

blapi.py code documentation

Two functions are included in blapi, they are getAllEntries() and getEntries()

Definition of getAllEntries():

```
def getAllEntries(accession, gene_id, product, location):
```

It takes the accession number, gene ID, protein product and location all in string format as input.

The data returned will be a list of dictionaries of all of the entries that matches the input(s).

If there is no input for a particular parameter, then 'None' will be received for that parameter from the front end, which will be passed on to the dbapi and that parameter will not be filtered. Therefore if no input was given to the front end listall.py, then all 4 parameters will be 'None', and all the records from the database will be returned.

Here is an example of the returned data structure -

```
[{'gene_id': 'HLA-DQA1',  
  'accession': 'AB006907',  
  'product': 'HMC class II surface glycoprotein',  
  'location': '6p21.3'},  
 {'gene_id': 'HLA-DQA1',  
  'accession': 'AB006908',  
  'product': 'MHC class II surface glycoprotein',  
  'location': '6p21.3'},  
 {'gene_id': 'HLA-DMA',  
  'accession': 'AB010385',  
  'product': 'HLA-DMA',  
  'location': '6p21.3'}]
```

Internally, this function calls dbapi.getAllEntries() with the same parameters and returns its output with no additional processing done. dbapi will then do the SQL query and returns all the records that matched all of the parameters.

This function is used in the listall.cgi script where

```
entries = blapi.getAllEntries(accession = accession, gene_id = gene_id, product = product, location  
= location)
```

Call tree for blapi.getAllEntries:

```
listall.py  →  blapi.getAllEntries()  →  dbapi.getAllEntries()
```

Second function:

```
def getEntry(accession, rez = ''):
```

This function takes the accession and rez parameters as strings. rez is an optional parameter.

The output is the specific record that matches the accession returned as a dictionary.

This function first takes the accession as a string returned from the front end, passes it to the dbapi, returning the record as a dictionary. Then the coding region of the DNA sequence is pre-tagged by this function with <mark> and </mark> tags for the front end.

On the detail page (search.py) of the chosen record displayed to the user, the restriction enzyme that could be used to cut the gene can then be chosen. Once the restriction enzyme (returned as a string) is passed to this function, the DNA sequence will be further processed where the location of the cutting site will be pre-tagged with '★' and returned back to the front end for display.

Additionally, the total codon frequency of all the records in chromosome 6 and the chosen record by the user will also be returned to the front end in the same dictionary.



Example data returned from dbapi to use this function :

```
{'gene_id': 'HLA-DQA1',
'accession': 'AB006907',
'product': 'HMC class II surface glycoprotein',
'location': '6p21.3',
'protein_seq': 'MILNKALMLGALALTTVMSPCGGEDIV',
'dna_seq': 'ATGATCCTAAACAAAGCTCTGATGCT<mark>GGGGGCCCTG</mark>CCCTGACCACCGTGATGAGCCCTGT<span class =
"re">&starf;</span>GGAGGTGAAGACATTGTGG',
'cds': '1..82',
'codon_freq': {'AAA': '0'..'GGG': '0'},
'total_codon_freq': {'AAA': '0'..'GGG': '0'}}
```

This function is used in the search.py script where

```
entry = blapi.getEntry(accession, rez)
```

Call tree for blapi.getEntry:

search.py  blapi.getEntry ()  dbapi.getEntry ()

For both functions:

accession = Genbank accession number, typically Abxxxxxx (x = numbers) and accession is expected to be unique, so it is used as a primary in the database.

gene_id = Gene identifiers

location = Chromosomal location

product = Protein product names

rez = Restriction enzyme