Laboratory 6: more beginning Python3 (BLAST & FASTA processing)

This exercise is designed to teach simple Python3 regular expressions, built–in functions, recursion, file parsing, and use of external programs. Today you will create a script ('extractBLAST.py') that exhaustively queries a DNA seed sequence against a BLAST database and extracts all the matching sequences (this overcomes limitations imposed by inconsistent annotation and sub–optimal search tools). The instruction for creating the script assume that you are using a programming oriented text editor (e.g. Visual Studio Code) that provides line numbers (starting from 1). A complete script is provided at the end of this document for your reference. You should save and test your work often. Small errors can prevent the script from working properly and it is easiest to catch them if you try to run the script frequently.

Tasks

(1) First, install all of the packages and modules that will be required by the script:

   (a) Install the pip package installer by typing `sudo apt install python3-pip` in the terminal. Type your password and agree to the install when prompted.

   (b) Install the Python3 modules ftfy and xxhash by typing `pip3 install ftfy` followed by `pip3 install xxhash` in the terminal.

   (c) Install BLAST+ and a Python3 wrapper for it by typing `sudo apt install ncbi-blast+ python3-biopython` in the terminal. Type your password, if prompted.

(2) To use and test the script, you will need to obtain a BLAST database of all Cupressaceae subfamily Callitroideae sequences deposited in GenBank and a seed sequence:

   (a) To download the sequences, first create a polite bash download function by typing `downloadGenBank() { MINWAIT=3; MAXWAIT=7; sleep $((MINWAIT+RANDOM % (MAXWAIT-MINWAIT))); X=$(esearch -db nuccore -query "\"$1\"[Organism]" | efetch -format fasta); printf "${X}\n"; }` in the terminal.

   (b) Export your download function by typing `export -f downloadGenBank` in the terminal.

   (c) Create a directory to store the sequences in by typing `mkdir Callitroideae` in the terminal.

   (d) Download the sequences, two genera at a time, using your download function with xargs and save them in the Callitroideae directory by typing `echo -n 'Austrocedrus Callitris Diselma Fitzroya Libocedrus Papuacedrus Pilgerodendron Widdringtonia' | tr ' ' '\0' | xargs -0 -I {} -P 2 bash -c 'downloadGenBank "{}" > Callitroideae/{}.fasta'` in the terminal.

   (e) Check that there are 8 FASTA files in the Callitroideae directory and that each contains DNA sequences.

   (f) Convert the FASTA format sequence files to a BLAST database by typing `cat Callitroideae/*.fasta | makeblastdb -dbtype nucl -input_type fasta -parse_seqids -hash_index -title Callitroideae -out Callitroideae/Callitroideae` in the terminal.

   (g) Check that 10 new files matching the pattern 'Callitroideae/Callitroideae.n*' have been created.

(h) Extract a seed *matK* sequence belonging to *Callitris pyramidalis* contained in the BLAST database by typing `blastdbcmd -db Callitroideae/Callitroideae -dbtype nucl -entry MW470972 -strand minus -range 110480-112015 > Callitroideae/seed.fasta` in the terminal.

(i) Check that the seed sequence file contains 1,536 bases and 1,634 total characters.

(3) Open a text editor and create a blank file.

(4) Enter the first 12 lines of the script:

```
#!/usr/bin/env python3

### IMPORTS
import Bio.Blast.Applications
import ftfy
import getopt
import multiprocessing
import os
import re
import textwrap
import sys
import xxhash
```

(5) Save the file as 'extractBLAST.py'. Answer question (1).

(6) Make the script executable by typing `chmod 0755 extractBLAST.py` in the terminal.

(7) Test the script (it does nothing yet) by typing `./extractBLAST.py` in the terminal. Not getting an error message means all is well.

(8) Add a set of global constants starting at line 14 (line 13 is intentionally blank):

```

### GLOBAL CONSTANTS
BLASTFILE = 'temporary-do-not-touch'
COLUMNS = 80 ### .fasta line width
DEFLINE = re.compile('^>')
DNA = re.compile('[^ABCDGHKMNRSTVWY]')
EVALUE = 0.01
INTRA = re.compile('(convar\.|f\.|forma|grex|lusus|microgene|modif\.|monstr\.|mut\.|' \
   'nothosubsp\.|nothovar\.|proles|provar\.|spp\.|spp|stirps|subf\.|sublusus|subproles|' \
   'subso|subsp\.|subvar\.|var\.|var)' \
)
PIPE = re.compile('\|')
QUERYFILE = 'temporary-query-do-not-touch'
TARGETS = 1000
WRAP = int(os.popen('stty size', 'r').read().split()[1])
XML = re.compile('<|>')

### GLOBAL USER SETTINGS AND DEFAULTS
settings = {}
settings['cores'] = multiprocessing.cpu_count()
settings['database'] = ''
settings['seed'] = ''
```

(9) Test the script for errors (it still does nothing). Answer question (2).

(10) Next insert a block of functions starting at line 36 (line 35 is intentionally blank):

```
35
36 ### FUNCTIONS
37 def breakLines(sequence):
38    output = ''
39    for position, base in enumerate(sequence):
40       if position > 0 and position % COLUMNS == 0:
41          output += '\n'
42       output += base
43    return output
44
45 def eprint(*arguments, **keywordArguments):
46    print(*arguments, file = sys.stderr, **keywordArguments)
47
48 def seedStore(seed, sequence):
49    if len(sequence) > 0:
50       cleanSequence = re.sub(DNA, '', sequence.upper())
51       seed[xxhash.xxh128_hexdigest(cleanSequence)] = cleanSequence
52
53 def species(name): ### assumes properly formed names (not always true)
54    words = name.split(' ')
55    if len(words) < 3: ### Genus | Genus specific
56       return name
57    elif testWords(words, 0, 'x'): ### x Genus ...
58       return f"× {species(' '.join(words[1:]))}"
59    elif testWords(words, 1, 'x'): ### Genus x specific ...
60       return f'{words[0]} × {specificEpithet(words[2:])}'
61    elif len(words) >= 4 and testWords(words, 2, 'x'): ### Genus specific x specific ...
62       return f'{words[0]} {words[1]} × {specificEpithet(words[3:])}'
63    else: ### Genus specificEpithet ...
64       return f'{words[0]} {specificEpithet(words[1:])}'
65
66 def specificEpithet(words):
67    if len(words) == 0:
68       return ''
69    elif len(words) >= 3 and re.search(INTRA, words[1]): ### specific intra intraspecific ...
70       return f'{words[0]} {words[1]} {specificEpithet(words[2:])}'
71    else: ### specificEpithet ...
72       return words[0]
73
74 def testWords(words, index, value):
75    try:
76       return words[index] == value
77    except IndexError:
78       return False
79
80 def wrap(string, columns = WRAP):
81    return '\n'.join(textwrap.wrap(string, columns))
```

(11) Test the script for errors after entering each function (it still does nothing). Answer question (3).

(12) Parse the user supplied arguments by inserting the following code starting on line 83 (line 82 is intentionally blank):

```
82
83 ### READ OPTIONS
84 databaseError = wrap('Database to be searched (required): -d name | --database=name')
85 seedError = wrap('Search seed (required): -s file.fasta | --seed=file.fasta')
86 try:
```

```
87     arguments, values = getopt.getopt(
88        sys.argv[1:],
89        'c:d:hs:',
90        ['cores=', 'database=', 'help', 'seed=']
91     )
92  except getopt.error as error:
93     eprint(str(error))
94     sys.exit(2)
95  for argument, value in arguments:
96     if argument in ('-c', '--cores') and int(value) > 0:
97        settings['cores'] = int(value)
98     elif argument in ('-d', '--database'):
99        extensions = ['ndb', 'nhd', 'nhi', 'nhr', 'nin', 'nos', 'not', 'nsq', 'ntf']
100       allFiles = True
101       for extension in extensions:
102          if os.path.isfile(f'{value}.{extension}') == False:
103             eprint(wrap(f"Database file '{value}.{extension}' is missing!"))
104             allFiles = False
105             break
106       if allFiles:
107          settings['database'] = value
108    elif argument in ('-h', '--help'):
109       eprint('')
110       eprint(wrap('A Python3 script for exhaustively BLASTing a seed nucleotide sequence.'))
111       eprint(wrap(f"Cores for BLAST+ (optional; default = {settings['cores']}):" \
112          ' -c int | --cores=int'
113       ))
114       eprint(databaseError)
115       eprint(seedError)
116       eprint('')
117       sys.exit(0)
118    elif argument in ('-s', '--seed'):
119       if os.path.isfile(value):
120          settings['seed'] = value
121       else:
122          eprint(wrap(f"Seed file '{value}' does not exist!"))
123          sys.exit(2)
```

(13) Test the script for errors. The help option should work, but the script does nothing else.

(14) Insert some code to check if the user has provided the appropriate input by inserting the following code starting on line 125 (line 124 is intentionally blank):

```
124
125 ### START OR END
126 if not settings['database']:
127    eprint(databaseError)
128    sys.exit(2)
129 elif not settings['seed']:
130    eprint(seedError)
131    sys.exit(2)
132 else:
133    eprint('started...')
134    for key, value in settings.items():
135       eprint(f'{key} = {value}')
```

(15) Test the script for errors. Each of the user options should work now (try passing the name of the seed sequence file and the BLAST database), but the script does not actually conduct an analysis. Answer question (4).

(16) Insert the following code to read the seed FASTA file starting on line 137 (line 136 is intentionally blank):

```
136
137 ### LOAD INITIAL SEED
138 seed = {}
139 sequence = ''
140 with open(settings['seed'], mode = 'rt', encoding = 'utf8', errors = 'replace') as file:
141    for line in file:
142       text = ftfy.fix_text(line).rstrip()
143       if re.search(DEFLINE, text):
144          seedStore(seed, sequence)
145          sequence = ''
146       else:
147          sequence += text
148 seedStore(seed, sequence)
```

(17) Test the script for errors. The script does not yet conduct the BLAST search, but it should read a seed file if you provide it with a valid file. Answer question (5).

(18) Insert the following code to conduct the BLAST search(s) starting on line 150 (line 149 is intentionally blank):

```
149
150 ### QUERY LOOP
151 data = {}
152 while len(seed) > 0:
153    output = open(QUERYFILE, 'w')
154    for hash, sequence in seed.items():
155       output.write(f'>{hash}\n')
156       output.write(breakLines(sequence) + '\n')
157    output.close()
158    eprint(f'{len(seed)} sequences for BLAST search')
159
160    os.system(str(Bio.Blast.Applications.NcbiblastnCommandline(
161       db = settings['database'],
162       evalue = EVALUE,
163       max_target_seqs = TARGETS,
164       num_threads = settings['cores'],
165       out = BLASTFILE,
166       outfmt = 5,
167       query = QUERYFILE
168    )))
169    os.unlink(QUERYFILE)
170    eprint('BLAST search complete')
171
172    accession = ''
173    seed = {}
174    sequence = ''
175    taxon = ''
176    with open(BLASTFILE, mode = 'rt', encoding = 'utf8', errors = 'replace') as file:
177       for line in file:
178          tokens = re.split(XML, ftfy.fix_text(line).strip())
179          if testWords(tokens, 1, 'Hit_num'):
180             accession = ''
181             sequence = ''
182             taxon = ''
183          elif testWords(tokens, 1, 'Hit_def'):
184             taxon = species(tokens[2])
```

```
185            elif testWords(tokens, 1, 'Hit_id'):
186               if re.search(PIPE, tokens[2]):
187                  accession = tokens[2].split('|')[1]
188               else:
189                  accession = tokens[2]
190            elif testWords(tokens, 1, 'Hsp_hseq'):
191               sequence = re.sub(DNA, '', tokens[2].upper())
192               key = f'>{accession} {taxon}'
193               if not key in data or (key in data and len(data[key]) < len(sequence)):
194                  data[key] = sequence
195                  seedStore(seed, sequence)
196      os.unlink(BLASTFILE)
```

(19) Test the script for errors. The script can now conduct the BLAST search, but it will not yet output the extracted sequences. Answer question (6).

(20) To add the code that outputs extracted sequences, insert the following starting on line 198 (line 197 is intentionally blank):

```
197
198   ### OUTPUT
199   for accessionTaxon, sequence in data.items():
200      print(accessionTaxon)
201      print(breakLines(sequence))
202
203   sys.exit(0)
```

(21) Test the script for errors.

(22) Run the complete script by typing `./extractBLAST.py -d Callitroideae/Callitroideae -s Callitroideae/seed.fasta > output.fasta` in the terminal. You should have extracted 171 sequences. Confirm this via a grep search. Answer question (7).

Questions (https://forms.gle/xdQ8ubX5WpDqiQzbA)

(1) For task (5), what does 'import' do?

(2) For task (9):

   (a) What characters will the DNA regular expression match to?

   (b) What will the default value be for '`settings['cores']`'?

   (c) Will it have the same value for every computer?

(3) For task (11):

   (a) What does the modulus (%) operator inside the 'breakLines' function do?

   (b) What would you change to make the script output 128 bases per line?

   (c) Why does the 'specificEpithet' function call itself in some circumstances?

   (d) Would the 'specificEpithet' function be able to properly parse a name like '*Miconia quadrangularis* f. *latifolia nervulosa*'? Explain.

(4) For task (15):

(a) If the user supplies all of the required options, what will be printed by line 135?
(b) Will ‘cores’ be printed even if the user does not specify that option?

(5) For task (17):

(a) Why does the ‘seedStore’ function not return anything?
(b) Were do the input data go?
(c) What would happen if two identical seed sequences were included in the seed file?

(6) For task (19), what must happen to break the while loop that controls the BLAST search?

(7) For task (22):

(a) What grep search did you perform?
(b) How can you be sure that it provides an accurate count?

Due at the start of class March 7.

Final extractBLAST.py. code

```
#!/usr/bin/env python3

### IMPORTS
import Bio.Blast.Applications
import ftfy
import getopt
import multiprocessing
import os
import re
import textwrap
import sys
import xxhash

### GLOBAL CONSTANTS
BLASTFILE = 'temporary-do-not-touch'
COLUMNS = 80 ### .fasta line width
DEFLINE = re.compile('^>')
DNA = re.compile('[^ABCDGHKMNRSTVWY]')
EVALUE = 0.01
INTRA = re.compile('(convar\.|f\.|forma|grex|lusus|microgene|modif\.|monstr\.|mut\.|' \
   'nothosubsp\.|nothovar\.|proles|provar\.|spp\.|spp|stirps|subf\.|sublusus|subproles|' \
   'subso|subsp\.|subvar\.|var\.|var)' \
)
PIPE = re.compile('\|')
QUERYFILE = 'temporary-query-do-not-touch'
TARGETS = 1000
WRAP = int(os.popen('stty size', 'r').read().split()[1])
XML = re.compile('<|>')

### GLOBAL USER SETTINGS AND DEFAULTS
settings = {}
settings['cores'] = multiprocessing.cpu_count()
settings['database'] = ''
settings['seed'] = ''

### FUNCTIONS
def breakLines(sequence):
   output = ''
   for position, base in enumerate(sequence):
      if position > 0 and position % COLUMNS == 0:
         output += '\n'
      output += base
   return output

def eprint(*arguments, **keywordArguments):
   print(*arguments, file = sys.stderr, **keywordArguments)

def seedStore(seed, sequence):
   if len(sequence) > 0:
      cleanSequence = re.sub(DNA, '', sequence.upper())
      seed[xxhash.xxh128_hexdigest(cleanSequence)] = cleanSequence

def species(name): ### assumes properly formed names (not always true)
   words = name.split(' ')
   if len(words) < 3: ### Genus | Genus specific
      return name
```

```
   elif testWords(words, 0, 'x'): ### x Genus ...
      return f"× {species(' '.join(words[1:]))}"
   elif testWords(words, 1, 'x'): ### Genus x specific ...
      return f'{words[0]} × {specificEpithet(words[2:])}'
   elif len(words) >= 4 and testWords(words, 2, 'x'): ### Genus specific x specific ...
      return f'{words[0]} {words[1]} × {specificEpithet(words[3:])}'
   else: ### Genus specificEpithet ...
      return f'{words[0]} {specificEpithet(words[1:])}'

def specificEpithet(words):
   if len(words) == 0:
      return ''
   elif len(words) >= 3 and re.search(INTRA, words[1]): ### specific intra intraspecific ...
      return f'{words[0]} {words[1]} {specificEpithet(words[2:])}'
   else: ### specificEpithet ...
      return words[0]

def testWords(words, index, value):
   try:
      return words[index] == value
   except IndexError:
      return False

def wrap(string, columns = WRAP):
   return '\n'.join(textwrap.wrap(string, columns))

### READ OPTIONS
databaseError = wrap('Database to be searched (required): -d name | --database=name')
seedError = wrap('Search seed (required): -s file.fasta | --seed=file.fasta')
try:
   arguments, values = getopt.getopt(
      sys.argv[1:],
      'c:d:hs:',
      ['cores=', 'database=', 'help', 'seed=']
   )
except getopt.error as error:
   eprint(str(error))
   sys.exit(2)
for argument, value in arguments:
   if argument in ('-c', '--cores') and int(value) > 0:
      settings['cores'] = int(value)
   elif argument in ('-d', '--database'):
      extensions = ['ndb', 'nhd', 'nhi', 'nhr', 'nin', 'nos', 'not', 'nsq', 'ntf']
      allFiles = True
      for extension in extensions:
         if os.path.isfile(f'{value}.{extension}') == False:
            eprint(wrap(f"Database file '{value}.{extension}' is missing!"))
            allFiles = False
            break
      if allFiles:
         settings['database'] = value
   elif argument in ('-h', '--help'):
      eprint('')
      eprint(wrap('A Python3 script for exhaustively BLASTing a seed nucleotide sequence.'))
      eprint(wrap(f"Cores for BLAST+ (optional; default = {settings['cores']}):" \
         ' -c int | --cores=int'
      ))
      eprint(databaseError)
      eprint(seedError)
```

```
      eprint('')
      sys.exit(0)
   elif argument in ('-s', '--seed'):
      if os.path.isfile(value):
         settings['seed'] = value
      else:
         eprint(wrap(f"Seed file '{value}' does not exist!"))
         sys.exit(2)

### START OR END
if not settings['database']:
   eprint(databaseError)
   sys.exit(2)
elif not settings['seed']:
   eprint(seedError)
   sys.exit(2)
else:
   eprint('started...')
   for key, value in settings.items():
      eprint(f'{key} = {value}')

### LOAD INITIAL SEED
seed = {}
sequence = ''
with open(settings['seed'], mode = 'rt', encoding = 'utf8', errors = 'replace') as file:
   for line in file:
      text = ftfy.fix_text(line).rstrip()
      if re.search(DEFLINE, text):
         seedStore(seed, sequence)
         sequence = ''
      else:
         sequence += text
seedStore(seed, sequence)

### QUERY LOOP
data = {}
while len(seed) > 0:
   output = open(QUERYFILE, 'w')
   for hash, sequence in seed.items():
      output.write(f'>{hash}\n')
      output.write(breakLines(sequence) + '\n')
   output.close()
   eprint(f'{len(seed)} sequences for BLAST search')

   os.system(str(Bio.Blast.Applications.NcbiblastnCommandline(
      db = settings['database'],
      evalue = EVALUE,
      max_target_seqs = TARGETS,
      num_threads = settings['cores'],
      out = BLASTFILE,
      outfmt = 5,
      query = QUERYFILE
   )))
   os.unlink(QUERYFILE)
   eprint('BLAST search complete')

   accession = ''
   seed = {}
   sequence = ''
```

```
   taxon = ''
   with open(BLASTFILE, mode = 'rt', encoding = 'utf8', errors = 'replace') as file:
      for line in file:
         tokens = re.split(XML, ftfy.fix_text(line).strip())
         if testWords(tokens, 1, 'Hit_num'):
            accession = ''
            sequence = ''
            taxon = ''
         elif testWords(tokens, 1, 'Hit_def'):
            taxon = species(tokens[2])
         elif testWords(tokens, 1, 'Hit_id'):
            if re.search(PIPE, tokens[2]):
               accession = tokens[2].split('|')[1]
            else:
               accession = tokens[2]
         elif testWords(tokens, 1, 'Hsp_hseq'):
            sequence = re.sub(DNA, '', tokens[2].upper())
            key = f'>{accession} {taxon}'
            if not key in data or (key in data and len(data[key]) < len(sequence)):
               data[key] = sequence
               seedStore(seed, sequence)
   os.unlink(BLASTFILE)

### OUTPUT
for accessionTaxon, sequence in data.items():
   print(accessionTaxon)
   print(breakLines(sequence))

sys.exit(0)
```