Library Customization

This Del Sequencing Package can be used for libraries featuring different numbers of encoded elements (encoding sequences, encoding ids, and building block smiles). Libraries with up to 3 building block elements and 2 pcr elements are currently supported. Theoretically this number can be expanded to for libraries with 4 or more BBs but doing so will require some knowledge of python.

This guide will walk through the process of updating the files located in the "library" subfolder to suit your library's specific parameters accordingly.

As a first step, you may want to read through the **Operation.pdf** document to get an idea of how the program works and gain some context, then come back here to continue this guide.

There are four .py files that will need to be updated when preparing to run the tool on a new library:

`fragment_codes.py` – correlates ID values with building block encoding sequences.

`fragment_smiles.py` – correlates ID values with building block smiles.

`primers_all.py` – correlates ID values with PCR codes for a full set of available PCR primers.

> -This file is not actively used by the program; it is just a useful starting point for removing any unused primers for a given screen and re-saving the file as "primers_screen.py"

`primers_screen.py` – correlates ID values with PCR codes for only the PCR primers used in a specific screen.

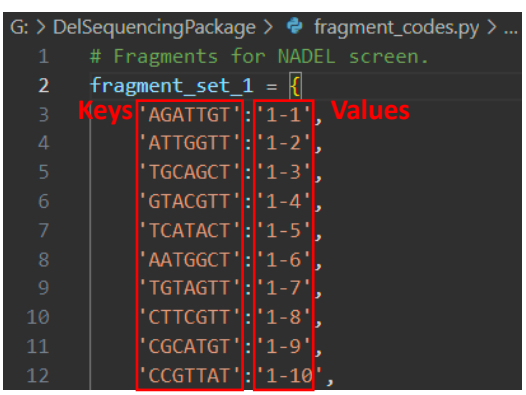
Updating the python dictionaries

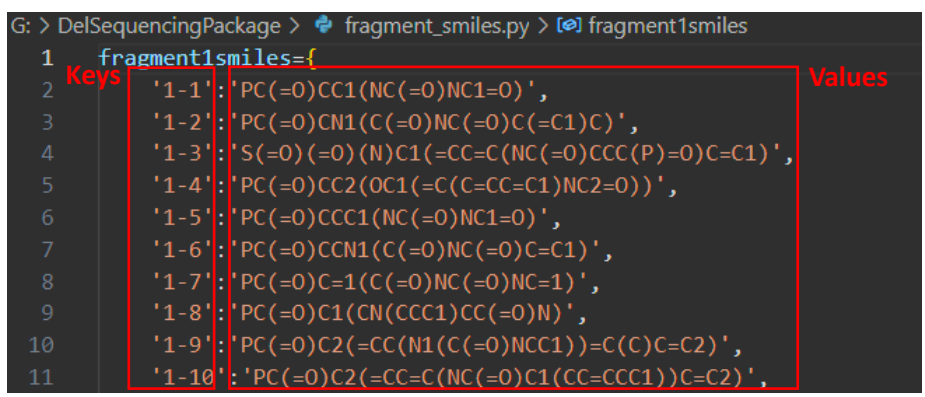
These files contain python dictionary objects that are loaded by the program. A dictionary is a python data structure the consists of a series of key:value pairs enclosed in a set of curly brackets {…}

Example:

`fruit_colors = {'strawberry':'red', 'lemon':'yellow', 'pear':'green'}`

Lets see how these dictionaries look inside these .py files...





The dark orange text in these files is what you need to update accordingly depending on the specific library you are using. Note how the keys and values are all enclosed in a set of single quotes (‘ ’). These quotes are required in python to represent textual information as the string data type. Keys and values must be separated with a colon “:” and key:value pairs are separated with a comma “,”.

Applying this format is arduous to do by hand at scale so I am including a helpful script located in the utils subfolder to make this process faster.

To use the script, follow these instructions:

1) Organize your codes and ids in excel like so (below, left) and save as a .csv file to the Del Sequencing Package’s utils folder.
2) Inside the utils folder, open **DictFormatter.ipynb** and change the inputs appropriately to match your csv data.
3) Run the code cell to generate a txt file (below, right) containing the csv data with proper formatting for use as a python dictionary.
4) Open the txt file and copy and paste the text into the .py file you would like to update to replace the contents of an existing dictionary. Note that the pasted text lines need to fall between the set of curly brackets { }.
5) Save the changes to the .py file.

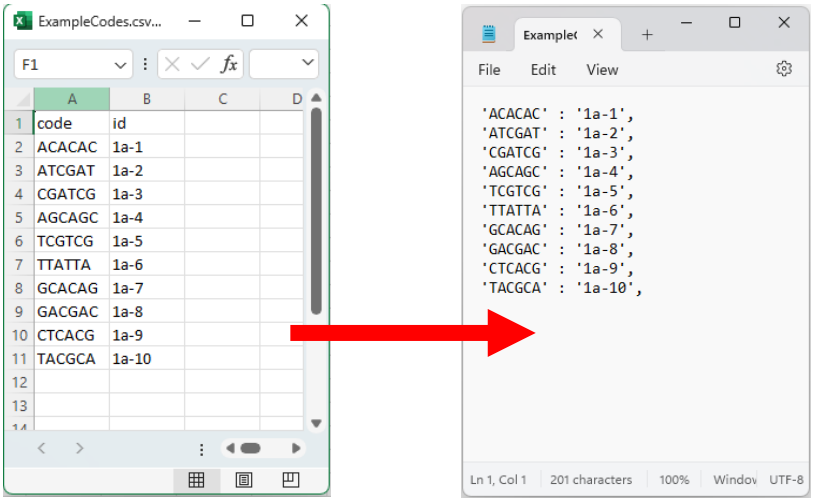
Avoid changing the names of the dictionaries themselves, otherwise you will need to also change them in the main program.

The **ExampleCodes.csv** file is provided as an example. <u>Note that your primer IDs should follow the convention used in the primer.py files for proper functionality with the main program.</u>

This naming convention is as follows (w/ no brackets): **`<primer set><fwd/rev primer>-<ID>`**

Where **`primer set`** is a single digit integer, **`fwd primer`** = ‘a’ and **`rev primer`** = ‘b’, and **`ID`** can be combination of letters and/or numbers.

Repeat this process with your smiles information and your PCR primer code



Updating the sequence search parameters

Inside the main **DelSceenProcessing.ipynb** notebook, you can change the parameters used for the sequence search algorithm.

There are 3 main input sections:

```
##### INPUTS #####
# (Change as needed according according to the library's encoding scheme)
# Note that under the current encoding scheme, the PCR1 code is the first 6 bp so searching for it is unnecessary.

# Provide a short name for organizing the results in the output folder
user_inputs['NAME']= 'TestBatch1'

#####################
### INDEX INPUTS  ###
#####################

#This should be the number of BP that precedes the first bb encoding region (BB1)
user_inputs['BB1_START_IDX'] = 30

#This should be the number of BP that precedes the second bb encoding region (BB2)
user_inputs['BB2_START_IDX'] = 51

#This should be the number of BP that precedes the second bb encoding region (BB3)
user_inputs['BB3_START_IDX'] = None

#This should be the number of BP that precedes the second pcr encoding region (PCR2)
user_inputs['PCR2_START_IDX'] = 75
```

The first main section handles the inputs used by the slicing algorithm. The inputs here are integers representing the position of the first encoding bp in the full sequence read. The **SeqView** notebook will help with determining the values to use for this section and is discussed below.

```
#############################
### OTHER REQUIRED INPUTS  ###
#############################

#The illumina adapter sequence used to preprocess the fastq file.
#Only reads containing this sequence will be extracted for analysis.
user_inputs['ADAPT_SEQ']='GATCGGAAGAGCACACGTCTG'

#Specify the length of the codes used for BB encoding
user_inputs['BB_ENCODING_LEN'] = 7

#Specify the length of the codes used for PCR encoding
user_inputs['PCR_ENCODING_LEN'] = 6

#Specify the max number of mismatch errors to allow in the encoding sequences
user_inputs['ENCODING_TOLERANCE'] = 1

# Define the path to your raw FASTQ file (This is the name of the file that will be submitted for preprocessing)
user_inputs['RAW_FASTQ_FILE'] = "ExampleData.fastq"

# Define the name you wish to use for saving the preprocessed FASTQ file (This will be the name of the preprocessing output
user_inputs['PROC_FASTQ_FILE'] = "Preprocessed_ExampleData.txt"
```

The next main section contains other inputs used by the program. The purpose of each parameter is described in the green comment text.

```
#######################
### OPTIONAL INPUTS  ###
#######################

#Set to "True" to test only the first N_TEST_SEQUENCES
#Set to 'False" to process the entire sequencing file
user_inputs['TEST_RUN']=False
user_inputs['N_TEST_SEQUENCES']=200000
```

The last (optional) section lets you perform a test run with the specified number of sequences in your fastq file. This can be nice to quickly check the program for proper functionality with any settings you may have changed.

Using SeqView

The SeqView notebook (**SeqView.ipynb**, located in the utils folder) is useful for visualizing sequence reads graphically and makes it easy to see where encoding sequences are located. This helps determine the slice points you should use as inputs for the encoding sequence search algorithms.

1) To use **SeqView**, you must first have your preprocessed fastq file which is generated as part of the **DelScreenProcessing** workflow. Note that the preprocessing step only uses the **ADAPT_SEQ** (illumina primer sequence) input and can be run even if you don't know what values to use for the other inputs yet.
2) Once you have generated the preprocessed .txt file, go ahead and open **SeqView.ipynb** in VS Code.

   You will see two input variables:

   PROC_FASTQ_FILE – The name of your preprocessed .txt file.

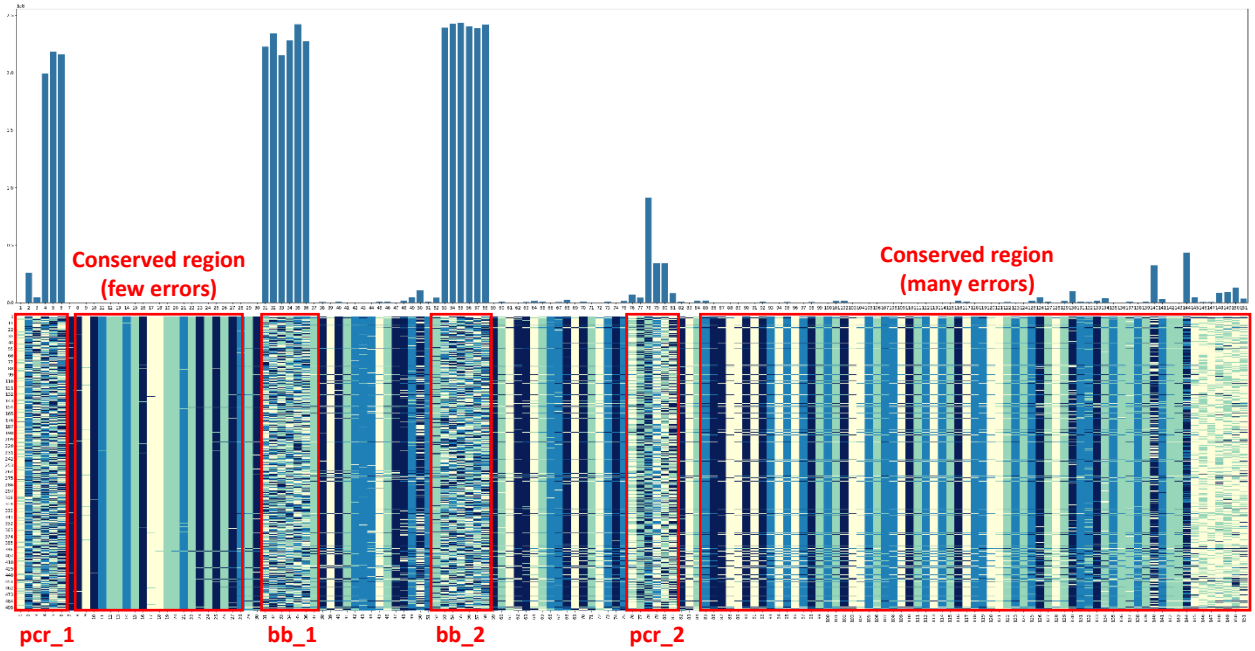
   N_sequences – The number of sequences to visualize (50-500 is the recommended range)

   Simply adjust these values to your liking and run the code cell. You should see a set of plots appear.

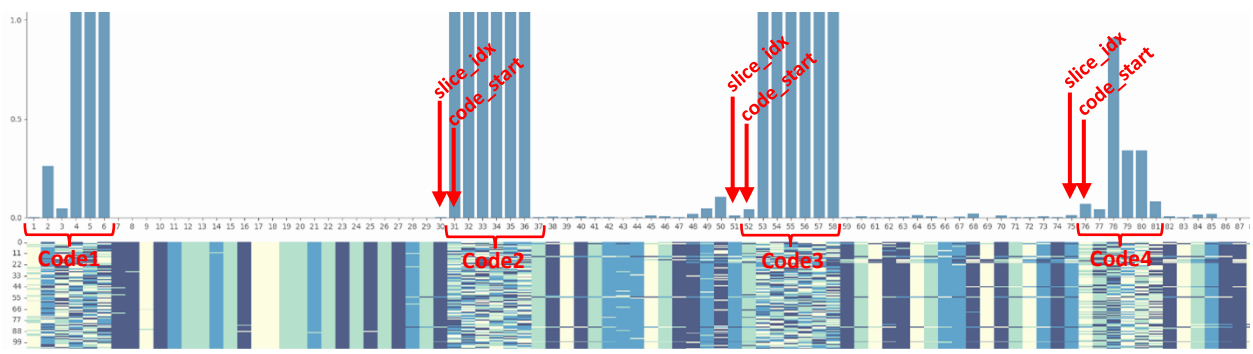
The top bar plot is a measure of variance at each position in the sequence read. High areas of variance should logically occur in encoding regions.

The heatmap plot under the barplot shows the collection of sequences colored according to nucleobase.

The heat map makes it easy to see areas of conservation, variance, and pcr/read errors.



Picking slicing indices



Note how cols 37 and 52 are depicted as being part of the encoding regions for code2 and code3, even though there is little variance at these positions. This is because the NADEL encoding sequences for code2 all end in “T” and almost all code3 sequences begin with a “T”. Therefore, these positions have low variance even though they are part of the encoding region, however they must still be considered to match the codes listed in the **fragment_codes.py** file.

Follow these steps:

1) Locate areas of high variance corresponding to the expected length of your encoding sequences.
2) For determining slice locations (**slice_idx**), choose the positions the *immediately precede* the start of an encoding region (**code_start**). 30, 51, and 75 are the slice points in NADEL.
3) Enter the slice_idx values in the input section for the main notebook.