STRING MATCHING ALGORITHMS: PRACTICE WITH PYTHON

DAVIDE BOLOGNINI

POSTDOCTORAL RESEARCHER

DEVELOPING @ HTTPS://GITHUB.COM/DAVIDEBOLO1993

DAVIDE.BOLOGNINI@UNIFI.IT

CONTENTS

1. JUPYTER NOTEBOOK

Project Jupyter is a nonprofit organization created to develop open-source software, open-standards, and services for interactive computing across dozens of programming languages

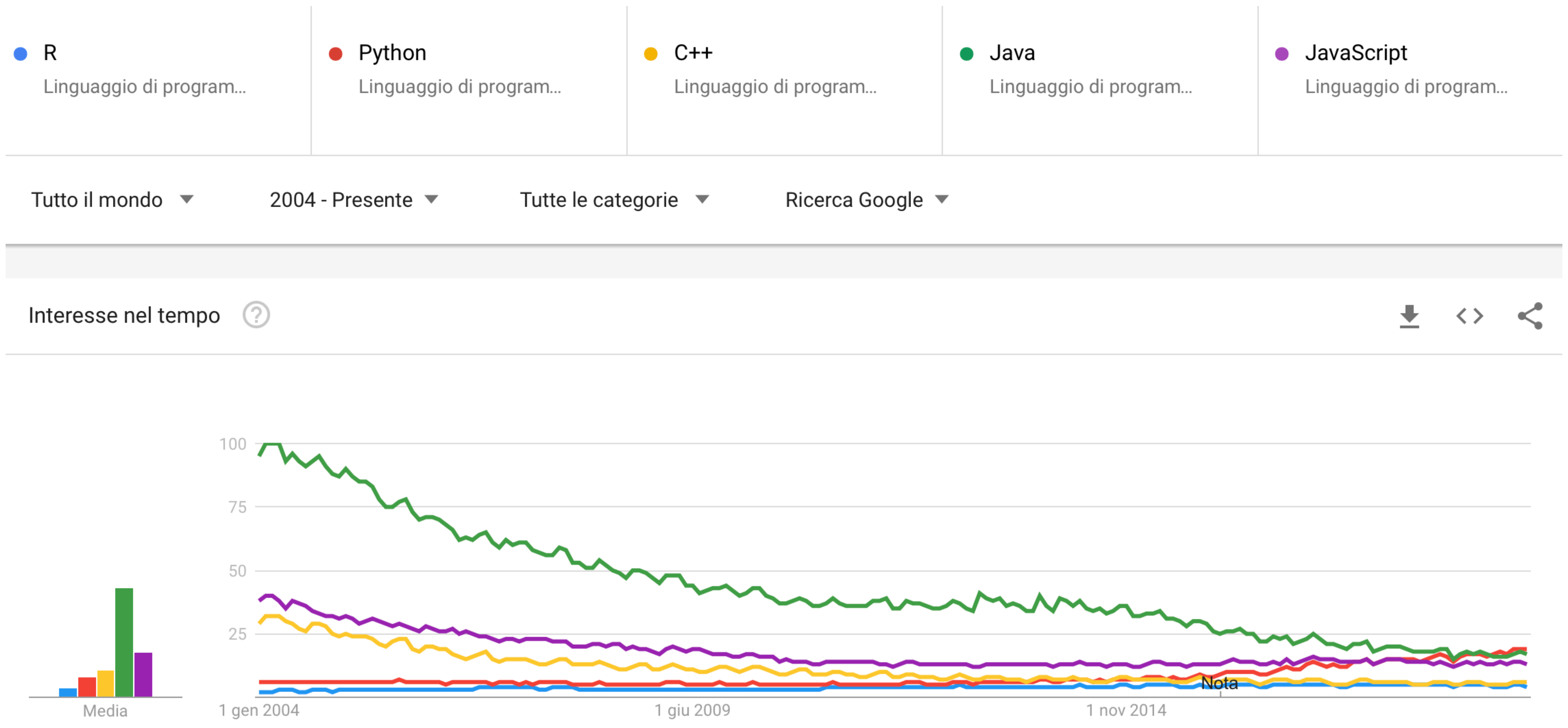
Project Jupyter's name is a reference to the three core programming languages supported by Jupyter: Julia, Python and R

Jupyter Notebook is a web-based interactive computational environment. It supports over 40 programming languages, including Python, R, Julia, C++. It generates notebooks (documents) that can be shared with others and opened through the Jupter Notebook Viewer

Access the Jupyter Project at: https://jupyter.org

2. PYTHON

WHY PYTHON ?

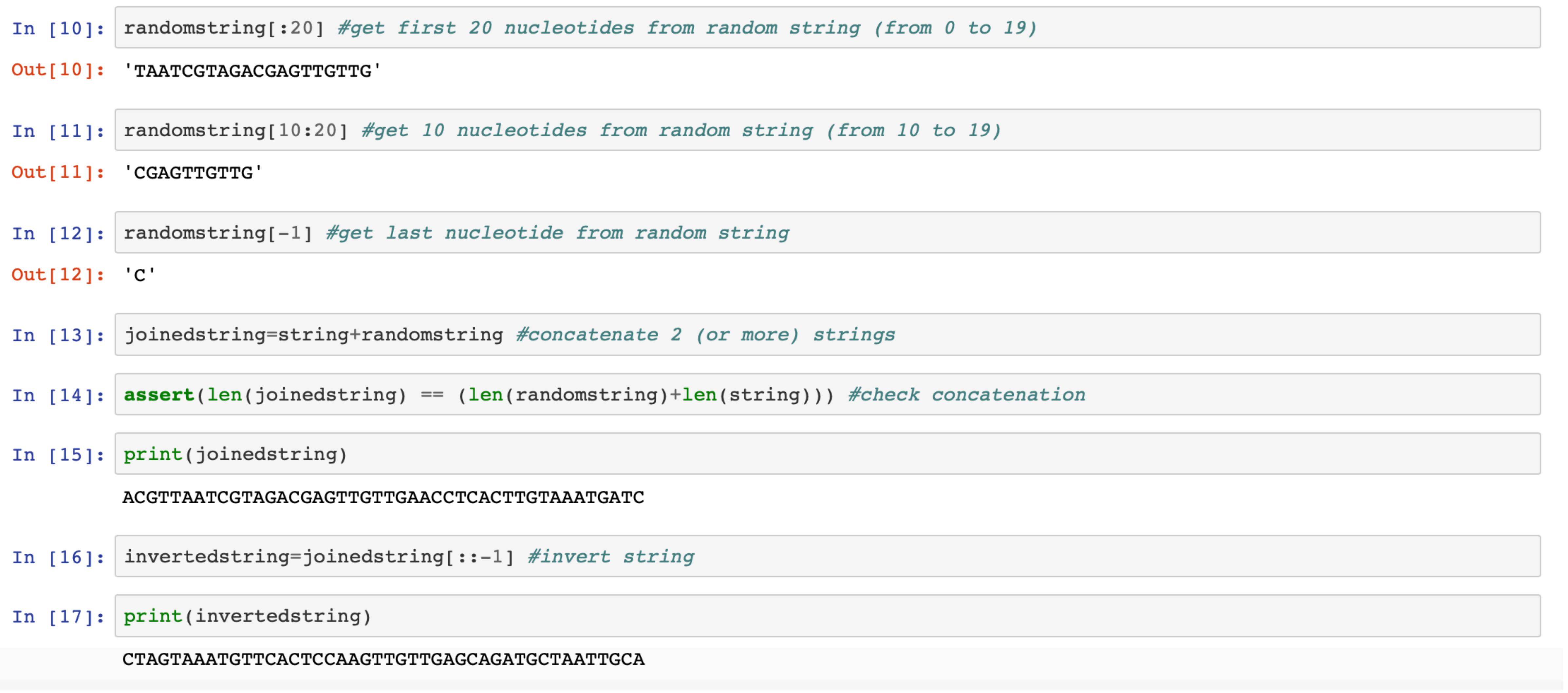


Google trends (2004-to date) for 5 scripting languages (R, Python, C++, Java, Javascript)

STRING BASICS WITH PYTHON (I)

```
In [1]: 'A' #string

Out[1]: 'A'

In [2]: 'ACGT' #longer string

Out[2]: 'ACGT'

In [3]: string='ACGT' #store string into variable

In [4]: print(string) #print string

        ACGT

In [5]: '' #empty string (also called epsilon)

Out[5]: ''

In [6]: import random #load random standard library, useful to get random numbers
        random.choice(string) #choose a random nucleotide from string

Out[6]: 'A'

In [7]: randomstring=''.join([random.choice(string) for _ in range(40)]) #choose a random nucleotide 40 times and join them

In [8]: print(randomstring)

        TAATCGTAGACGAGTTGTTGAACCTCACTTGTAAATGATC

In [9]: len(randomstring) #get length of random string

Out[9]: 40
```

STRING BASICS WITH PYTHON (II)



```
In [10]: randomstring[:20] #get first 20 nucleotides from random string (from 0 to 19)
Out[10]: 'TAATCGTAGACGAGTTGTTG'

In [11]: randomstring[10:20] #get 10 nucleotides from random string (from 10 to 19)
Out[11]: 'CGAGTTGTTG'

In [12]: randomstring[-1] #get last nucleotide from random string
Out[12]: 'C'

In [13]: joinedstring=string+randomstring #concatenate 2 (or more) strings

In [14]: assert(len(joinedstring) == (len(randomstring)+len(string))) #check concatenation

In [15]: print(joinedstring)

         ACGTTAATCGTAGACGAGTTGTTGAACCTCACTTGTAAATGATC

In [16]: invertedstring=joinedstring[::-1] #invert string

In [17]: print(invertedstring)

         CTAGTAAATGTTCACTCCAAGTTGTTGAGCAGATGCTAATTGCA
```

3. DOWNLOAD/READ THE REFERENCE FASTA

DOWNLOAD/READ FASTA

In [1]:
```
!wget --no-check-certificate --no-clobber https://d28rh4a8wq0iu5.cloudfront.net/ads1/data/phix.fa
```

```
--2019-11-19 11:44:58--  https://d28rh4a8wq0iu5.cloudfront.net/ads1/data/phix.fa
Resolving d28rh4a8wq0iu5.cloudfront.net (d28rh4a8wq0iu5.cloudfront.net)... 13.226.23.142, 13.226.23.70, 13.226.23.28,
...
Connecting to d28rh4a8wq0iu5.cloudfront.net (d28rh4a8wq0iu5.cloudfront.net)|13.226.23.142|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5528 (5.4K) [application/octet-stream]
Saving to: ‘phix.fa’

phix.fa             100%[===================>]   5.40K  --.-KB/s    in 0s

2019-11-19 11:44:59 (875 MB/s) - ‘phix.fa’ saved [5528/5528]
```

In [2]:
```
def ReadGenome(filename):

        '''
        Open and read a FASTA, concatenating different sequences into a single one
        '''

        genome=''
        with open(filename, 'r') as f:
                for line in f:
                        if line[0] != '>':
                                genome+=line.rstrip()
        return genome
```

In [3]:
```
genome=ReadGenome('phix.fa')
```

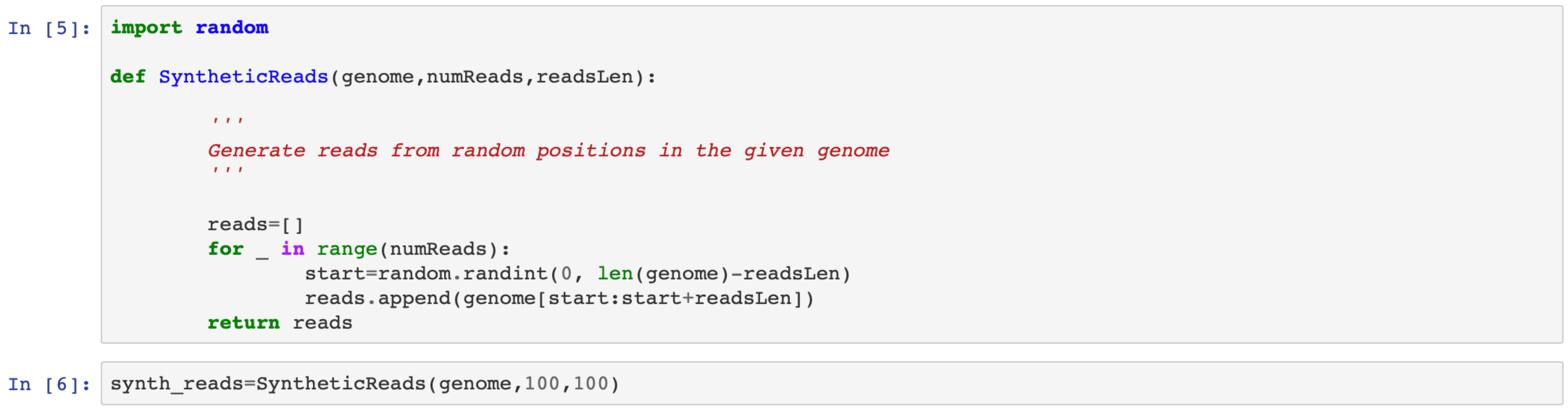
4. BRUTE FORCE EXACT STRING MATCHING

IMPLEMENTING BRUTE FORCE EXACT STRING MATCHING

```
In [4]: def BruteForce(P,T):

                '''
                Simple implementation of Brute Force string matching
                '''

                assert len(P) <= len(T)
                occurrences=[]
                for i in range(len(T)-len(P)+1):
                        match=True
                        for j in range(len(P)):
                                if T[i+j]!=P[j]:
                                        match=False
                                        break
                        if match:
                                occurrences.append(i)
                return occurrences
```

5.MATCH SYNTHETIC READS

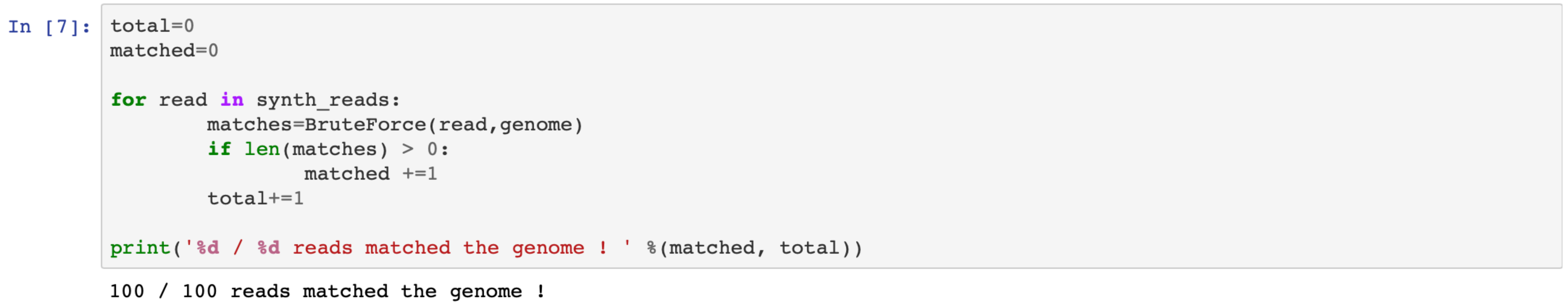
GENERATE SYNTHETIC READS



In [5]:
```
import random

def SyntheticReads(genome,numReads,readsLen):

        '''
        Generate reads from random positions in the given genome
        '''

        reads=[]
        for _ in range(numReads):
                start=random.randint(0, len(genome)-readsLen)
                reads.append(genome[start:start+readsLen])
        return reads
```

In [6]:
```
synth_reads=SyntheticReads(genome,100,100)
```

MATCH SYNTHETIC READS



In [7]:

```
total=0
matched=0

for read in synth_reads:
        matches=BruteForce(read,genome)
        if len(matches) > 0:
                matched +=1
        total+=1

print('%d / %d reads matched the genome ! ' %(matched, total))
```

```
100 / 100 reads matched the genome !
```

6. DOWNLOAD/READ THE FASTQ READS

DOWNLOAD/READ FASTQ

In [8]:
```
!wget --no-check-certificate --no-clobber https://d28rh4a8wq0iu5.cloudfront.net/ads1/data/ERR266411_1.first1000.fastq
```

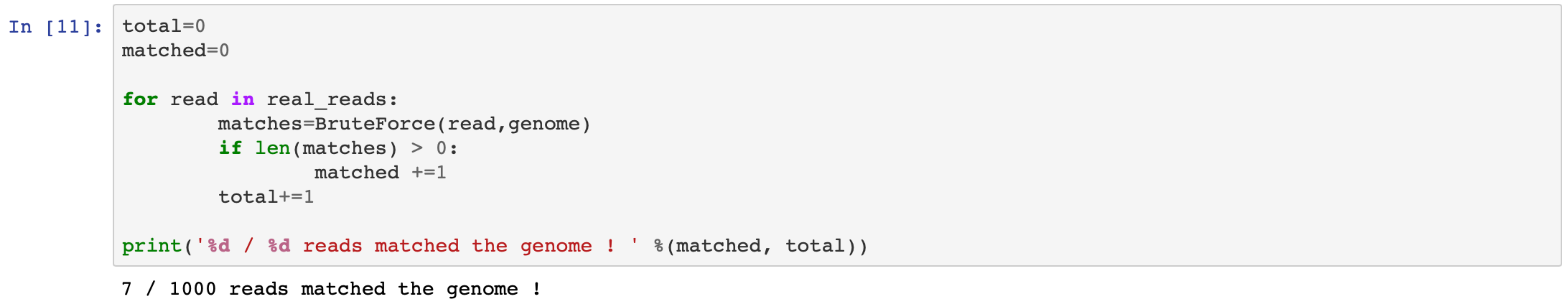
```
--2019-11-19 14:20:40--  https://d28rh4a8wq0iu5.cloudfront.net/ads1/data/ERR266411_1.first1000.fastq
Resolving d28rh4a8wq0iu5.cloudfront.net (d28rh4a8wq0iu5.cloudfront.net)... 13.226.23.70, 13.226.23.142, 13.226.23.55,
...
Connecting to d28rh4a8wq0iu5.cloudfront.net (d28rh4a8wq0iu5.cloudfront.net)|13.226.23.70|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 254384 (248K) [audio/mpeg]
Saving to: 'ERR266411_1.first1000.fastq'

ERR266411_1.first10 100%[===================>] 248.42K  --.-KB/s    in 0.08s

2019-11-19 14:20:41 (2.98 MB/s) - 'ERR266411_1.first1000.fastq' saved [254384/254384]
```

In [9]:
```python
def ReadFastq(filename):

        '''
        Open and read a 4-line FASTQ. Discard first/second/fourth line for each sequence
        '''

        sequences=[]
        with open(filename, 'r') as f:
                while True:
                        f.readline()
                        seq=f.readline().rstrip()
                        if seq=='':
                                break
                        sequences.append(seq)
                        f.readline()
                        f.readline()

        return sequences
```

In [10]:
```python
real_reads=ReadFastq('ERR266411_1.first1000.fastq')
```

7. MATCH REAL READS

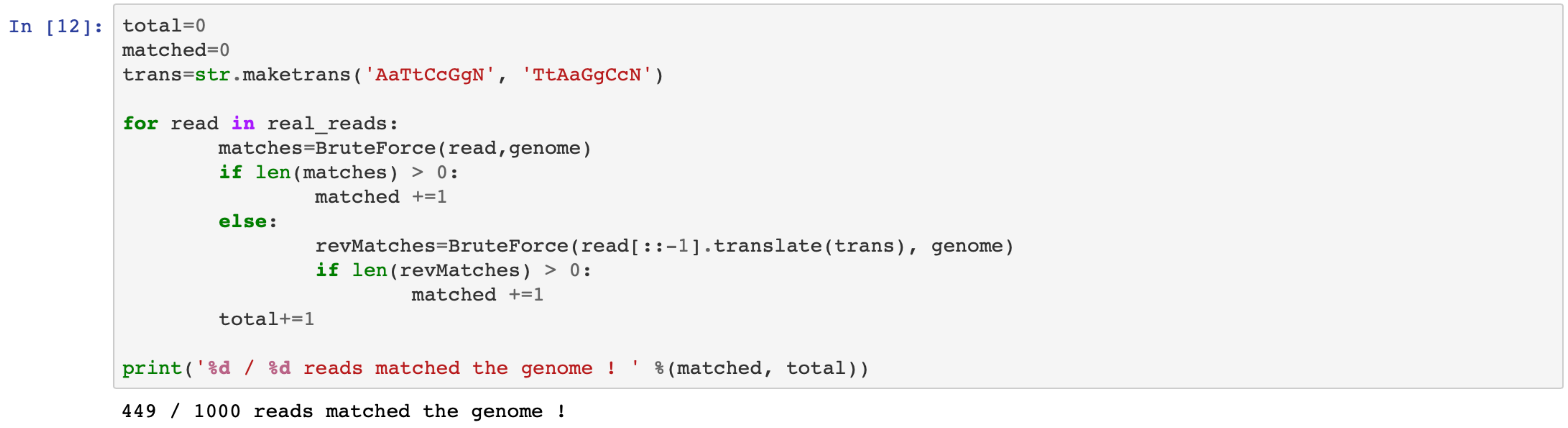
MATCH REAL READS (I)



In [11]:
```
total=0
matched=0

for read in real_reads:
        matches=BruteForce(read,genome)
        if len(matches) > 0:
                matched +=1
        total+=1

print('%d / %d reads matched the genome ! ' %(matched, total))
```

```
7 / 1000 reads matched the genome !
```

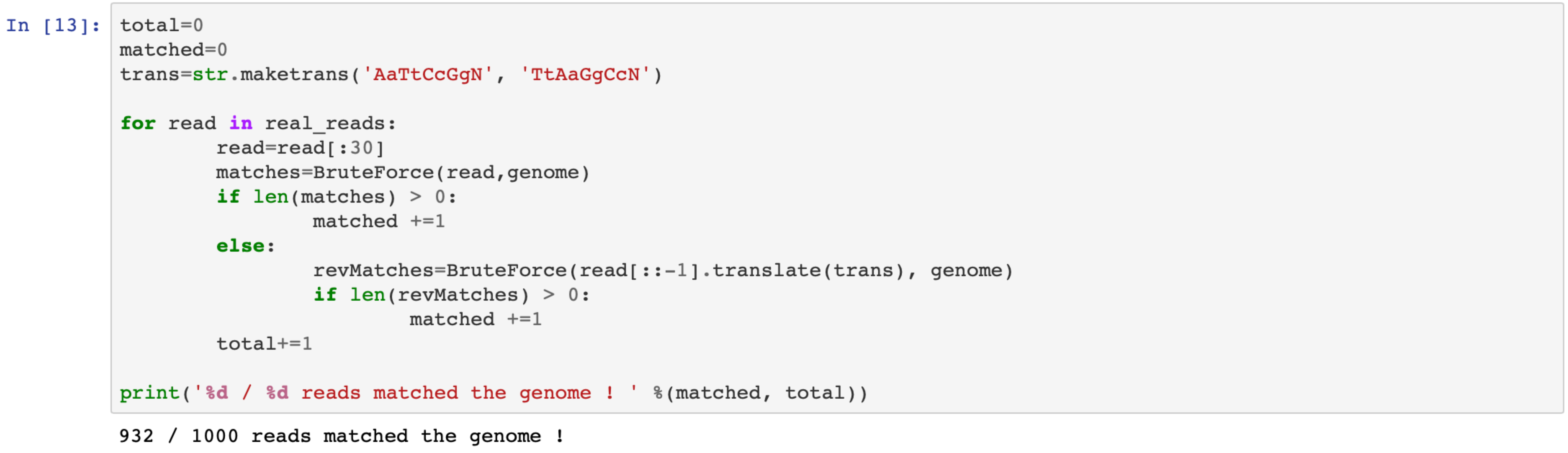
MATCH REAL READS (II)



In [12]:
```
total=0
matched=0
trans=str.maketrans('AaTtCcGgN', 'TtAaGgCcN')

for read in real_reads:
        matches=BruteForce(read,genome)
        if len(matches) > 0:
                matched +=1
        else:
                revMatches=BruteForce(read[::-1].translate(trans), genome)
                if len(revMatches) > 0:
                        matched +=1
        total+=1

print('%d / %d reads matched the genome ! ' %(matched, total))
```

```
449 / 1000 reads matched the genome !
```

MATCH REAL READS (III)



```
In [13]: total=0
         matched=0
         trans=str.maketrans('AaTtCcGgN', 'TtAaGgCcN')

         for read in real_reads:
                 read=read[:30]
                 matches=BruteForce(read,genome)
                 if len(matches) > 0:
                         matched +=1
                 else:
                         revMatches=BruteForce(read[::-1].translate(trans), genome)
                         if len(revMatches) > 0:
                                 matched +=1
                 total+=1

         print('%d / %d reads matched the genome ! ' %(matched, total))
```

```
932 / 1000 reads matched the genome !
```

8. HAMMING/EDIT DISTANCE

IMPLEMENTING HAMMING DISTANCE AND EDIT DISTANCE

In [14]:

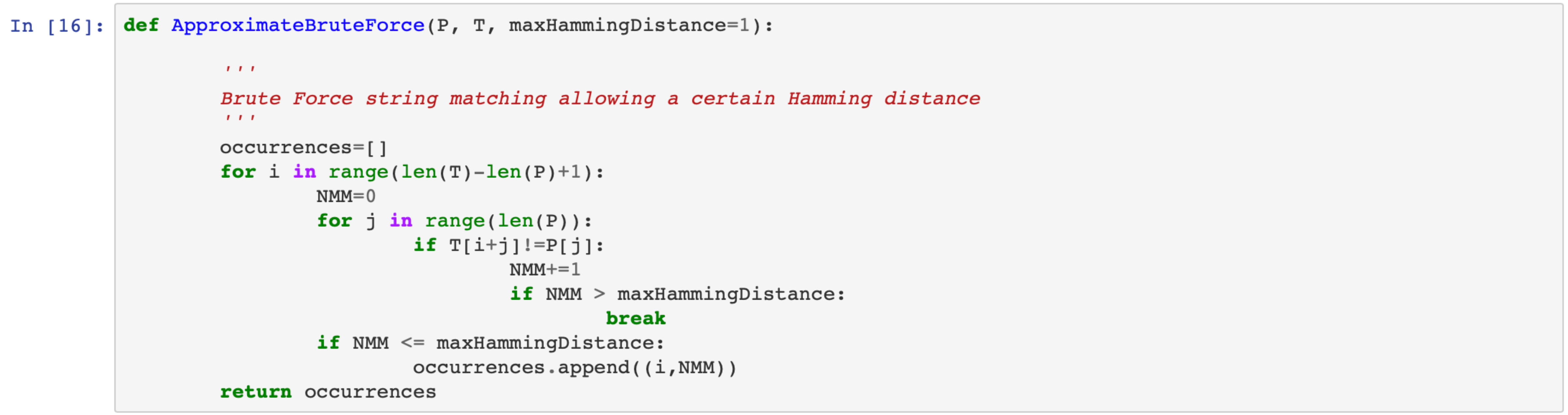
```python
def HammingDistance(P, T):
        '''
        Simple calculation of Hamming distance between 2 strings
        '''

        assert len(P) == len(T)
        NMM = 0
        for i in range(0, len(P)):
                if P[i] != T[i]:
                        NMM += 1
        return NMM
```

In [15]:

```python
def EditDistance(P,T):

        '''
        Calculate edit distance using dynamic programming
        '''

        D = numpy.zeros((len(T)+1, len(P)+1), dtype=int)
        D[0, 1:] = range(1, len(P)+1)
        D[1:, 0] = range(1, len(T)+1)
        for i in range(1, len(T)+1):
                for j in range(1, len(P)+1):
                        delt = 1 if T[i-1] != P[j-1] else 0
                        D[i, j] = min(D[i-1, j-1]+delt, D[i-1, j]+1, D[i, j-1]+1)
        return D[len(T), len(P)]
```

9. BRUTE FORCE APPROXIMATE STRING MATCHING

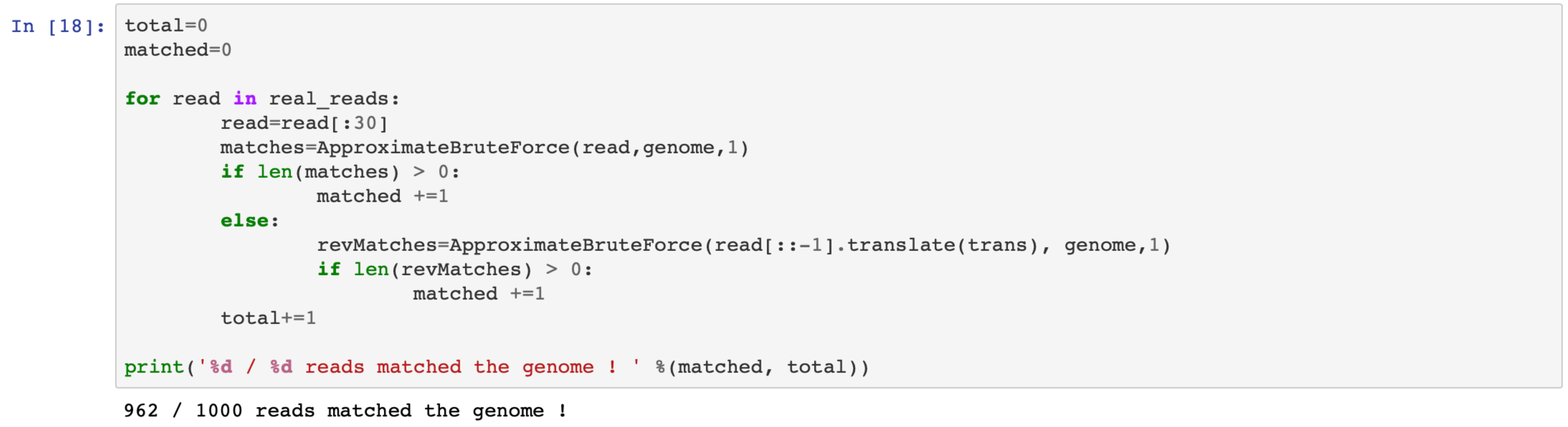
IMPLEMENTING BRUTE FORCE APPROXIMATE STRING MATCHING



In [16]:

```
def ApproximateBruteForce(P, T, maxHammingDistance=1):

        '''
        Brute Force string matching allowing a certain Hamming distance
        '''
        occurrences=[]
        for i in range(len(T)-len(P)+1):
                NMM=0
                for j in range(len(P)):
                        if T[i+j]!=P[j]:
                                NMM+=1
                                if NMM > maxHammingDistance:
                                        break
                if NMM <= maxHammingDistance:
                        occurrences.append((i,NMM))
        return occurrences
```

MATCH REAL READS

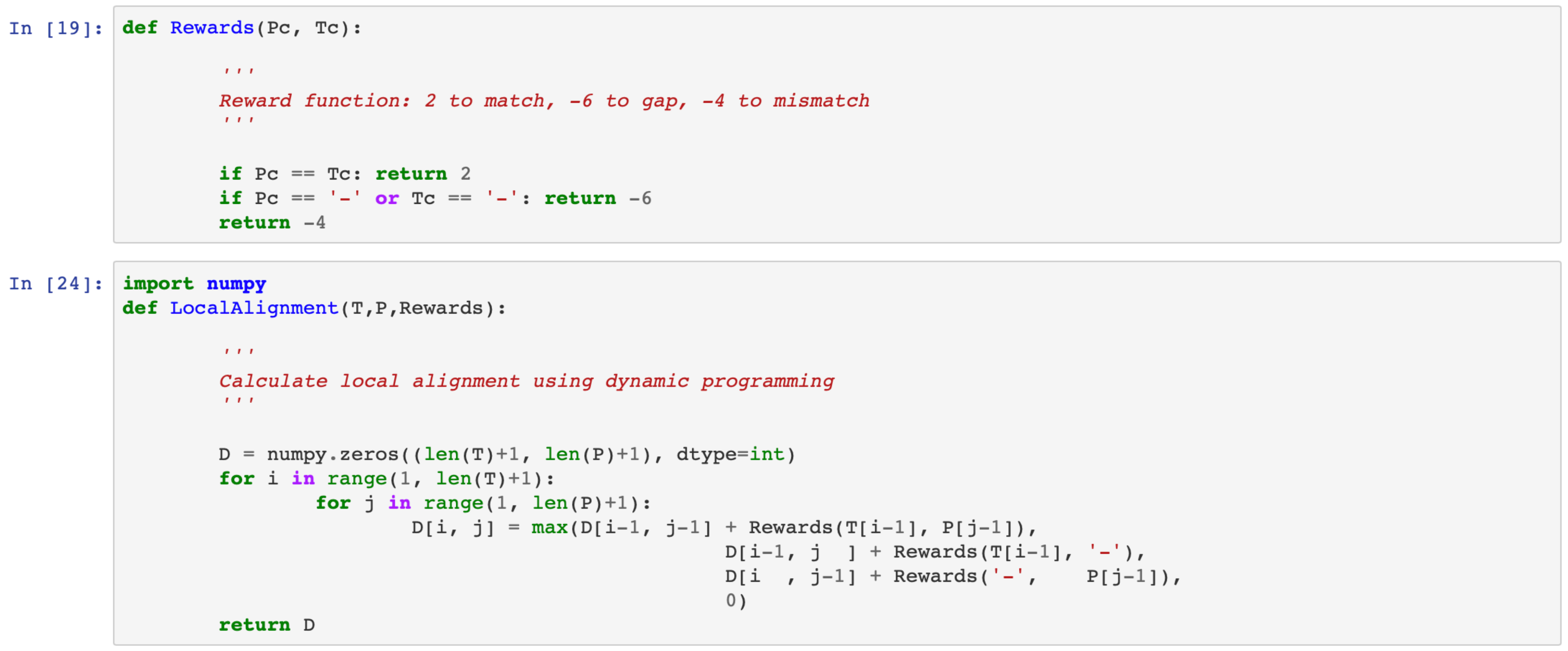


```
In [18]: total=0
         matched=0

         for read in real_reads:
                 read=read[:30]
                 matches=ApproximateBruteForce(read,genome,1)
                 if len(matches) > 0:
                         matched +=1
                 else:
                         revMatches=ApproximateBruteForce(read[::-1].translate(trans), genome,1)
                         if len(revMatches) > 0:
                                 matched +=1
                 total+=1

         print('%d / %d reads matched the genome ! ' %(matched, total))
```

```
962 / 1000 reads matched the genome !
```

10. LOCAL ALIGNMENT

IMPLEMENTING LOCAL ALIGNMENT



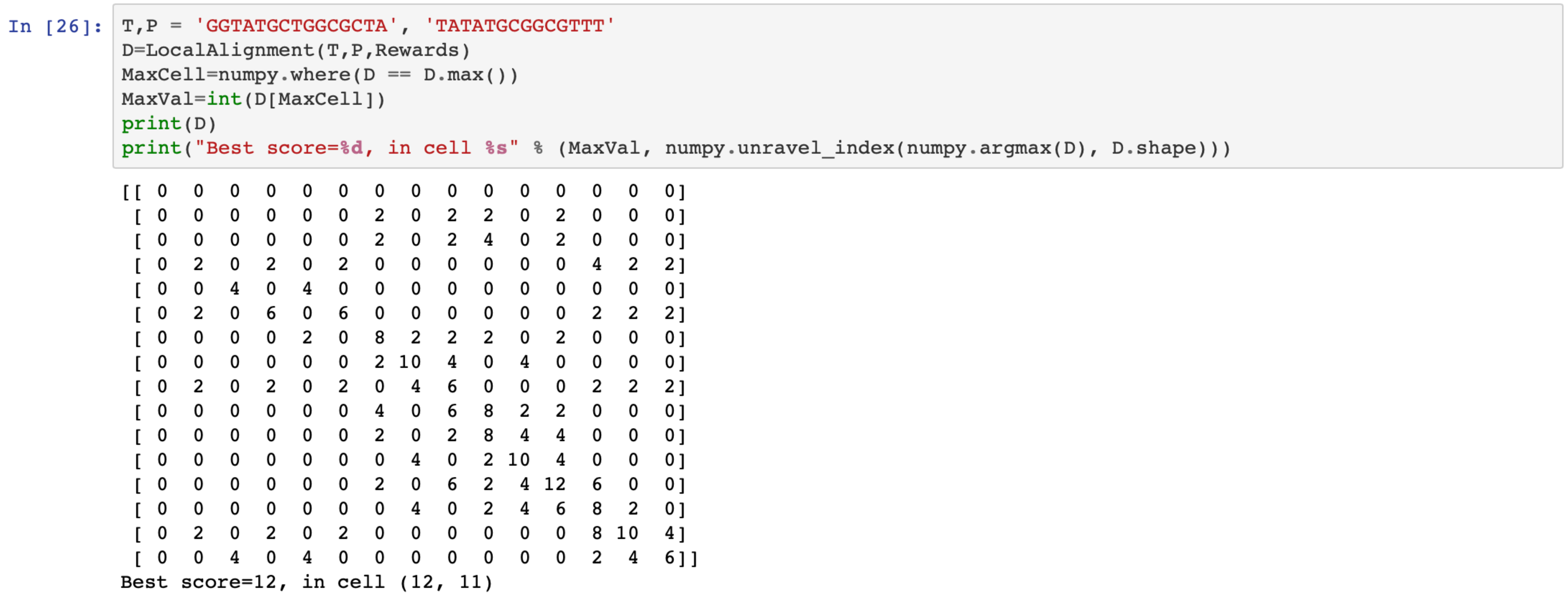
In [19]:

```python
def Rewards(Pc, Tc):
        '''
        Reward function: 2 to match, -6 to gap, -4 to mismatch
        '''

        if Pc == Tc: return 2
        if Pc == '-' or Tc == '-': return -6
        return -4
```

In [24]:

```python
import numpy
def LocalAlignment(T,P,Rewards):
        '''
        Calculate local alignment using dynamic programming
        '''

        D = numpy.zeros((len(T)+1, len(P)+1), dtype=int)
        for i in range(1, len(T)+1):
                for j in range(1, len(P)+1):
                        D[i, j] = max(D[i-1, j-1] + Rewards(T[i-1], P[j-1]),
                                                          D[i-1, j  ] + Rewards(T[i-1], '-'),
                                                          D[i  , j-1] + Rewards('-',    P[j-1]),
                                                          0)
        return D
```

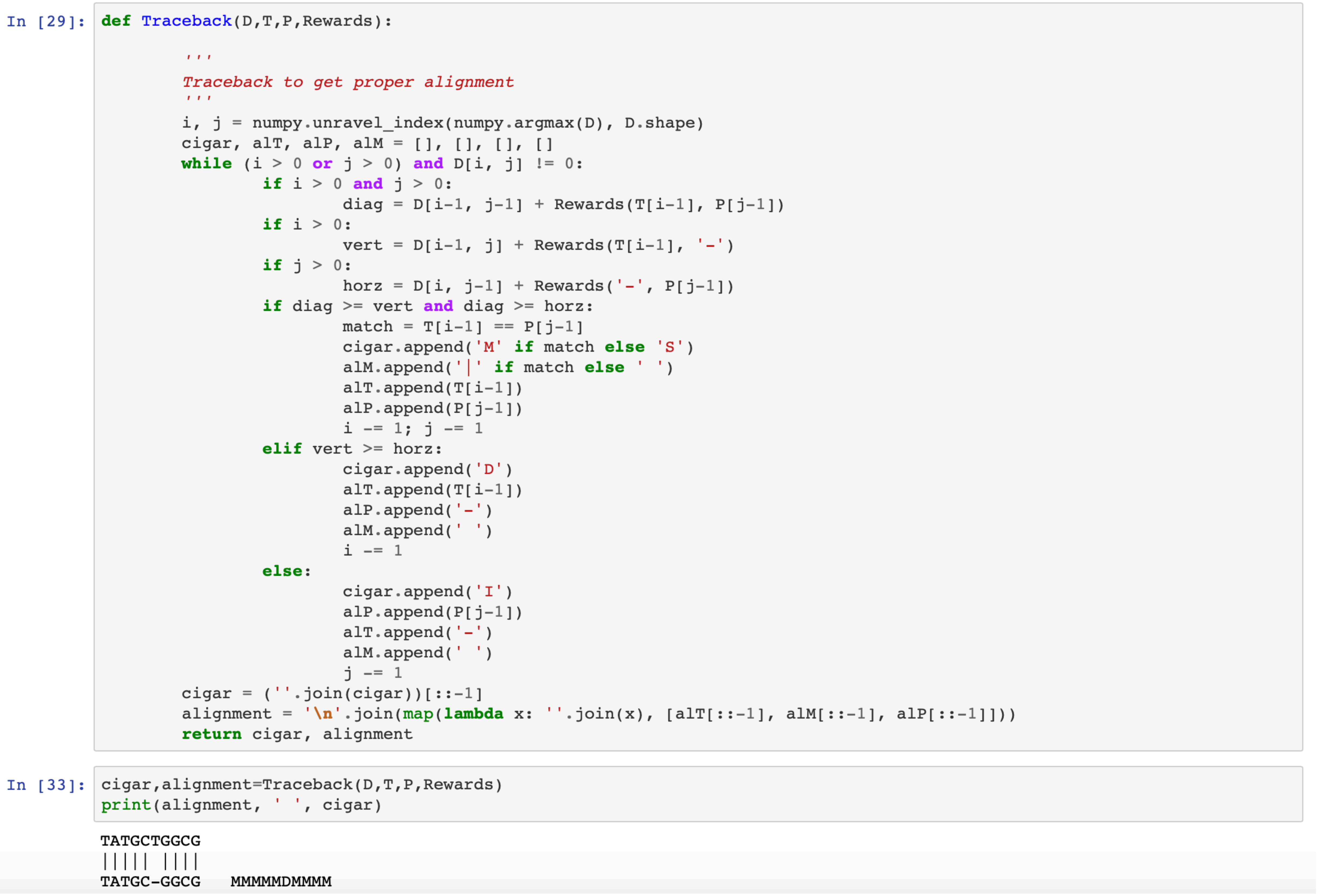
THE SCORING MATRIX



```
In [26]: T,P = 'GGTATGCTGGCGCTA', 'TATATGCGGCGTTT'
         D=LocalAlignment(T,P,Rewards)
         MaxCell=numpy.where(D == D.max())
         MaxVal=int(D[MaxCell])
         print(D)
         print("Best score=%d, in cell %s" % (MaxVal, numpy.unravel_index(numpy.argmax(D), D.shape)))
```

```
[[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  2  0  2  2  0  2  0  0  0]
 [ 0  0  0  0  0  0  2  0  2  4  0  2  0  0  0]
 [ 0  2  0  2  0  2  0  0  0  0  0  0  4  2  2]
 [ 0  0  4  0  4  0  0  0  0  0  0  0  0  0  0]
 [ 0  2  0  6  0  6  0  0  0  0  0  0  2  2  2]
 [ 0  0  0  0  2  0  8  2  2  2  0  2  0  0  0]
 [ 0  0  0  0  0  0  2 10  4  0  4  0  0  0  0]
 [ 0  2  0  2  0  2  0  4  6  0  0  0  2  2  2]
 [ 0  0  0  0  0  0  4  0  6  8  2  2  0  0  0]
 [ 0  0  0  0  0  0  2  0  2  8  4  4  0  0  0]
 [ 0  0  0  0  0  0  0  4  0  2 10  4  0  0  0]
 [ 0  0  0  0  0  0  2  0  6  2  4 12  6  0  0]
 [ 0  0  0  0  0  0  0  4  0  2  4  6  8  2  0]
 [ 0  2  0  2  0  2  0  0  0  0  0  0  8 10  4]
 [ 0  0  4  0  4  0  0  0  0  0  0  0  2  4  6]]
Best score=12, in cell (12, 11)
```

TRACING BACK



```
In [29]: def Traceback(D,T,P,Rewards):
                 '''
                 Traceback to get proper alignment
                 '''
                 i, j = numpy.unravel_index(numpy.argmax(D), D.shape)
                 cigar, alT, alP, alM = [], [], [], []
                 while (i > 0 or j > 0) and D[i, j] != 0:
                         if i > 0 and j > 0:
                                 diag = D[i-1, j-1] + Rewards(T[i-1], P[j-1])
                         if i > 0:
                                 vert = D[i-1, j] + Rewards(T[i-1], '-')
                         if j > 0:
                                 horz = D[i, j-1] + Rewards('-', P[j-1])
                         if diag >= vert and diag >= horz:
                                 match = T[i-1] == P[j-1]
                                 cigar.append('M' if match else 'S')
                                 alM.append('|' if match else ' ')
                                 alT.append(T[i-1])
                                 alP.append(P[j-1])
                                 i -= 1; j -= 1
                         elif vert >= horz:
                                 cigar.append('D')
                                 alT.append(T[i-1])
                                 alP.append('-')
                                 alM.append(' ')
                                 i -= 1
                         else:
                                 cigar.append('I')
                                 alP.append(P[j-1])
                                 alT.append('-')
                                 alM.append(' ')
                                 j -= 1
                 cigar = (''.join(cigar))[::-1]
                 alignment = '\n'.join(map(lambda x: ''.join(x), [alT[::-1], alM[::-1], alP[::-1]]))
                 return cigar, alignment
```

```
In [33]: cigar,alignment=Traceback(D,T,P,Rewards)
         print(alignment, ' ', cigar)
```

```
TATGCTGGCG
||||| ||||
TATGC-GGCG   MMMMMDMMMM
```

1. Email me @:
   davide.bolognini@unifi.it
2. Get slides (.key + .pdf) from GitHub:
   https://github.com/davidebolo1993/Classes

THAT'ALL, FOLKS !