCATTATCCACTGCGGGGTTCAT" ) := 1.00 (0.011) 1
```

The last bit of processing we need to do now, is to handle the gene's orientation. When it is *Counterclockwise*, we need to inverse the sequence, and take its *complement*. Taking the *complement* of a DNA sequence is done by swapping `A` for a `T`, `C` for a `G`, `G` for a `C` and `T` for an `A`. When the orientation is *Clockwise* no further processing of the reassembled sequence is necessary. We will create a new *procedural knowledge* to handle this:

```
1 frag.orient {
2
3     (Clockwise,:s,:s)           ^:- true;
4     (Counterclockwise,:s,:s.c)   :- str.flip(:s,:s.i), str.swap(:s.i,[["A","T"],["C","G"],["G","C"],["T","A"]],:s.c);
5
6 }
```

It makes use of the *primitive* `str.flip` to invert the sequence, and of `str.swap` to swap the characters.

We are now ready to put the *procedural knowledge* `gene.find` together by combining all the *procedural knowledge* we have just created. Once we have finished processing the gene's whole sequence, we will use the *primitive* `str.find` to test if it contains the fragment we are looking for:

```
1 gene.find {
2
3     (:frag,:n) :-   #gene.offset(:n,:of,:l,:or),
4                     #frag.fetch(:of,:l,:s),
5                     #frag.orient(:or,:s,:s.o),
6                     str.find(:s.o,:frag,_);
7
8 }
```

We can then look for `GATTACA` in the entire *E. coli* genome:

```
?- #gene.find("GATTACA",:name)
-> ( "ymgJ" ) := 1.00 (8.917) 1
-> ( "ybfQ'" ) := 1.00 (11.420) 2
-> ( "ydfC" ) := 1.00 (12.051) 3
-> ( "ynaE" ) := 1.00 (12.067) 4
-> ( "ydfK" ) := 1.00 (12.168) 5
-> ( "yjbD" ) := 1.00 (12.825) 6
-> ( "acpP" ) := 1.00 (13.674) 7
-> ( "soxS" ) := 1.00 (15.548) 8
-> ( "tfaS'" ) := 1.00 (15.664) 9
-> ( "yeeX" ) := 1.00 (15.686) 10
-> ( "yegR" ) := 1.00 (16.249) 11
-> ( "clpS" ) := 1.00 (16.326) 12
-> ( "grcA" ) := 1.00 (17.798) 13
-> ( "iscA" ) := 1.00 (17.979) 14
-> ( "ybcV" ) := 1.00 (18.267) 15
-> ( "yffQ" ) := 1.00 (18.283) 16
-> ( "symE" ) := 1.00 (18.563) 17
-> ( "tusC" ) := 1.00 (18.586) 18
-> ( "greB" ) := 1.00 (19.987) 19
-> ( "flgN" ) := 1.00 (20.483) 20
-> ( "yfdK" ) := 1.00 (20.552) 21
-> ( "ymfA" ) := 1.00 (20.947) 22
-> ( "hycA" ) := 1.00 (21.006) 23
-> ( "yfbU" ) := 1.00 (21.122) 24
-> ( "rplM" ) := 1.00 (21.130) 25
-> ( "rplO" ) := 1.00 (21.132) 26
-> ( "ymfQ" ) := 1.00 (22.688) 27
-> ( "yafW" ) := 1.00 (22.804) 28
-> ( "msrC" ) := 1.00 (23.311) 29
-> ( "tfaE" ) := 1.00 (24.516) 30
-> ( "yfdL'" ) := 1.00 (24.599) 31
-> ( "nrfG" ) := 1.00 (24.756) 32
```

```
-> ( "yfjJ" ) := 1.00 (24.772) 33
-> ( "yffP" ) := 1.00 (24.798) 34
-> ( "elfA" ) := 1.00 (24.843) 35
-> ( "ycdY" ) := 1.00 (24.862) 36
-> ( "yceI" ) := 1.00 (24.948) 37
-> ( "yraR" ) := 1.00 (25.014) 38
-> ( "ybdM" ) := 1.00 (25.052) 39
-> ( "pabA" ) := 1.00 (25.181) 40
-> ( "purN" ) := 1.00 (25.217) 41
-> ( "pyrE" ) := 1.00 (25.221) 42
-> ( "fetA" ) := 1.00 (25.522) 43
-> ( "araD" ) := 1.00 (25.866) 44
-> ( "pnuC" ) := 1.00 (26.053) 45
-> ( "yjjG" ) := 1.00 (26.232) 46
-> ( "yhjY" ) := 1.00 (26.262) 47
-> ( "pepE" ) := 1.00 (26.339) 48
-> ( "alsE" ) := 1.00 (26.341) 49
-> ( "cysQ" ) := 1.00 (26.445) 50
-> ( "frdB" ) := 1.00 (26.674) 51
-> ( "yegL" ) := 1.00 (26.924) 52
-> ( "yfbN" ) := 1.00 (26.946) 53
-> ( "trmH" ) := 1.00 (26.980) 54
-> ( "flgF" ) := 1.00 (27.064) 55
-> ( "ygeR" ) := 1.00 (27.217) 56
-> ( "ybfD" ) := 1.00 (27.752) 57
-> ( "lldR" ) := 1.00 (27.991) 58
-> ( "yjjP" ) := 1.00 (28.101) 59
-> ( "torR" ) := 1.00 (28.103) 60
-> ( "allE" ) := 1.00 (28.236) 61
-> ( "ycdX" ) := 1.00 (28.256) 62
-> ( "ygdG" ) := 1.00 (28.326) 63
-> ( "fadR" ) := 1.00 (28.335) 64
-> ( "fucR" ) := 1.00 (28.439) 65
-> ( "map" ) := 1.00 (28.481) 66
-> ( "ynfC" ) := 1.00 (28.488) 67
-> ( "pstB" ) := 1.00 (28.541) 68
-> ( "sdhB" ) := 1.00 (28.571) 69
-> ( "flgG" ) := 1.00 (28.670) 70
-> ( "ecpR" ) := 1.00 (28.686) 71
-> ( "insF4" ) := 1.00 (28.847) 72
-> ( "ppsR" ) := 1.00 (29.047) 73
-> ( "insF5" ) := 1.00 (29.091) 74
-> ( "hisJ" ) := 1.00 (29.349) 75
-> ( "yajG" ) := 1.00 (29.359) 76
-> ( "yiaJ" ) := 1.00 (29.381) 77
-> ( "yqeI" ) := 1.00 (29.514) 78
-> ( "queF" ) := 1.00 (29.517) 79
-> ( "ecpA" ) := 1.00 (29.550) 80
-> ( "rutD" ) := 1.00 (29.577) 81
-> ( "ycjP" ) := 1.00 (29.580) 82
-> ( "atpB" ) := 1.00 (29.615) 83
-> ( "betI" ) := 1.00 (29.616) 84
-> ( "ybgF" ) := 1.00 (29.747) 85
-> ( "hofM" ) := 1.00 (29.751) 86
-> ( "yegX" ) := 1.00 (29.973) 87
-> ( "cbl" ) := 1.00 (30.037) 88
-> ( "nac" ) := 1.00 (30.186) 89
-> ( "insF3" ) := 1.00 (30.214) 90
-> ( "fdnH" ) := 1.00 (30.440) 91
-> ( "lipA" ) := 1.00 (30.471) 92
-> ( "rbsB" ) := 1.00 (30.531) 93
-> ( "ykgE" ) := 1.00 (30.555) 94
-> ( "gluQ" ) := 1.00 (30.696) 95
-> ( "lpxP" ) := 1.00 (30.869) 96
-> ( "ygbJ" ) := 1.00 (30.931) 97
-> ( "abgR" ) := 1.00 (30.948) 98
-> ( "dgcZ" ) := 1.00 (31.012) 99
-> ( "nadK" ) := 1.00 (31.035) 100
-> ( "era" ) := 1.00 (31.050) 101
-> ( "yhcC" ) := 1.00 (31.063) 102
-> ( "glpG" ) := 1.00 (31.105) 103
-> ( "metA" ) := 1.00 (31.180) 104
-> ( "lysR" ) := 1.00 (31.227) 105
-> ( "mcrA" ) := 1.00 (31.233) 106
-> ( "aes" ) := 1.00 (31.632) 107
-> ( "proX" ) := 1.00 (31.789) 108
-> ( "ispB" ) := 1.00 (31.931) 109
-> ( "rseB" ) := 1.00 (32.112) 110
-> ( "glsA" ) := 1.00 (32.115) 111
-> ( "ltaE" ) := 1.00 (32.188) 112
-> ( "ybjX" ) := 1.00 (32.189) 113
```

```
-> ( "yegV" ) := 1.00 (32.214) 114
-> ( "yggF" ) := 1.00 (32.216) 115
-> ( "yhbE" ) := 1.00 (32.255) 116
-> ( "ydjG" ) := 1.00 (32.291) 117
-> ( "galE" ) := 1.00 (32.304) 118
-> ( "hemH" ) := 1.00 (32.315) 119
-> ( "ilvE" ) := 1.00 (32.320) 120
-> ( "lepB" ) := 1.00 (32.326) 121
-> ( "motB" ) := 1.00 (32.338) 122
-> ( "rbsR" ) := 1.00 (32.374) 123
-> ( "waaH" ) := 1.00 (32.772) 124
-> ( "potF" ) := 1.00 (32.868) 125
-> ( "eutR" ) := 1.00 (33.018) 126
-> ( "dppB" ) := 1.00 (33.058) 127
-> ( "waaB" ) := 1.00 (33.079) 128
-> ( "waaR" ) := 1.00 (33.080) 129
-> ( "glcE" ) := 1.00 (33.083) 130
-> ( "ychF" ) := 1.00 (33.085) 131
-> ( "aroC" ) := 1.00 (33.193) 132
-> ( "yhhT" ) := 1.00 (33.217) 133
-> ( "ydaN" ) := 1.00 (33.251) 134
-> ( "yeeL'" ) := 1.00 (33.253) 135
-> ( "galT" ) := 1.00 (33.267) 136
-> ( "gapA" ) := 1.00 (33.267) 137
-> ( "mcrC" ) := 1.00 (33.291) 138
-> ( "recF" ) := 1.00 (33.329) 139
-> ( "yncI'" ) := 1.00 (33.345) 140
-> ( "insF2" ) := 1.00 (33.510) 141
-> ( "yciM" ) := 1.00 (33.680) 142
-> ( "yhhI" ) := 1.00 (33.716) 143
-> ( "ydcC" ) := 1.00 (33.716) 144
-> ( "trmA" ) := 1.00 (33.859) 145
-> ( "yhdY" ) := 1.00 (33.864) 146
-> ( "ybfL'" ) := 1.00 (33.936) 147
-> ( "macA" ) := 1.00 (33.947) 148
-> ( "ydcS" ) := 1.00 (33.952) 149
-> ( "bcsZ" ) := 1.00 (33.992) 150
-> ( "potA" ) := 1.00 (34.079) 151
-> ( "uxuA" ) := 1.00 (34.291) 152
-> ( "insF1" ) := 1.00 (34.474) 153
-> ( "lldD" ) := 1.00 (34.500) 154
-> ( "yhjX" ) := 1.00 (34.531) 155
-> ( "gspF" ) := 1.00 (34.566) 156
-> ( "ybdL" ) := 1.00 (34.591) 157
-> ( "allC" ) := 1.00 (34.598) 158
-> ( "gfcE" ) := 1.00 (34.602) 159
-> ( "dacD" ) := 1.00 (34.609) 160
-> ( "moeA" ) := 1.00 (34.631) 161
-> ( "dacA" ) := 1.00 (34.680) 162
-> ( "yhaC" ) := 1.00 (34.690) 163
-> ( "argA" ) := 1.00 (34.694) 164
-> ( "yhfX" ) := 1.00 (34.705) 165
-> ( "ydaM" ) := 1.00 (34.714) 166
-> ( "yedS'" ) := 1.00 (34.766) 167
-> ( "yfaY" ) := 1.00 (34.773) 168
-> ( "panE" ) := 1.00 (34.808) 169
-> ( "ugpB" ) := 1.00 (34.962) 170
-> ( "mdtG" ) := 1.00 (35.074) 171
-> ( "yihS" ) := 1.00 (35.212) 172
-> ( "rhaA" ) := 1.00 (35.218) 173
-> ( "uraA" ) := 1.00 (35.253) 174
-> ( "menF" ) := 1.00 (35.279) 175
-> ( "intR" ) := 1.00 (35.330) 176
-> ( "amiC" ) := 1.00 (35.336) 177
-> ( "ybbY" ) := 1.00 (35.358) 178
-> ( "aroA" ) := 1.00 (35.434) 179
-> ( "dsdA" ) := 1.00 (35.455) 180
-> ( "yddW" ) := 1.00 (35.502) 181
-> ( "fliI" ) := 1.00 (35.581) 182
-> ( "xseA" ) := 1.00 (35.707) 183
-> ( "tauA" ) := 1.00 (35.746) 184
-> ( "rhlE" ) := 1.00 (35.749) 185
-> ( "yahB" ) := 1.00 (35.759) 186
-> ( "dgoT" ) := 1.00 (35.807) 187
-> ( "yajO" ) := 1.00 (35.817) 188
-> ( "pepB" ) := 1.00 (35.855) 189
-> ( "ppnN" ) := 1.00 (35.859) 190
-> ( "sgcC" ) := 1.00 (35.872) 191
-> ( "frlA" ) := 1.00 (35.934) 192
-> ( "ygeH" ) := 1.00 (35.938) 193
-> ( "ssnA" ) := 1.00 (35.939) 194
```

```
-> ( "aldA" ) := 1.00 (35.943) 195
-> ( "ydgI" ) := 1.00 (35.965) 196
-> ( "deoA" ) := 1.00 (35.984) 197
-> ( "tdcG" ) := 1.00 (36.001) 198
-> ( "gor" ) := 1.00 (36.020) 199
-> ( "mcrB" ) := 1.00 (36.028) 200
-> ( "pheP" ) := 1.00 (36.039) 201
-> ( "phr" ) := 1.00 (36.040) 202
-> ( "tdcC" ) := 1.00 (36.060) 203
-> ( "trkA" ) := 1.00 (36.062) 204
-> ( "baeS" ) := 1.00 (36.308) 205
-> ( "cycA" ) := 1.00 (36.337) 206
-> ( "nrfA" ) := 1.00 (36.348) 207
-> ( "norV" ) := 1.00 (36.354) 208
-> ( "glrK" ) := 1.00 (36.369) 209
-> ( "patD" ) := 1.00 (36.384) 210
-> ( "cysN" ) := 1.00 (36.453) 211
-> ( "hyuA" ) := 1.00 (36.467) 212
-> ( "ygcU" ) := 1.00 (36.471) 213
-> ( "degP" ) := 1.00 (36.490) 214
-> ( "yidJ" ) := 1.00 (36.783) 215
-> ( "mltF" ) := 1.00 (36.790) 216
-> ( "ybcK" ) := 1.00 (36.842) 217
-> ( "glpD" ) := 1.00 (36.939) 218
-> ( "glpK" ) := 1.00 (36.940) 219
-> ( "lysS" ) := 1.00 (36.949) 220
-> ( "sufB" ) := 1.00 (36.958) 221
-> ( "tolC" ) := 1.00 (36.972) 222
-> ( "abgT" ) := 1.00 (37.000) 223
-> ( "renD'" ) := 1.00 (37.105) 224
-> ( "aceB" ) := 1.00 (37.113) 225
-> ( "pgm" ) := 1.00 (37.191) 226
-> ( "yeeR" ) := 1.00 (37.234) 227
-> ( "opgE" ) := 1.00 (37.240) 228
-> ( "mppA" ) := 1.00 (37.240) 229
-> ( "purH" ) := 1.00 (37.339) 230
-> ( "flgK" ) := 1.00 (37.591) 231
-> ( "dcuS" ) := 1.00 (37.689) 232
-> ( "dld" ) := 1.00 (37.711) 233
-> ( "yhiJ" ) := 1.00 (37.718) 234
-> ( "bcsG" ) := 1.00 (37.720) 235
-> ( "glpA" ) := 1.00 (37.734) 236
-> ( "treA" ) := 1.00 (37.758) 237
-> ( "rclA" ) := 1.00 (37.780) 238
-> ( "yfjW" ) := 1.00 (37.936) 239
-> ( "proA" ) := 1.00 (37.986) 240
-> ( "typA" ) := 1.00 (38.121) 241
-> ( "gss" ) := 1.00 (38.204) 242
-> ( "codA" ) := 1.00 (38.213) 243
-> ( "btuB" ) := 1.00 (38.234) 244
-> ( "ynbC" ) := 1.00 (38.257) 245
-> ( "lepA" ) := 1.00 (38.259) 246
-> ( "yciQ" ) := 1.00 (38.445) 247
-> ( "hscA" ) := 1.00 (38.455) 248
-> ( "thiC" ) := 1.00 (38.467) 249
-> ( "intF" ) := 1.00 (38.613) 250
-> ( "gmr" ) := 1.00 (38.700) 251
-> ( "ygfT" ) := 1.00 (38.721) 252
-> ( "yoaA" ) := 1.00 (38.724) 253
-> ( "cpdB" ) := 1.00 (38.736) 254
-> ( "malS" ) := 1.00 (38.805) 255
-> ( "paaZ" ) := 1.00 (38.846) 256
-> ( "wbbL'" ) := 1.00 (38.859) 257
-> ( "yjcS" ) := 1.00 (38.860) 258
-> ( "prc" ) := 1.00 (38.875) 259
-> ( "ppk" ) := 1.00 (38.973) 260
-> ( "yncD" ) := 1.00 (39.008) 261
-> ( "ydhV" ) := 1.00 (39.032) 262
-> ( "yfjK" ) := 1.00 (39.043) 263
-> ( "nrdD" ) := 1.00 (39.077) 264
-> ( "crfC" ) := 1.00 (39.099) 265
-> ( "arpA" ) := 1.00 (39.115) 266
-> ( "fhuE" ) := 1.00 (39.118) 267
-> ( "speC" ) := 1.00 (39.137) 268
-> ( "nmpC'" ) := 1.00 (39.213) 269
-> ( "opgB" ) := 1.00 (39.258) 270
-> ( "ybhJ" ) := 1.00 (39.269) 271
-> ( "xdhA" ) := 1.00 (39.281) 272
-> ( "bglX" ) := 1.00 (39.370) 273
-> ( "bisC" ) := 1.00 (39.447) 274
-> ( "yddB" ) := 1.00 (39.449) 275
```

```
-> ( "ycjT" ) := 1.00 (39.457) 276
-> ( "torZ" ) := 1.00 (39.507) 277
-> ( "ybgQ" ) := 1.00 (39.507) 278
-> ( "pheT" ) := 1.00 (39.525) 279
-> ( "plsB" ) := 1.00 (39.526) 280
-> ( "yehB" ) := 1.00 (39.562) 281
-> ( "napA" ) := 1.00 (39.562) 282
-> ( "alaS" ) := 1.00 (39.626) 283
-> ( "acnA" ) := 1.00 (39.722) 284
-> ( "elfC" ) := 1.00 (39.724) 285
-> ( "barA" ) := 1.00 (39.796) 286
-> ( "ileS" ) := 1.00 (39.847) 287
-> ( "ycgV" ) := 1.00 (39.886) 288
-> ( "fadE" ) := 1.00 (40.070) 289
-> ( "bcsC" ) := 1.00 (40.115) 290
-> ( "pfo" ) := 1.00 (40.192) 291
-> ( "evgS" ) := 1.00 (40.206) 292
-> ( "metH" ) := 1.00 (40.231) 293
-> ( "yejO'" ) := 1.00 (40.254) 294
-> ( "tamB" ) := 1.00 (40.266) 295
-> ( "rhsA" ) := 1.00 (40.320) 296
-> ( "rhsC" ) := 1.00 (40.330) 297
-> ( "rhsB" ) := 1.00 (40.339) 298
-> ( "yfhM" ) := 1.00 (40.447) 299
-> ( "yneO'" ) := 1.00 (40.459) 300
-> ( "yaiT'" ) := 1.00 (40.462) 301
-> ( "yeeJ" ) := 1.00 (40.483) 302
-> ( "ydbA'" ) := 1.00 (40.496) 303
```

As each query in *fizz* has a set *time-to-live*, if the value specified in the runtime configuration you are using is too low for the query to fully complete, the number of matching genes you will found will be lower than in the above example. 303 genes appears to be the correct answers.

Using a binary store for the raw DNA sequences

The addition of the *elemental* `MRKCSBFStore` in *fizz* 0.4 allows for the DNA sequences to be stored and readily available without occupying the host system's memory and with no loading time since we won't be saving nor loading the fragments (as statements) from a *fizz* file.

In this section, we are going to look at using this new feature. First we are going to create a new *fizz* file which we will call `fragz.fizz`. In it, we will setup the *elemental* as follow:

```
1  frag {
2
3      class       = MRKCSBFStore,
4      filepath    = "./fragz.sbfz",
5      offloaded   = yes,
6      index       = 0,
7      verbose     = yes,
8      no.match    = fail
9
10 } {}
```

The `index` property specifies that we want the statements to be stored in the binary store to be indexed based on the first *term* of each *statement*, which is the identifier of the DNA fragments. This will allow for a much faster retrieval of the fragments since this is the way we will be fetching them. Please refer to page 72 of *fizz* 's user manual for more details on the *elemental*'s properties.

We are now ready to import the DNA sequences using the same `import.fizz` we used before:

```
$ ./fizz.x64 ./etc/articles/e.coli/import.fizz ./etc/articles/e.coli/fragz.fizz
fizz 0.4.0-X (20180829.2211) [lnx.x64|8|w|l]
Press the ESC key at anytime for input prompt

load : loading ./etc/articles/e.coli/import.fizz ...
```

```
load : loading ./etc/articles/e.coli/fragz.fizz ...
load : loaded  ./etc/articles/e.coli/fragz.fizz in 0.001s
frag - (re)indexing ...
frag - (re)indexed in 0.00s
load : loaded  ./etc/articles/e.coli/import.fizz in 0.013s
load : loading completed in 0.013s
?- /import.txt("./etc/data/U00096.3.txt",line.f,1)
import.txt : 1000 lines parsed ...
import.txt : 2000 lines parsed ...
import.txt : 3000 lines parsed ...
...
import.txt : 60000 lines parsed ...
import.txt : 61000 lines parsed ...
import.txt : 61889 lines read in 0.331s.
frag - asserting 3072 statements
frag - asserted 3072 statements 0.000423 (3072.000000)
frag - asserting 3072 statements
frag - asserted 3072 statements 0.000429 (3072.000000)
frag - asserting 3072 statements
frag - asserted 3072 statements 0.000437 (3072.000000)
frag - asserting 3072 statements
frag - asserted 3072 statements 0.000437 (3072.000000)
frag - asserting 3072 statements
frag - asserted 3072 statements 0.000443 (3072.000000)
frag - asserting 3072 statements
frag - asserted 3072 statements 0.000449 (3072.000000)
frag - asserting 3072 statements
frag - asserted 3072 statements 0.000437 (3072.000000)
frag - asserting 3072 statements
frag - asserted 3072 statements 0.000443 (3072.000000)
frag - asserting 3072 statements
frag - asserted 3072 statements 0.000440 (3072.000000)
frag - asserting 3072 statements
frag - asserted 3072 statements 0.000442 (3072.000000)
frag - asserting 3072 statements
frag - asserted 3072 statements 0.000447 (3072.000000)
frag - asserting 3072 statements
frag - asserted 3072 statements 0.000450 (3072.000000)
frag - asserting 3072 statements
frag - asserted 3072 statements 0.000451 (3072.000000)
frag - asserting 3072 statements
frag - asserted 3072 statements 0.000455 (3072.000000)
frag - asserting 3072 statements
frag - asserted 3072 statements 0.000448 (3072.000000)
frag - asserting 3072 statements
frag - asserted 3072 statements 0.000457 (3072.000000)
frag - asserting 3072 statements
frag - asserted 3072 statements 0.000454 (3072.000000)
frag - asserting 3072 statements
frag - asserted 3072 statements 0.000455 (3072.000000)
frag - asserting 3072 statements
frag - asserted 3072 statements 0.000452 (3072.000000)
frag - asserting 3072 statements
frag - asserted 3072 statements 0.000454 (3072.000000)
frag - asserting 449 statements
frag - asserted 449 statements 0.000449 (449.000000)
```

Because we have set the `frag` *elemental* in a *verbose* mode, we are able to observe it ingesting the asserted *statements* and reporting the elapsed time in ingesting a single *statement* (which is here in the 400-500 microseconds range). Let check now if we have all the *statements* we were expecting:

```
?- /stats
stats : e:12 k:8 s:61889 p:15 u:63.55 t:0 q:0 r:0 z:61889
```

`61889` is the expected number of *statements*. We will now query the *elemental* just to check:

```
?- #frag(60001,:s)
-> ( "AGAACCCATCATGTGCAGGAAAATTATCTTCGGAGAGGATGTATCCGCCAGCGCACGTTCTGTTTCCTGTAACAA" ) := 1.00 (0.001) 1
```

If you compare the elapsed time for the query (1 milisecond) when using the binary store to the one we got earlier when using in-memory *statements*, you will notice that it is the same.

To complete the import, we will request from the *elemental* to optimize its content. Note that this won't have any significant impact on the *statement* retreival speed since we quering using the indexed *term*:

```
?- /tells(frag,optimize)
frag - optimizing ...
frag - optimized in 5.50s
```

While this takes a bit of time, it's is only suggested to be done on occasion, depending on how much changes is made to the store content. Finaly, as the store is backed by an actual binary file, there is no need to save the *elemental*. To access the *statements* at a later time, we just need to import the `fragz.fizz` file:

```
$ ./fizz.x64 ./etc/articles/e.coli/genes-i.fizz ./etc/articles/e.coli/fragz.fizz ./etc/articles/e.coli/base.fizz
fizz 0.4.0-X (20180829.2211) [lnx.x64|8|w|l]
Press the ESC key at anytime for input prompt

load : loading ./etc/articles/e.coli/genes-i.fizz ...
load : loading ./etc/articles/e.coli/fragz.fizz ...
load : loading ./etc/articles/e.coli/base.fizz ...
load : loaded  ./etc/articles/e.coli/fragz.fizz in 0.002s
load : loaded  ./etc/articles/e.coli/base.fizz in 0.014s
load : loaded  ./etc/articles/e.coli/genes-i.fizz in 1.528s
load : loading completed in 1.528s
?- #gene.offset("feaR",:o,:l,:or)
-> ( 1446377 , 906 , Counterclockwise ) := 1.00 (0.002) 1
```

If we were to load only the *fizz* files that we created earlier to store the 62k fragments, we would get a loading time of `1.797s` vs `0.001s` for the binary store. Using the store does, however, come at a cost as it is use a non-trivial amount of disk space. The `fragz.sbfz` we have created is over 32mb vs 5.6mb for all the *fizz* files that contains the *fragments*.