Spelman College
A Choice to Change the World

Dissecting the Tools of Bioinformatics: Building SpelFinder, a primer identification tool in Python

Charles Hardnett and Cynthia Bauerle
Departments of Computer Science and Biology, Spelman College, Atlanta, GA 30314

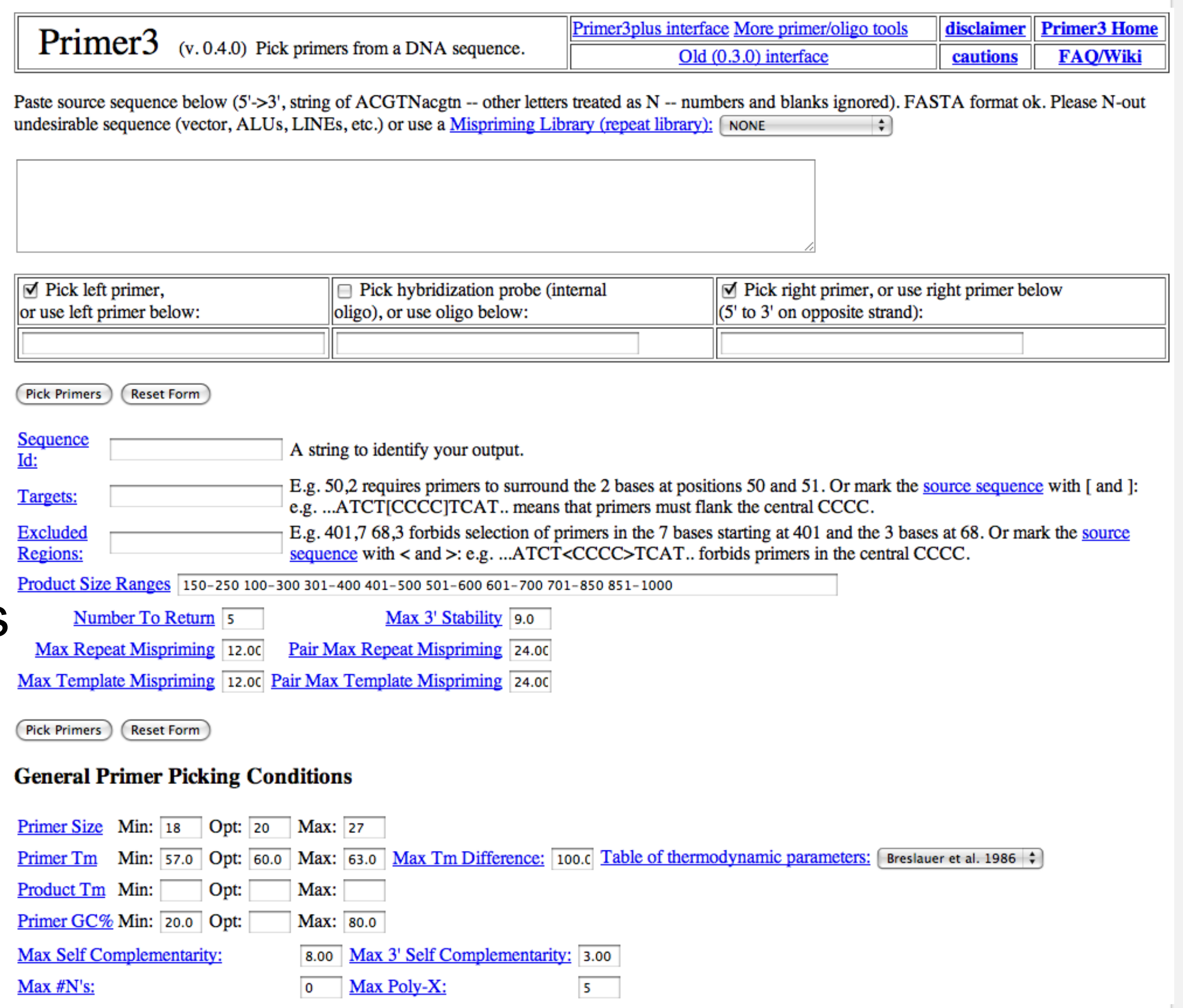
science education alliance
HOWARD HUGHES MEDICAL INSTITUTE

CONTEXT

Finishing the Genome: Students used Sequencher to identify low confidence/low coverage regions within the returned genomic sequence. They used Primer3 to select sites for the design of oligonucleotide primers to direct DNA sequencing across targeted regions.

Primer3: This tool contains a primer identification program readily available over the Internet, that identifies suitable primers. Users input a source sequence, set search parameters, and obtain a list of potential primers as output. Primer3 parameter variables include:

- Sequence length
- Melting T(T_m)
- 3' stability
- GC%
- Poly-X runs
- self-complementarity
- Excludable regions



Primer3 (v. 0.4.0) Pick primers from a DNA sequence. | Primer3plus interface More primer/oligo tools | disclaimer | Primer3 Home | Old (0.3.0) interface | cautions | FAQ/Wiki

Paste source sequence below (5'->3', string of ACGTNacgtn -- other letters treated as N -- numbers and blanks ignored). FASTA format ok. Please N-out undesirable sequence (vector, ALUs, LINEs, etc.) or use a Mispriming Library (repeat library): NONE

☑ Pick left primer, or use left primer below: | ☐ Pick hybridization probe (internal oligo), or use oligo below: | ☑ Pick right primer, or use right primer below (5' to 3' on opposite strand):

Sequence Id: A string to identify your output.
Targets: E.g. 50,2 requires primers to surround the 2 bases at positions 50 and 51. Or mark the source sequence with [and]: e.g. ...ATCT[CCCC]TCAT.. means that primers must flank the central CCCC.
Excluded Regions: E.g. 401,7 68,3 forbids selection of primers in the 7 bases starting at 401 and the 3 bases at 68. Or mark the source sequence with < and >: e.g. ...ATCT<CCCC>TCAT.. forbids primers in the central CCCC.
Product Size Ranges: 150-250 100-300 301-400 401-500 501-600 601-700 701-850 851-1000
Number To Return: 5 | Max 3' Stability: 9.0
Max Repeat Mispriming: 12.00 | Pair Max Repeat Mispriming: 24.00
Max Template Mispriming: 12.00 | Pair Max Template Mispriming: 24.00

General Primer Picking Conditions

Primer Size Min: 18 Opt: 20 Max: 27
Primer Tm Min: 57.0 Opt: 60.0 Max: 63.0 Max Tm Difference: 100.0 Table of thermodynamic parameters: Breslauer et al. 1986
Product Tm Min: Opt: Max:
Primer GC% Min: 20.0 Opt: Max: 80.0
Max Self Complementarity: 8.00 Max 3' Self Complementarity: 3.00
Max #N's: 0 Max Poly-X: 5

Understanding Primer3: To better understand how Primer3 operates to identify acceptable primer sequences, students built a simple sequence identification program in Python. Spelfinder searches a source sequence and identifies primers based on a simplified set of parameters:

- Sequence length: 18 ≥ 30 nucleotides
- GC%: 40 ≥ 60
- Poly-X ≤ 3 (where X = A, T, G, or C)
- T_m: 55°C ≤ T_m ≤ 75°C
  where $T_m = [(G+C)_{primer} \times 4°C] + [(A+T)_{primer} \times 2°C]$

GENERAL PROGRAMMING STRATEGY

Creating a program is an *iterative* process:

1. Define a search criterion
2. Design the algorithm using *pseudocode*
3. Implement the *algorithm* as a computer program
4. Test the computer program
5. Fix errors by repeating steps 1-3 until all tests pass

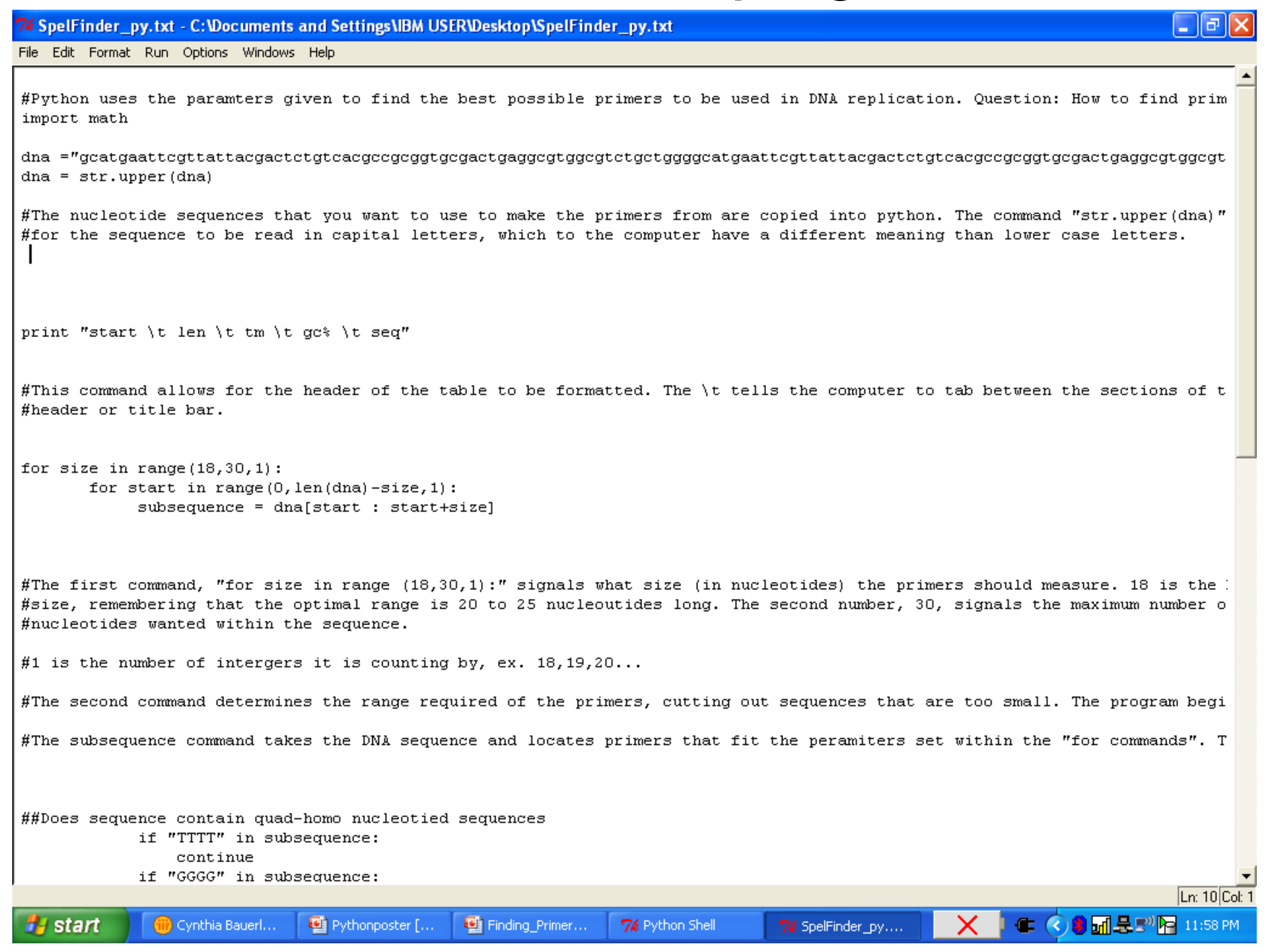
Program elements:
String: Nucleotide source sequence is stored as a collection of characters called a *string*
Algorithm: Each search parameter is represented by a computational procedure called an *algorithm* that sets criteria for the search
Program: All algorithms defining search parameters are assembled into a *script* that directs the function of the program

OBJECTIVE
To help students better understand the construction and utility of standard sequence analysis tools

ABSTRACT

As participant in the National Genomics Research Initiative supported by HHMI, the Biology department at Spelman College offered a course-based research opportunity for first year science students to isolate and characterize novel mycobacteriophages. In fall 2008, twenty first year students isolated novel mycophages and conducted standard microscopic and molecular characterization. In spring 2009, fourteen of these students, joined by six advanced biology majors, conducted genomic annotation of one phage isolate using standard sequence analysis approaches and bioinformatic tools available via the Internet. While prior experience *applying* bioinformatics tools varied among the cohort, no students described any prior experience in computer programming. To help students better understand the construction and utility of standard molecular analysis tools, we developed a course activity in which students used the Python programming language to build a primer identification tool that they named SpelFinder. Students were introduced to basic elements of the Python language in a hands-on group exercise, and then used these elements to build simple algorithms. Ultimately, students assembled algorithms into a script that analyzes an input nucleotide sequence and reports potential primer target sites as output. The program was based on an algorithm that reflected four relevant biological criteria. Writing the program *de novo* in Python introduced students to basic programming strategies for developing functional algorithms useful in standard bioinformatics tools. Specifically, students were challenged to understand how biological criteria may be translated into a set of rules that drives the algorithm of a sequence analysis software program.

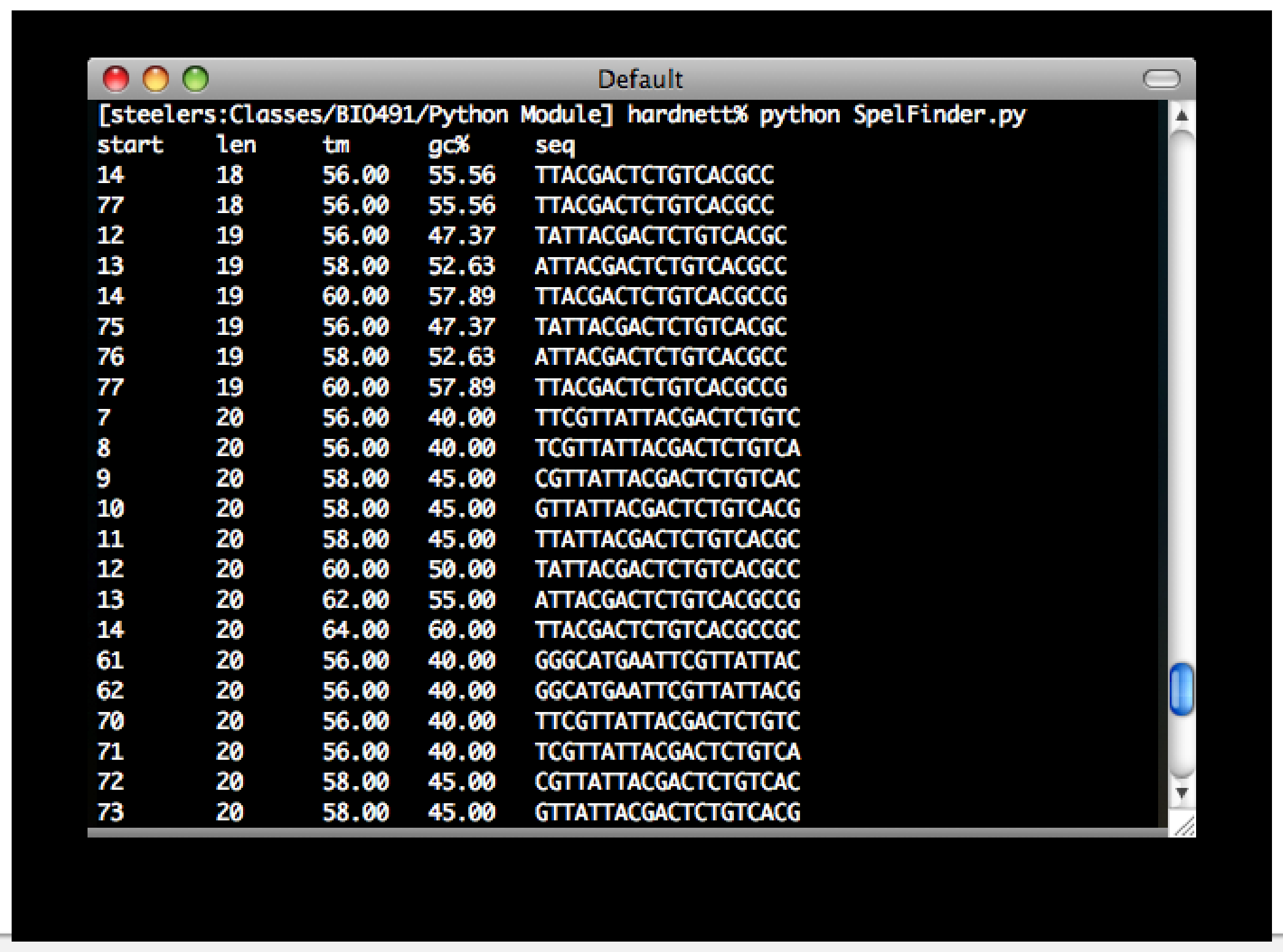
SPELFINDER: What the program looks like



SpelFinder_py.txt - C:\Documents and Settings\IBM USER\Desktop\SpelFinder_py.txt

```
#Python uses the paramters given to find the best possible primers to be used in DNA replication. Question: How to find prim
import math

dna ="gcatgaattcgttattacgactctgtcacgccgcggtgcgactgaggcgtggcgtctgctggggcatgaattcgttattacgactctgtcacgccgcggtgcgactgaggcgtggcgt
dna = str.upper(dna)

#The nucleotide sequences that you want to use to make the primers from are copied into python. The command "str.upper(dna)"
#for the sequence to be read in capital letters, which to the computer have a different meaning than lower case letters.



print "start \t len \t tm \t gc% \t seq"


#This command allows for the header of the table to be formatted. The \t tells the computer to tab between the sections of t
#header or title bar.


for size in range(18,30,1):
       for start in range(0,len(dna)-size,1):
            subsequence = dna[start : start+size]



#The first command, "for size in range (18,30,1):" signals what size (in nucleotides) the primers should measure. 18 is the 
#size, remembering that the optimal range is 20 to 25 nucleoutides long. The second number, 30, signals the maximum number o
#nucleotides wanted within the sequence.

#1 is the number of intergers it is counting by, ex. 18,19,20...

#The second command determines the range required of the primers, cutting out sequences that are too small. The program begi

#The subsequence command takes the DNA sequence and locates primers that fit the peramiters set within the "for commands". T



##Does sequence contain quad-homo nucleotied sequences
            if "TTTT" in subsequence:
                continue
            if "GGGG" in subsequence:
```

SPELFINDER: What the output looks like:

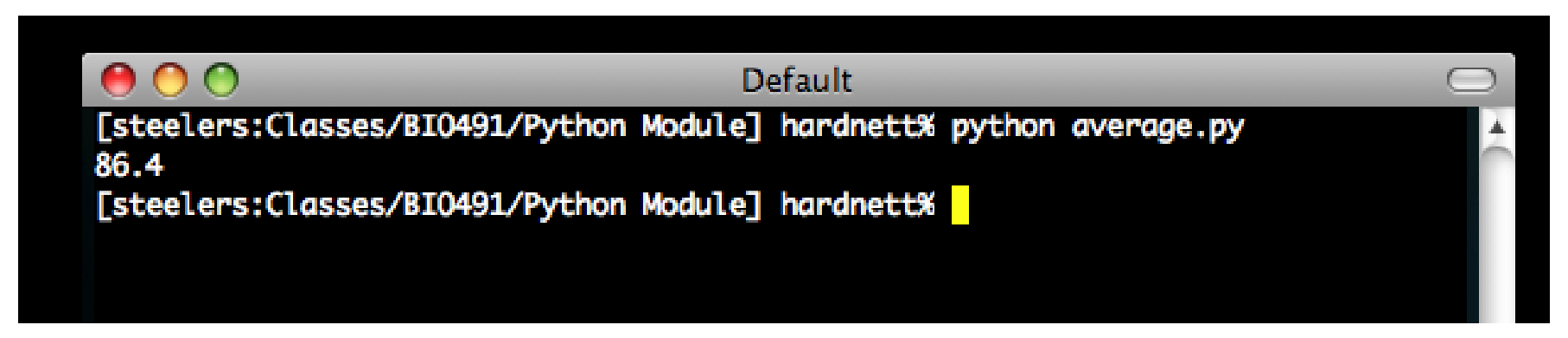


```
[steelers:Classes/BIO491/Python Module] hardnett% python SpelFinder.py
start    len     tm      gc%     seq
14       18      56.00   55.56   TTACGACTCTGTCACGCC
77       18      56.00   55.56   TTACGACTCTGTCACGCC
12       19      56.00   47.37   TATTACGACTCTGTCACGC
13       19      58.00   52.63   ATTACGACTCTGTCACGCC
14       19      60.00   57.89   TTACGACTCTGTCACGCCG
75       19      56.00   47.37   TATTACGACTCTGTCACGC
76       19      58.00   52.63   ATTACGACTCTGTCACGCC
77       19      60.00   57.89   TTACGACTCTGTCACGCCG
7        20      56.00   40.00   TTCGTTATTACGACTCTGTC
8        20      56.00   40.00   TCGTTATTACGACTCTGTCA
9        20      58.00   45.00   CGTTATTACGACTCTGTCAC
10       20      58.00   45.00   GTTATTACGACTCTGTCACG
11       20      58.00   45.00   TTATTACGACTCTGTCACGC
12       20      60.00   50.00   TATTACGACTCTGTCACGCC
13       20      62.00   55.00   ATTACGACTCTGTCACGCCG
14       20      64.00   60.00   TTACGACTCTGTCACGCCGC
61       20      56.00   40.00   GGGCATGAATTCGTTATTAC
62       20      56.00   40.00   GGCATGAATTCGTTATTACG
70       20      56.00   40.00   TTCGTTATTACGACTCTGTC
71       20      56.00   40.00   TCGTTATTACGACTCTGTCA
72       20      58.00   45.00   CGTTATTACGACTCTGTCAC
73       20      58.00   45.00   GTTATTACGACTCTGTCACG
```

PYTHON BASICS

Python is a general purpose, highly readable, high level computer programming language.

Python program may be created in a standard text editor such as Notepad or TextEdit. Once saved, the program may be run by using the python command to execute the program. The output will appear on the next line:



```
[steelers:Classes/BIO491/Python Module] hardnett% python average.py
86.4
[steelers:Classes/BIO491/Python Module] hardnett%
```

Python performs calculations using common arithmetic symbols:
Addition: +
Subtraction: -
Multiplication: *
Division: / and / /
Equal: =
Exponentiation: * *
Modulus: % (finds remainder, e.g. 10 % 3 = 1)

TASK 1: Compute all subsequences of lengths 18 ≥ 30 from the source sequence string

Pseudocode:

```
Start = 1
While (Start < length of DNA Sequence)
    For each extracted subsequence starting at Start with length
        18, 19, 20, 21,...30
      Analyze the subsequence using criteria
         1. Disallowed consecutive nucleotides
         2. GC Content range
         3. Melting Temperature Range
    End For
    increment Start (create next start position)
End While
```

Python:

```
for size in range(18,30,1):
    for start in range(0,len(dna)-size,1):
        subsequence = dna[start : start+size]
```

Output:

- The first statement assigns values 18, 19, 20, ... 30 to size on at a time
- The second statement assigns starting positions 0, 1, 2, 3, ..., end
  - Python strings start at position 0 and not 1!
- The third extracts the subsequence from the start to its position at a length of size (start+size is the end position for the subsequence)

TASK 2: Skipping subsequences containing poly-X_n where X is A, T, G, or C, and n ≥ 4

Pseudocode:

```
if (subsequence contains "TTTT") or
(subsequence contains "AAAA") or
(subsequence contains "CCCC") or
(subsequence contains "GGGG") then
skip this subsequence
```

Python:

```
 if "TTTT" in subsequence:
       continue
if "GGGG" in subsequence:
    continue
if "AAAA" in subsequence:
    continue
if "CCCC" in subsequence:
    continue
```

Output:

- The **continue** means to skip the other criteria in the loop and go to the next loop iteration i.e. next subsequence
- The **in** operator is a Python string operator for determining the existence of a substring within a string