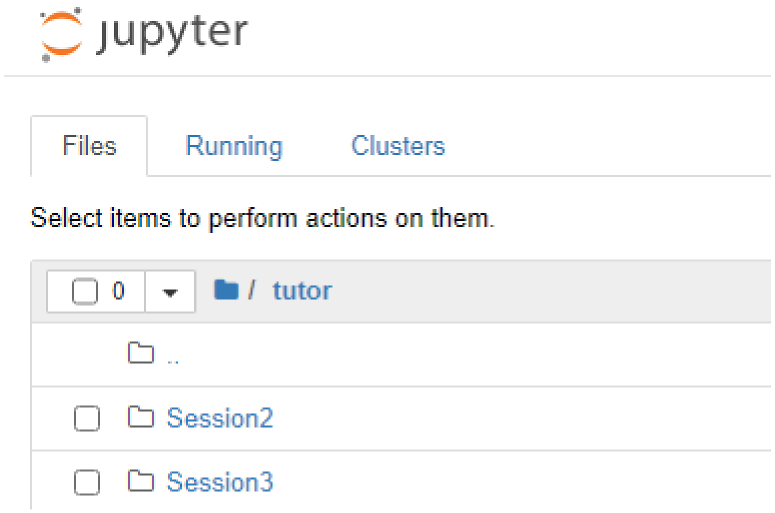
BIOINFORMATICS
SESSION 3 PRACTICE

2023-09-18

Cutting and ligating DNA

Contents

Make Session3 directory



Select items to perform actions on them.

0 / tutor

..

Session2

Session3

Basic Python - if statement

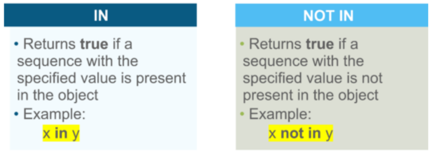
```
$ python_conditional.ipynb
```

```
>>> if 1==1:    print(“true”)
>>> else:       print(“false”)
```

→ true

Membership Operators

```
>>> dna = 'AACGGAATTCCCTCTC'
>>> if 'GAATTC' in dna:
>>>     print('match’)
```



IN

- Returns **true** if a sequence with the specified value is present in the object
- Example:
  x **in** y

NOT IN

- Returns **true** if a sequence with the specified value is not present in the object
- Example:
  x **not in** y

→ match

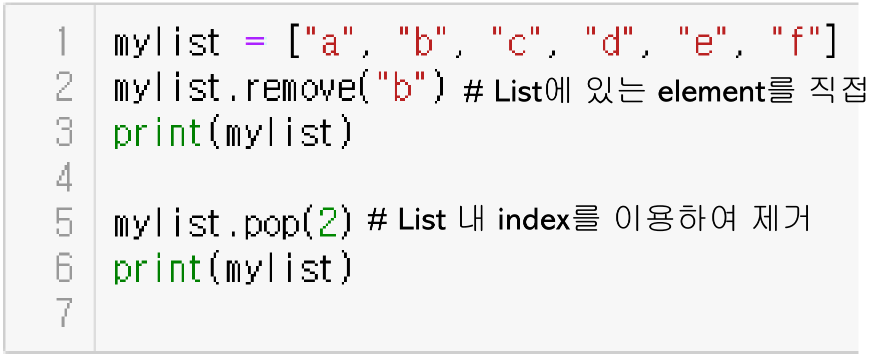
Basic Python - if statement

```
$ python_conditional.ipynb
```

```
>>> a = 2
>>> if a == 1:
        print(“a is 1”)
>>> elif a == 2:
        print(“a is 2”)
>>> else:
        print(“a is not 1 or 2”)
```

→ true

Basic Python – list



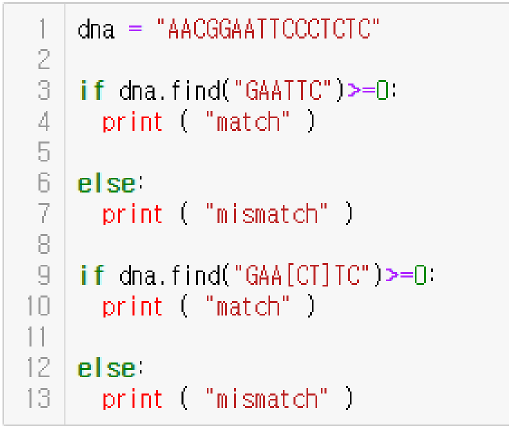
```python
mylist = ["a", "b", "c", "d", "e", "f"]
mylist.remove("b") # List에 있는 element를 직접 명시하여 제거
print(mylist)

mylist.pop(2) # List 내 index를 이용하여 제거
print(mylist)

```

```
['a', 'c', 'd', 'e', 'f']
['a', 'c', 'e', 'f']
```

Pattern matching

```
$ patternMatch.ipynb
```

string_of_interest.find(target_seq) : string_of_interest 에 target_seq가 존재하는 첫번째 index. 만약에 없으면 -1이 반환됨



```
dna = "AACGGAATTCCCTCTC"

if dna.find("GAATTC")>=0:
  print ( "match" )

else:
  print ( "mismatch" )

if dna.find("GAA[CT]TC")>=0:
  print ( "match" )

else:
  print ( "mismatch" )
```

```
match
mismatch
```

obviously ‘testsequence’ will match ‘testsequence’

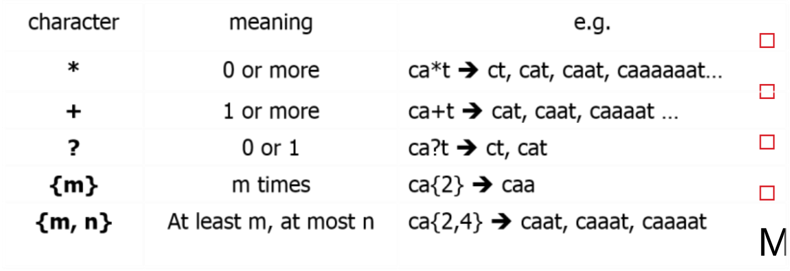
then what about ‘testsequence1’? or ‘Testsequence’? or ‘test_sequence’?

Basic Python – regular expressions

regular expression is a formal language consisting of words whose letters are taken according to a specific set of rules

```
. ^ $ * + ? { } [ ] \ | ( ) #meta characters
```

```
0<x<3 = 1,2
3! = 3x2x1
4C2 = 4!/2!(4-2)!
```



character	meaning	e.g.
*	0 or more	ca*t ➔ ct, cat, caat, caaaaaat...
+	1 or more	ca+t ➔ cat, caat, caaaat ...
?	0 or 1	ca?t ➔ ct, cat
{m}	m times	ca{2} ➔ caa
{m, n}	At least m, at most n	ca{2,4} ➔ caat, caaat, caaaat

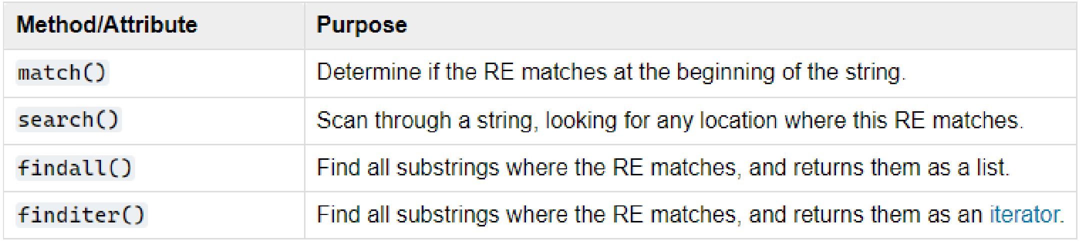
Examples

- [CT]: C or T
- [^CT] : not C and not T
- [CT][AG]: C or T and A or G
- [AB*]: A or AB or ABB or ABBB, …
- [AB+]: AB or ABB or ABBB,…
- [AB?]: A or AB
- A{6}: AAAAAA
- A{4,6}: AAAA, AAAAA, AAAAAA

More..

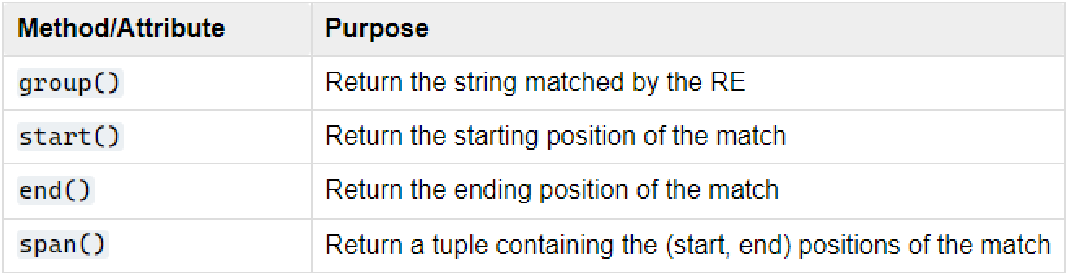
Basic Python – re module

performing pattern matching - if match: return match_object; else: return None type
∟<re.Match object; span=(0, 5), match='abcde'>



Method/Attribute	Purpose
`match()`	Determine if the RE matches at the beginning of the string.
`search()`	Scan through a string, looking for any location where this RE matches.
`findall()`	Find all substrings where the RE matches, and returns them as a list.
`finditer()`	Find all substrings where the RE matches, and returns them as an iterator.

querying match object



Method/Attribute	Purpose
`group()`	Return the string matched by the RE
`start()`	Return the starting position of the match
`end()`	Return the ending position of the match
`span()`	Return a tuple containing the (start, end) positions of the match

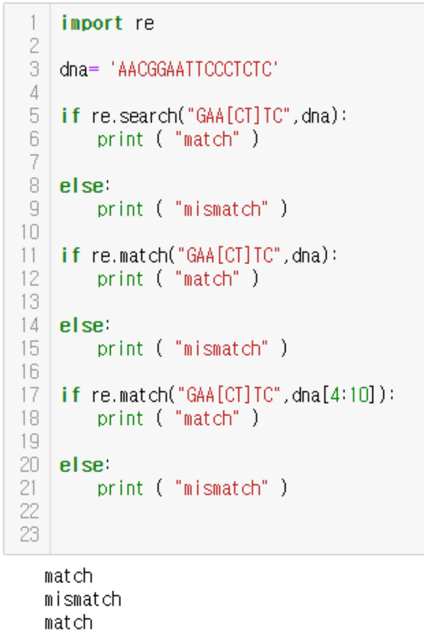
Regular Expression HOWTO — Python 3.11.5 documentation

Basic Python – re module

Method	Description
re.**search**(*pattern*, *string*, *flags=0*)	**Scan through *string* looking for <u>the first location where the regular expression *pattern* produces a match</u>**, and return a corresponding **MatchObject** instance. Return None if no position in the string matches the pattern; note that this is different from finding a zero-length match at some point in the string.
re.**match**(*pattern*, *string*, *flags=0*)	**If zero or more characters <u>at the beginning of *string*</u> match the regular expression *pattern***, return a corresponding **MatchObject** instance. Return None if the string does not match the pattern; note that this is different from a zero-length match.
re.**findall**(*pattern*, *string*, *flags=0*)	**Return <u>all non-overlapping matches of *pattern*</u> in *string*, <u>as a list of strings</u>**. The *string* is scanned left-to-right, and matches are returned in the order found. If one or more groups are present in the pattern, return a list of groups; this will be a list of tuples if the pattern has more than one group. Empty matches are included in the result unless they touch the beginning of another match.

Pattern matching

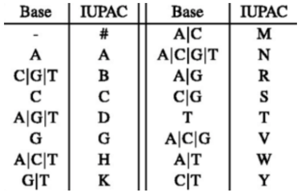
```
$ rePattern.ipynb
```



```
import re

dna= 'AACGGAATTCCCTCTC'

if re.search("GAA[CT]TC",dna):
    print ( "match" )

else:
    print ( "mismatch" )

if re.match("GAA[CT]TC",dna):
    print ( "match" )

else:
    print ( "mismatch" )

if re.match("GAA[CT]TC",dna[4:10]):
    print ( "match" )

else:
    print ( "mismatch" )
```

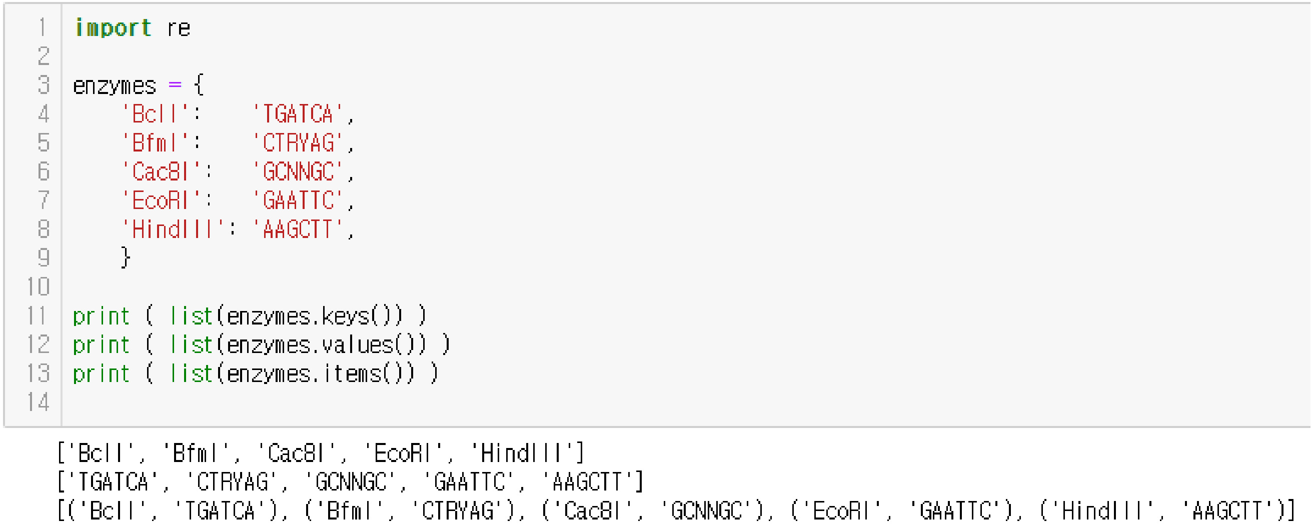
```
match
mismatch
match
```

Basic Python – Dictionary usage



Base	IUPAC	Base	IUPAC
-	#	A|C	M
A	A	A|C|G|T	N
C|G|T	B	A|G	R
C	C	C|G	S
A|G|T	D	T	T
G	G	A|C|G	V
A|C|T	H	A|T	W
G|T	K	C|T	Y

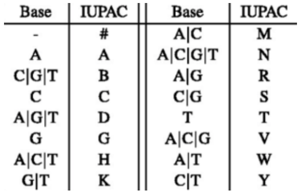
cut.ipynb



```python
import re

enzymes = {
    'BclI':    'TGATCA',
    'BfmI':    'CTRYAG',
    'Cac8I':   'GCNNGC',
    'EcoRI':   'GAATTC',
    'HindIII': 'AAGCTT',
    }

print ( list(enzymes.keys()) )
print ( list(enzymes.values()) )
print ( list(enzymes.items()) )

```

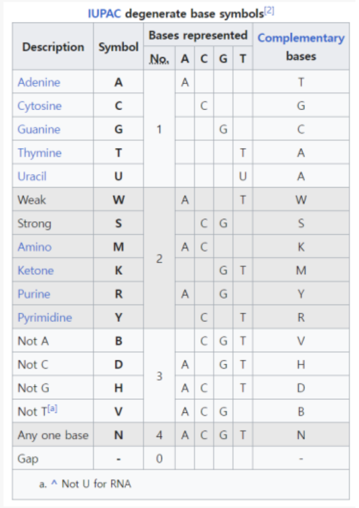
```
['BclI', 'BfmI', 'Cac8I', 'EcoRI', 'HindIII']
['TGATCA', 'CTRYAG', 'GCNNGC', 'GAATTC', 'AAGCTT']
[('BclI', 'TGATCA'), ('BfmI', 'CTRYAG'), ('Cac8I', 'GCNNGC'), ('EcoRI', 'GAATTC'), ('HindIII', 'AAGCTT')]
```

Basic Python – Dictionary usage



Base	IUPAC	Base	IUPAC
-	#	A|C	M
A	A	A|C|G|T	N
C|G|T	B	A|G	R
C	C	C|G	S
A|G|T	D	T	T
G	G	A|C|G	V
A|C|T	H	A|T	W
G|T	K	C|T	Y

```
print ( enzymes )
enzymes_mod = enzymes.copy() # be careful not to use enzymes_mod = enzymes
print ( enzymes_mod )

amb = {
    'R': '[AG]',
    'Y': '[CT]',
    'N': '[AGCT]',
    'W': '[AT]',
    'M': '[AC]',
    'S': '[CG]',
    'K': '[TG]',
    'V': '[ACG]',
    'H': '[ACT]',
    'D': '[AGT]',
    'B': '[CGT]',
    }

for key in enzymes_mod.keys():
    for ambkey in amb.keys():
        enzymes_mod[key] = enzymes_mod[key].replace(ambkey, amb[ambkey])

print ( enzymes_mod )
```



IUPAC degenerate base symbols[2]

Description	Symbol	Bases represented					Complementary bases
		No.	A	C	G	T	
Adenine	A	1	A				T
Cytosine	C			C			G
Guanine	G				G		C
Thymine	T					T	A
Uracil	U					U	A
Weak	W	2	A			T	W
Strong	S			C	G		S
Amino	M		A	C			K
Ketone	K				G	T	M
Purine	R		A		G		Y
Pyrimidine	Y			C		T	R
Not A	B	3		C	G	T	V
Not C	D		A		G	T	H
Not G	H		A	C		T	D
Not T[a]	V		A	C	G		B
Any one base	N	4	A	C	G	T	N
Gap	-	0					-

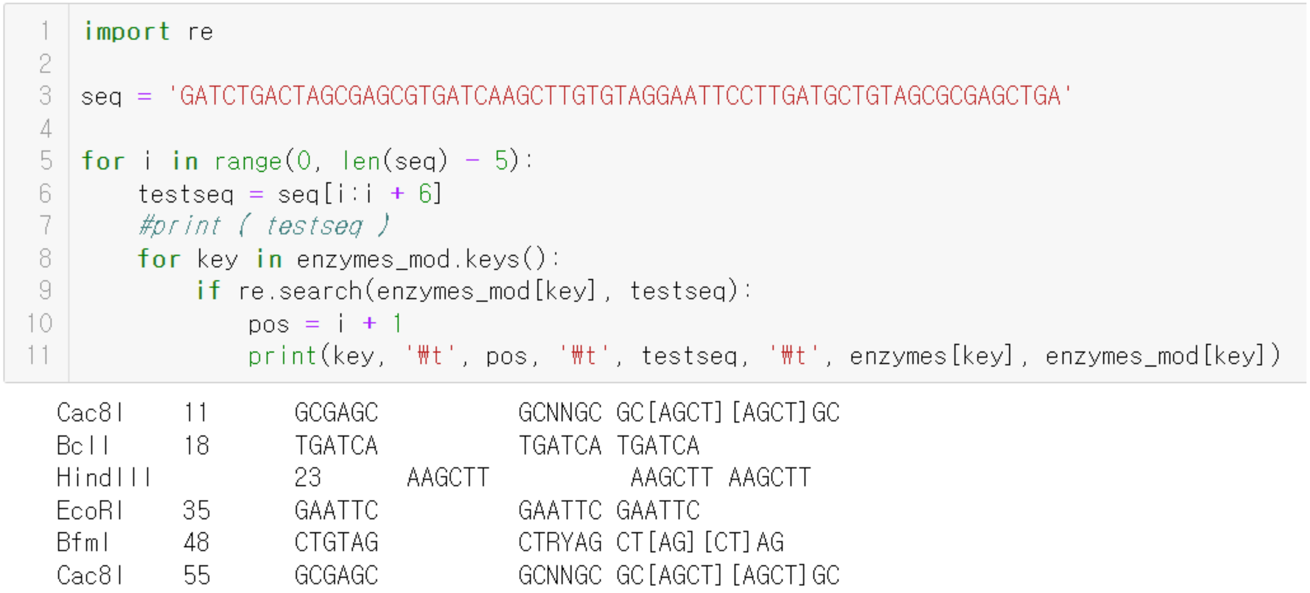
a. ^ Not U for RNA

en.wikipedia.org/wiki/Nucleic_acid_notation

```
{'BclI': 'TGATCA', 'BfmI': 'CTRYAG', 'Cac8I': 'GCNNGC', 'EcoRI': 'GAATTC', 'HindIII': 'AAGCTT'}
{'BclI': 'TGATCA', 'BfmI': 'CTRYAG', 'Cac8I': 'GCNNGC', 'EcoRI': 'GAATTC', 'HindIII': 'AAGCTT'}
{'BclI': 'TGATCA', 'BfmI': 'CT[AG][CT]AG', 'Cac8I': 'GC[AGCT][AGCT]GC', 'EcoRI': 'GAATTC', 'HindIII': 'AAGCTT'}
```

Code2.1 cut.py

seq = 'GATCTGACTAGCGAGCGTGATCAAGCTTGTGTAGGAATTCCTTGATGCTGTAGCGCGAGCTGA’



```python
import re

seq = 'GATCTGACTAGCGAGCGTGATCAAGCTTGTGTAGGAATTCCTTGATGCTGTAGCGCGAGCTGA'

for i in range(0, len(seq) - 5):
    testseq = seq[i:i + 6]
    #print ( testseq )
    for key in enzymes_mod.keys():
        if re.search(enzymes_mod[key], testseq):
            pos = i + 1
            print(key, '₩t', pos, '₩t', testseq, '₩t', enzymes[key], enzymes_mod[key])
```

```
Cac8I     11        GCGAGC          GCNNGC  GC[AGCT][AGCT]GC
BclI      18        TGATCA          TGATCA  TGATCA
HindIII             23       AAGCTT         AAGCTT  AAGCTT
EcoRI     35        GAATTC          GAATTC  GAATTC
BfmI      48        CTGTAG          CTRYAG  CT[AG][CT]AG
Cac8I     55        GCGAGC          GCNNGC  GC[AGCT][AGCT]GC
```

Exercise 1

- Assume you want to analyse the same sequence as in Code 2.1, but instead you are interested in identifying recognition sites of the two enzymes *Alu* and *DpnI*. These enzymes recognize sequences AGCT and GATC, repectively. Modify Code 2.1 to achieve this analysis

Exercise script and result

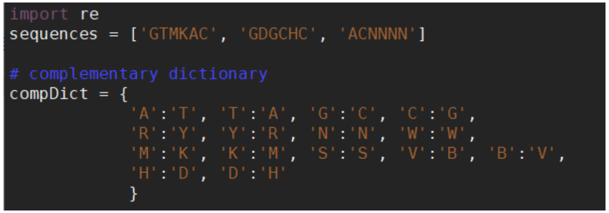
```python
import re

enzymes = {'Alu':"AGCT", "DpnI": "GATC"}

enzymes_mod = enzymes.copy()

seq = 'GATCTGACTAGCGAGCGTGATCAAGCTTGTGTAGGAATTCCTTGATGCTGTAGCGCGAGCTGA'

for i in range(0, len(seq) - 3):
    testseq = seq[i:i + 4]
    for key in enzymes_mod.keys():
        if re.search(enzymes_mod[key], testseq):
            pos = i + 1
            print ( key, '₩t', pos, '₩t', testseq, '₩t', enzymes[key] )
```

```
DpnI    1       GATC    GATC
DpnI    19      GATC    GATC
Alu     24      AGCT    AGCT
Alu     58      AGCT    AGCT
```

Assignment

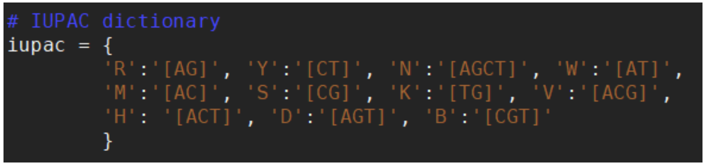
1) Print the reverse complement of the following three restriction enzyme recognition sequences according to IUPAC nucleotide degenrate code, RE_A : GTMKAC, RE_B: GDGCHC, and RE_C : ACNNNN. For instance, the symbol R is either A or G. The complementary bases in that case are T and C, and these may be represented by Y. Therefore, the 'complement' of R is Y. 2) And convert the reverse complement sequence into regular expression using the iupac dictionary below.

1. Reverse complement of GTMKAC, GDGCHC, and ACNNNN



```
import re
sequences = ['GTMKAC', 'GDGCHC', 'ACNNNN']

# complementary dictionary
compDict = {
            'A':'T', 'T':'A', 'G':'C', 'C':'G',
            'R':'Y', 'Y':'R', 'N':'N', 'W':'W',
            'M':'K', 'K':'M', 'S':'S', 'V':'B', 'B':'V',
            'H':'D', 'D':'H'
            }
```

2. Convert to regular expression using iupac dictionary



```
# IUPAC dictionary
iupac = {
        'R':'[AG]', 'Y':'[CT]', 'N':'[AGCT]', 'W':'[AT]',
        'M':'[AC]', 'S':'[CG]', 'K':'[TG]', 'V':'[ACG]',
        'H': '[ACT]', 'D':'[AGT]', 'B':'[CGT]'
        }
```

Assignment

3. Are there reverse complement sequence of restriction enzyme A,B,C present in "*seq*" from code2.1? If so, print out 1)- position, 2) actual sequence of enzyme recognition in "seq" 3)- original enzyme recognition sequences.

seq = 'GATCTGACTAGCGAGCGTGATCAAGCTTGTGTAGGAATTCCTTGATGCTGTAGCGCGAGCTGA'

- 과제 제출 기한: 09/24 Sunday 23:59 @ LMS
- 해당 코드 캡처를 한 뒤 워드에 첨부. 기입 하고 코드에 대한 설명 간략히 작성
  워드 파일명은 n주차_학번_이름 형식으로 제출(e.g. 3주차_2023123456_김현우)