DNAとRNAの配列の作成・読み込みおよびファイルの書き換えについてのマニュアルを作成した。

学習できること

1. (multi-)FASTA形式の配列について

2. ファイルへのデータの書き込み, 読み込み

3. Biopythonを用いた配列の取り扱い, 計算

4. 関数の作り方

5. リスト内包表記

Biopythonという分子生物学系の計算に特化したpythonのパッケージがあるため, 今回はBiopythonを使用して配列を取り扱う
まずはBiopythonをインストールする

コマンドラインで

pip install biopython

と打ち込む (https://biopython.org/wiki/Download (https://biopython.org/wiki/Download))

multi-FASTA形式のファイルを作って読み込み, 計算し, 改変する

1. multi-FASTA形式のファイルを作成する

DNAなどの配列情報が保存されている形式にFASTA形式がある.
FASTA形式では > 記号の後に配列の名前, 説明などが記録され, 次の行から配列が記録されている.
sampleDNAという名前で長さ20 (length=20というdescriptionもあり) のDNAをFASTA形式で表記した例を以下に示す.

```
>sampleDNA length=20
AAGGCCTTAAGGCCTTAAGG
```

multi-FASTA形式は以下の例に示すように複数のFASTA形式から構成されている.

>sampleDNA length=20
AAGGCCTTAAGGCCTTAAGG
>sampleDNA2 length=22
AAGGCCTTAAGGCCTTAAGGCC
>sampleDNA3 length=24
AAGGCCTTAAGGCCTTAAGGCCTT

次に, これらの配列をリストに書き込んだ後ファイルに書き込む.

まずそれぞれの配列の説明をseqDescsというリストに書き込む

In [1]:
```
seqDescs = ["sampleDNA length=20", "sampleDNA2 length=22", "sampleDNA3 length=24"]
```

次にそれぞれの配列をseqsというリストに書き込む

In [2]:
```
seqs = ["AAGGCCTTAAGGCCTTAAGG", "AAGGCCTTAAGGCCTTAAGGCC", "AAGGCCTTAAGGCCTTAAGGCCTT"]
```

これらのリストに含まれる配列とdescriptionをsampleDNAs.fastaというファイルに書き込む

In [3]:
```
f = open("/Users/Sora 1/Desktop/sampleDNAs.fasta", "w") # 一つ目の""にはファイルのpathを入れる. 二つ目の""にはw (writeモード) を入れ, 書き込み可能な状態で開く
for i in range(len(seqDescs)): #seqDescsの要素数だけ繰り返す
    f.write(f">{seqDescs[i]}\n{seqs[i]}\n") #>の後にseqDescsのi番目の要素を書く. \n で改行した後にseqsのi番目の要素を書き込む. (Pythonでは"\n"が改行を示す) 次の>に備えて\nで改行する.
f.close() # openしたので閉じる.
```

ここで, fから始まる行ではファイルを書き込み(write)モードで開いている. 指定したディレクトリにファイルが存在しなければ新たに作成する.
次のfor文では, 回数にseqDescsのリストの要素数を指定しているため, seqDescsの数が変動しても対応した回数だけ繰り返す.
これにより, 配列の個数と同じ回数だけloopを繰り返すことができる.

2. 作った配列を読み込み, 中身を確認する

まず, BiopythonからSeqIOというモジュールをimportする

In [4]:
```
from Bio import SeqIO
```

先ほど作ったファイルを開き, 中身を要素ごとにprintすることで確認する

In [5]:
```
with open("/Users/Sora 1/Desktop/sampleDNAs.fasta") as handle: #""の間にファイルのpathを記入する. handleでファイルを開くことができる. withで開くとcloseの必要がない
    for record in SeqIO.parse(handle, "fasta"):  #fasta形式で開き, 配列の数だけ以下を繰り返す.
        print(record.id) # recordのid, すなわち>の後の最初のスペースまでを取り出す
        print(record.description) # recordのdescription, すなわち>の行の>以外を全て取り出す
        print(record.seq) # recordのsequence, すなわち>の次の行を取り出す
        print() #見やすくするためにrecordの前に一行空ける
        print(record) #recordの中身を見る
        print("\n") #見やすくするために各配列間に二行空ける.
```

```
sampleDNA
sampleDNA length=20
AAGGCCTTAAGGCCTTAAGG

ID: sampleDNA
Name: sampleDNA
Description: sampleDNA length=20
Number of features: 0
Seq('AAGGCCTTAAGGCCTTAAGG', SingleLetterAlphabet())


sampleDNA2
sampleDNA2 length=22
AAGGCCTTAAGGCCTTAAGGCC

ID: sampleDNA2
Name: sampleDNA2
Description: sampleDNA2 length=22
Number of features: 0
Seq('AAGGCCTTAAGGCCTTAAGGCC', SingleLetterAlphabet())


sampleDNA3
sampleDNA3 length=24
AAGGCCTTAAGGCCTTAAGGCCTT

ID: sampleDNA3
Name: sampleDNA3
Description: sampleDNA3 length=24
Number of features: 0
Seq('AAGGCCTTAAGGCCTTAAGGCCTT', SingleLetterAlphabet())
```

SeqIOによってファイルの読み書きが可能になっている.
各エントリー(配列)がSeqRecord型のオブジェクトとして保存されており, for文により一つ一つを取り出し, それぞれの成分をprintすることで中身を確認している.

3. それぞれの配列のTm値, GC率を計算する

BiopythonからTm値を計算するためのMeltingTempとGC率を計算するためのGCをimportする

In [6]:
```
from Bio.SeqUtils import MeltingTemp as mt #mtとしてimportしたため, コード上で使用する際にはmtで十分(下記コード参照)
from Bio.SeqUtils import GC
```

作成したファイルから配列を取り出し, Tm値, GC率を計算して一つの配列の情報をtupleに入れ, それらのtupleを要素にもつlistを作成する

In [7]:
```
with open("/Users/Sora 1/Desktop/sampleDNAs.fasta") as handle: #作成したファイルを開く
    Tm = [] #配列とそれらの情報を書き込むためのリストを用意する
    for record in SeqIO.parse(handle, "fasta"): #record(配列)の個数だけ以下を繰り返す
        myseq = record.seq #配列をmyseqに格納
        meltTemp = '%0.2f' % mt.Tm_NN(myseq, nn_table=mt.DNA_NN1) #myseqのTm値を計算. "%0.2f"で小数点以下2桁までのfloat(小数)が返るようにしている.
        GCcontent = GC(myseq) #GCcontentに配列のGC率を格納
        desc = record.description #descに配列の説明を格納
        seqInfo = desc, meltTemp, GCcontent, myseq #これらを順番通りに要素にもつtupleとしてseqInfoを作成
        Tm.append(seqInfo) #TmというlistにseqInfoを要素として追加
print(Tm)
```

```
[('sampleDNA length=20', '62.11', 50.0, Seq('AAGGCCTTAAGGCCTTAAGG'
, SingleLetterAlphabet())), ('sampleDNA2 length=22', '68.20', 54.5
4545454545455, Seq('AAGGCCTTAAGGCCTTAAGGCC', SingleLetterAlphabet(
))), ('sampleDNA3 length=24', '69.57', 50.0, Seq('AAGGCCTTAAGGCCTT
AAGGCCTT', SingleLetterAlphabet()))]
```

なお, 今回はDNA-DNA二本鎖のTm値を Breslauer '86 をもとに算出したが, MeltingTempモジュールを用いて Sugimoto '96 をもとに算出したり, DNA-RNA二本鎖, RNA-RNA二本鎖のTm値を求めることも可能である. (BiopythonのHPを参照)

また, meltTempに代入する値は "%0.2f"の2を他の数字に変えることで小数点以下の桁数を変更できる(例: "%0.3f"にすると小数点以下3桁まで計算).
同様にGC(myseq)の前に "%0.2f %" を代入してGC率の小数点以下を二桁のみ表示することも可能である.

4. 計算した値を書き加えた新しいファイルを作成する

listであるTm内のそれぞれのtupleから情報を抽出して, fileに書き込む

In [8]:
```
f = open("/Users/Sora 1/Desktop/modSampleDNAs.fasta", "w")
for items in Tm:
    f.write(f">{items[0]} Tm={items[1]} GCcontent={items[2]}\n{items[3]}\n")
f.close()
```

Tm内の各tupleをitemsとして取り出し, tuple内の要素をitems[]で抽出し, 共通の文字列("Tm="や"GCcontent="など)と組み合わせて書き込んでいる.
同一の名前のファイルがすでに存在する場合には新しく作成されたファイルに上書きされるが, w (=write) の代わりに a (=append) を使用すると既存のファイルに書き足すことができる.

これにより作成されるファイルの中身は以下の通りである.

```
>sampleDNA length=20 Tm=62.11 GCcontent=50.0
AAGGCCTTAAGGCCTTAAGG
>sampleDNA2 length=22 Tm=68.20 GCcontent=54.54545454545455
AAGGCCTTAAGGCCTTAAGGCC
>sampleDNA3 length=24 Tm=69.57 GCcontent=50.0
AAGGCCTTAAGGCCTTAAGGCCTT
```

先ほどと同様にSeqIOを用いてファイルを読み込めることを確認する

In [9]:
```
with open("/Users/Sora 1/Desktop/modSampleDNAs.fasta") as handle:
    for record in SeqIO.parse(handle, "fasta"):
        print(record.id)
        print(record.description)
        print(record.seq)
        print()
        print(record)
        print()
```

```
sampleDNA
sampleDNA length=20 Tm=62.11 GCcontent=50.0
AAGGCCTTAAGGCCTTAAGG

ID: sampleDNA
Name: sampleDNA
Description: sampleDNA length=20 Tm=62.11 GCcontent=50.0
Number of features: 0
Seq('AAGGCCTTAAGGCCTTAAGG', SingleLetterAlphabet())

sampleDNA2
sampleDNA2 length=22 Tm=68.20 GCcontent=54.54545454545455
AAGGCCTTAAGGCCTTAAGGCC

ID: sampleDNA2
Name: sampleDNA2
Description: sampleDNA2 length=22 Tm=68.20 GCcontent=54.5454545454
5455
Number of features: 0
Seq('AAGGCCTTAAGGCCTTAAGGCC', SingleLetterAlphabet())

sampleDNA3
sampleDNA3 length=24 Tm=69.57 GCcontent=50.0
AAGGCCTTAAGGCCTTAAGGCCTT

ID: sampleDNA3
Name: sampleDNA3
Description: sampleDNA3 length=24 Tm=69.57 GCcontent=50.0
Number of features: 0
Seq('AAGGCCTTAAGGCCTTAAGGCCTT', SingleLetterAlphabet())
```

5. 無作為に作成した配列から特定の条件を満たすものだけを取得する

以上では便宜的に作った配列を使用したが, 実際にはデータベース上の配列を使用したり, 自らデザインした配列に対してこれらの作業を行うことが想定される.
その場合には望みの特徴を持つ配列のみを取得することが望ましい.
したがって, 望みの特徴を持つ配列を選別する関数を作成し, 取得する.

def 関数名(引数):
引数に対する処理

によって任意の関数を作成することができる.
引数には任意の数設定することができるが, 今回はseqという一つの引数に含まれる配列が

1. 単一の塩基を4連続以上で持たないこと
2. GC率が35%以上65%以下であること

を満たすと True を返すqualityCheckという名前の関数を作成する.

In [10]:
```
def qualityCheck(seq): #"seq"という引数に対して以下の処理を行う"qualityFilter"という関数を定義(define)する
    if(seq.find("AAAA") == -1 and seq.find("TTTT") == -1 and seq.find("GGGG") == -1 and seq.find("CCCC") == -1): # 4 塩基以上の単一の連続配列を持たない配列を選択 (目的の配列が見つからない時find関数は-1を返す)
        if 35 <= GC(seq) <= 65: # GC content が 35 % 以上 65 % 以下のものを選択
            return True
```

20 ntのランダムな配列を作成し, qualityCheck関数を用いて目的の特徴を持つ配列のみを選別する.

In [11]:
```
import random # randomモジュールを使用してランダム配列を作成

n_code = {0: "A", 1: "T", 2: "G", 3:"C"} # keyに0から3, valueにA, T, G, Cを持つ辞書 n_code を作成
i = 0 # i の初期値を設定
goodSeqs = [] # qualityCheck == True の時の配列を格納するリスト
badSeqs = {} # qualityCheck != True の時の配列を格納する. 同時にGC率も格納したいため辞書型
while i < 100: # i が100未満の時
    seq = "".join(n_code[random.randint(0,3)] for j in range(25)) #25塩基のランダム配列を作る.
    #0から3のランダムな整数に対応するn_codeのvalueを出力することを25回繰り返した後 join関数でつなげる.
    if qualityCheck(seq) == True:
        goodSeqs.append(seq) # 目的の特徴を持つ配列はgoodSeqsに追加.
        i += 1 #目的の特徴を持つ配列が作れた時のみ i を1増やす
    else: #どの程度の割合で目的の特徴を持たない配列があるか確認するためのコード
        badSeqs[seq] = GC(seq) # 目的の特徴を持たない配列はbadSeqsに追加
        # 4nt以上の連続配列があるのか、GC率が高すぎるor低すぎるのかがわかりやすくなるようGC率もvalueとして格納
```

qualityCheckがTrueとなるような配列を作成した回数をカウントし, 100個作成した際に配列の作成を終了させるよう, while文を書いた.
seq の生成にはリストの内包表記を用いた.
これは

```
bases = []
for j in range(25):
bases.append(n_code[random.randint(0,3)])
seq = "".join(bases)
```

と同一の処理をしているが, 内包表記を用いると一行で簡潔に書け, また速度も速い.

また, qualityCheckで条件を満たした配列はgoodSeqsに記録したが, 条件を満たさなかった配列はbadSeqsに記録した.
以下でそれらのデータを実際に確認する.

In [12]:
```
print(len(goodSeqs)) #すべてprintすると多すぎるため, 100個作成できていることを確認
print(goodSeqs[0:10]) #最初の10個の配列を確認
print(list(GC(goodSeqs[i]) for i in range(0,10))) #最初の10個の配列のGC率をリストで表示
print(len(badSeqs)) #100個のgoodSeqsを作成する間に作成されたbadSeqsの数を確認
print(badSeqs) # 実際に中身を見てみる
```

```
100
['AATTCGTGAGATCTCTAGCTAACAA', 'AAGCCTCAGAATTCATACGCCAGAC', 'CAACAG
TCAACTCAGCACAAAGGAT', 'AGCCACCCAAGTTGCGCGCTGTTTA', 'CATGTTGCGGCTGA
CGCGATAGTAC', 'CCCGCTATACATAATCTAACGTGTA', 'GCCATACATCGATCTACATTGA
CGT', 'AGTGACAGACTTGGCTGTCACTTGT', 'TAAGGATTCCTCTGCCTTTCTACGG', 'A
TGTAAGTACCGCAGATTCCGAGAT']
[36.0, 48.0, 44.0, 56.0, 56.0, 40.0, 44.0, 48.0, 48.0, 44.0]
42
{'CCGCGCGTGCAACCATTGACCCGCA': 68.0, 'GCCCCTATTCTAGGGCCTTTGTATG': 5
2.0, 'TGCACTTTTAAATGTTGAGCTGCGG': 44.0, 'CTGTATTGGGGTTGGTCCATCTTGG
': 52.0, 'CTACCTCGTTTTAGTAGCGGACACG': 52.0, 'CACTTCCCCTTACCACTGGAC
AGCT': 56.0, 'TCACCATGAGGGCGCCCCGCGTCAC': 72.0, 'GAAGTTTTCAAGCCGGA
TCGCGCAT': 52.0, 'CAAAGTTATTTTTCCGGGAAATGAA': 32.0, 'TTATTGCCCTATC
TCTAAATGAAGT': 32.0, 'CGCTACACTTTTGCGGACAGTCTAT': 48.0, 'GACCCCTGT
GTATTTGATTGTCTTA': 40.0, 'TGAGGGGGGTCCAGCGTGCCGAACT': 68.0, 'TTTGT
TTTGTGCGATCACGATGTCT': 40.0, 'CATACTTAACTCCCCGTCTTTGCGT': 48.0, 'T
GGATCCTGACATATTGGGGAACGA': 48.0, 'AACGAGACAGGTAGAATGTGGGGTA': 48.0
, 'ACCGCAAGTACAAAATATACCCGAT': 40.0, 'GGCGACGGTCAGCTGGCCCAGACAT':
68.0, 'AGATCCCTAACTACCCCTCCGCAGG': 60.0, 'CTCGAAATCGTGGATTGGGAAAAA
A': 40.0, 'GATGCCATAAAACTTTCCGAGCTTA': 40.0, 'TCCATTAAGAATCTCGTTTA
AATGC': 32.0, 'TCCTTCGGGCCTTATTCCCCACGTT': 56.0, 'ACTCACTATGGATTTT
TGGTGGCAG': 44.0, 'GCGTACTCCGGAACGCCGCGACTGA': 68.0, 'GAATCGCAGGGA
GACTGGGGTCTCA': 60.0, 'GGCCTCGGTTTTACAGCGTAGTGGT': 56.0, 'CGAGTTTT
CGAAGACGTCGGTAAGT': 48.0, 'ATATCCAACCCCACGCATAGCGAAA': 48.0, 'TGCA
GGGCAGGGGCGTGGCAGGCAG': 76.0, 'AAGATCGCAATCAGTGGCAAAATCT': 40.0, '
CCGTCACAGGGACTACGTCCGGAGG': 68.0, 'GTTAGTGTCGGGGTCAATCGCTCAC': 56.
0, 'ACGCTTCGGGGTTGCCAGTGAGCCT': 64.0, 'ATGATGTGGGAGACAGGGGCCCACA':
60.0, 'CCCGTTCGATTCTACCGCATTTTAA': 44.0, 'AATAAGTCACACTTAGAATGGGGG
G': 44.0, 'AGACGACACGGGGGCTCCTGGTTGC': 68.0, 'TTGATCTAATAGATATCCAT
ACCTG': 32.0, 'TACACCACCGATATGGGGGGGAGTG': 60.0, 'TGATGATGCCGGTTTT
ACTGTTCGT': 44.0}
```

以上の通り, goodSeqsは100個作成されており, そのうち最初の10の配列に関しては条件を満たしていることが確認できた.
また, badSeqsには4以上の単一塩基の連続もしくは異常なGC率, あるいはその両方が確認された.
以上より, ランダムに生成した配列から目的の特徴を持つ配列を選別できたことを確認できた.

参照:

https://biopython.org/wiki/Download (https://biopython.org/wiki/Download)
https://bi.biopapyrus.jp/python/biopython/seqio.html
(https://bi.biopapyrus.jp/python/biopython/seqio.html)
https://biopython.org/wiki/SeqIO (https://biopython.org/wiki/SeqIO)
http://biopython.org/DIST/docs/api/Bio.SeqUtils.MeltingTemp-module.html
(http://biopython.org/DIST/docs/api/Bio.SeqUtils.MeltingTemp-module.html)
http://biopython.org/DIST/docs/api/Bio.SeqUtils-module.html
(http://biopython.org/DIST/docs/api/Bio.SeqUtils-module.html)
http://yukke.hateblo.jp/entry/2015/10/05/120924 (http://yukke.hateblo.jp/entry/2015/10/05/120924)
https://qiita.com/y__sama/items/a2c458de97c4aa5a98e7
(https://qiita.com/y__sama/items/a2c458de97c4aa5a98e7)