CS 5 Green

Learning Goals

- Finding extremes
- Distinguish between for loops and while loops
- Implement while loops
- Provide useful background on homework

Finding extremz!

dictionary = [
"abdomen",
"abdominal",
"abduct",
"abduction,"
"aberration,"
"abet,"
"abhor,"
"abhorrence,"
"abhorrent,"
"abide,"
"abiding,"
"ability,"
"abject,"
"ablaze,"
…
…
etc.
…
…]

```
def z(input):
    '''Count z's in a string'''
    counter = 0
    for symbol in input:
        if symbol == 'z':
            counter = counter +1
    return counter
```

Notes


```
def extremz(word_list):
    '''Find and return the word with the most z's'''
```

while loops

```
def mystery(n):
    while n > 1:
        if n % 2 != 0:
            return False
        n = n / 2
    return True
```

Repeat until this expression is no longer True

```
>>> mystery(1)

>>> mystery(6)

>>> mystery(16)
```

while vs. for loop

```
def mystery(n):
    while n > 1:
        if n % 2 != 0:
            return False
        n = n / 2
    return True
```

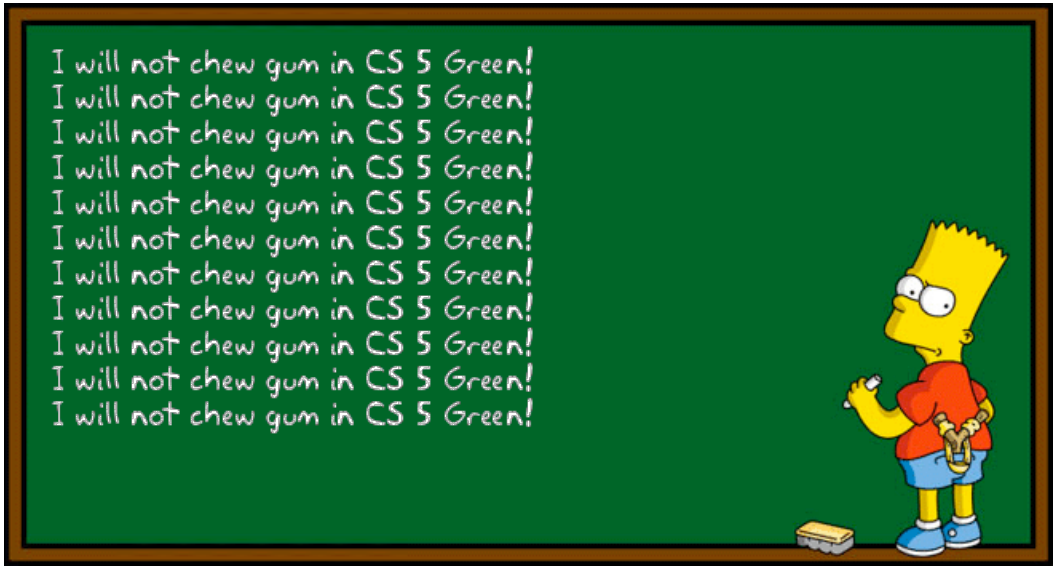
```
def mystery2(n):
    for k in range(1, n):
        if n == 1:
            return True
        elif not n % 2 == 0:
            return False
        n = n / 2
```

while loops

```
def mystery(n):
    start = 0
    while start <= n:
        print(start)
        start = start + 1
```

while loops

```
def bart():
    while True:
        print("I will not chew gum in CS5 Green.")
```

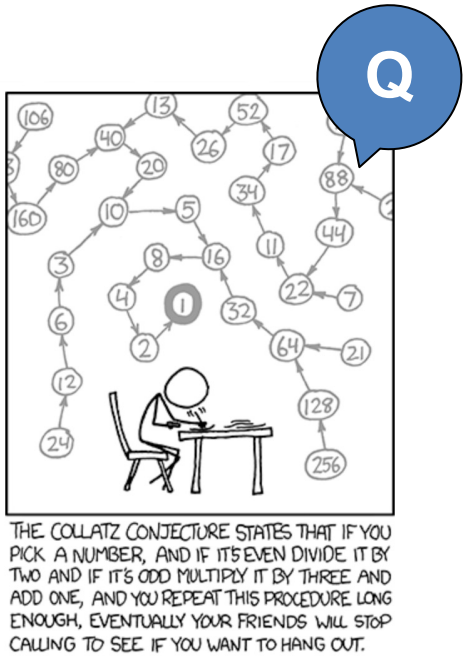



```
>>> bart(4)
1 I will not chew gum in CS5 Green.
2 I will not chew gum in CS5 Green.
3 I will not chew gum in CS5 Green.
4 I will not chew gum in CS5 Green.
```

```
def bart(times):
    # use for loops

        print(line, "I will not chew gum in CS5 Green.")
```

```
def bart(times):
    # use while loops
    line = 1
    while line <= times:
        print(line, "I will not chew gum in CS5 Green.")
        line = line + 1
```

while loops




```
def collatz(n):
    '''Applies the collatz function to n.'''
    if n % 2 == 0:
        return n/2
    else:
        return 3*n +1

def how_many_times(n):
    '''Determines the number of times collatz must be
    applied to n before we get 1.'''
    counter = 0
    
    return counter
```

```
>>> 5 == 5
True
>>> 5 != 6
True
```

```
>>> how_many_times(5)
5
```

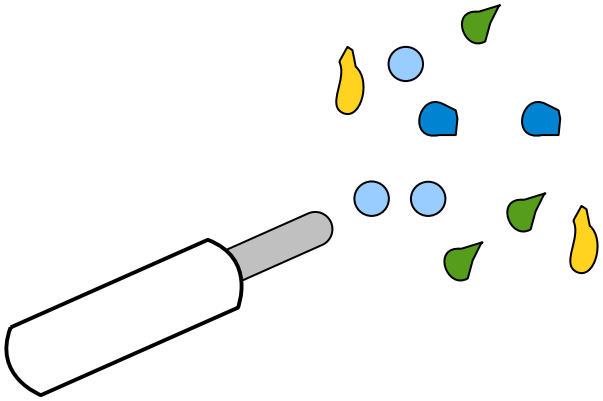
5 -> 16 -> 8 -> 4 -> 2 -> 1

A Prime Example of Looping!

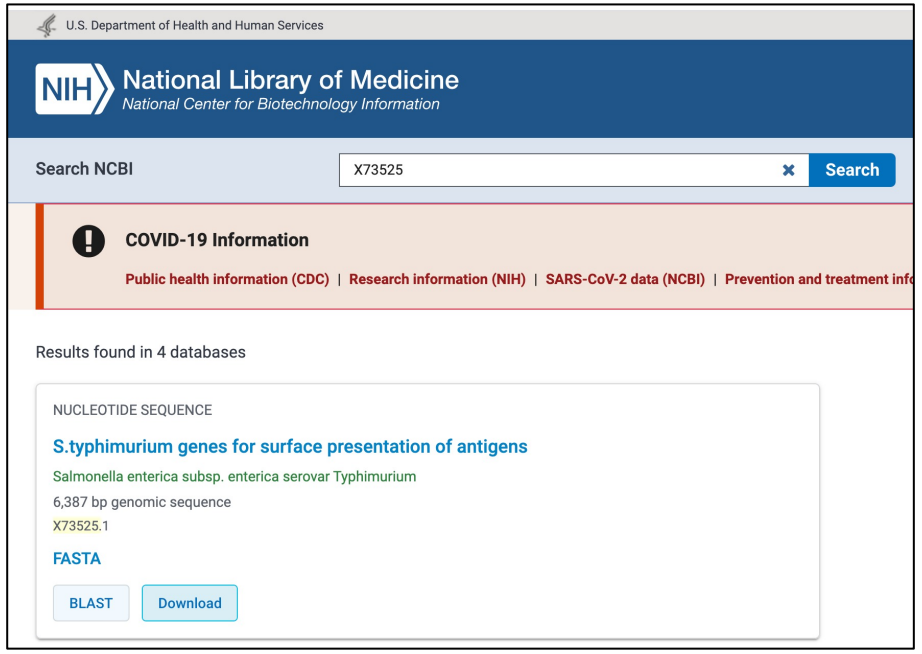
Worksheet Q

```
>>> this_many_primes(5)
[2, 3, 5, 7, 11]

def prime(k):
    '''Returns True if k is prime and False otherwise'''
    for d in range(2, k):
        if k % d == 0:
            return False
    return True

def this_many_primes(n):
    '''Returns a list of the first n primes.'''
```

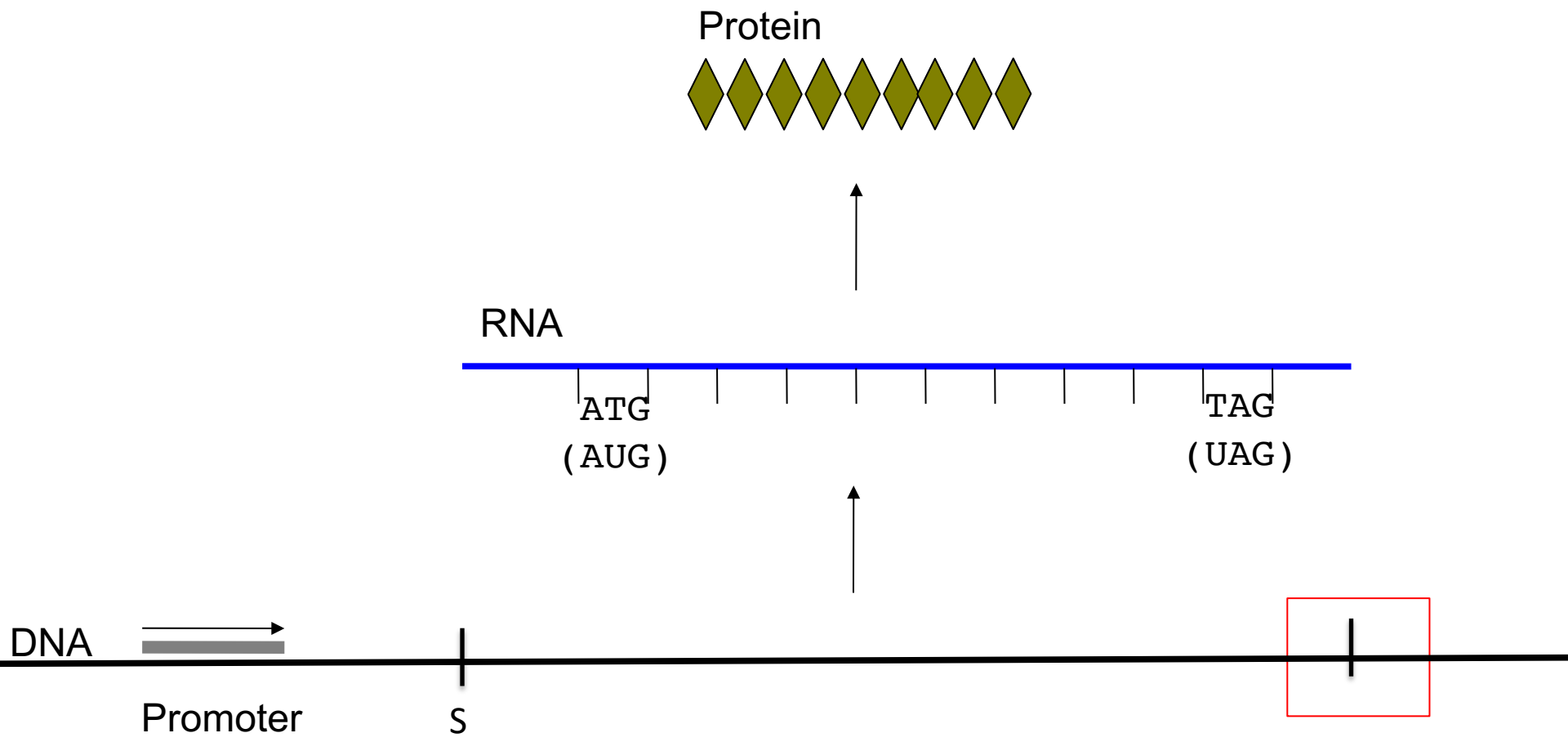
Homework: gene finding in a region of DNA unique to salmonella



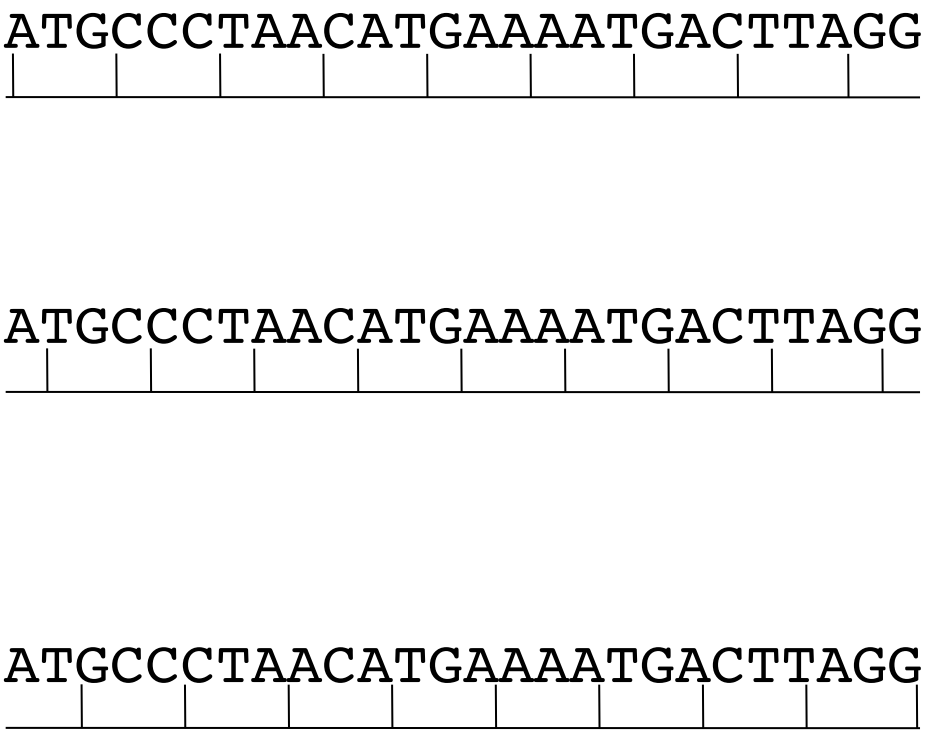
Salmonella pathogenicity island 1



The central dogma in a nutshell



Finding open reading frames (ORFs): check all 3 reading frames



Genes can occur on either strand

ATG = start codon
TGA, TAG, TAA = Stop codons

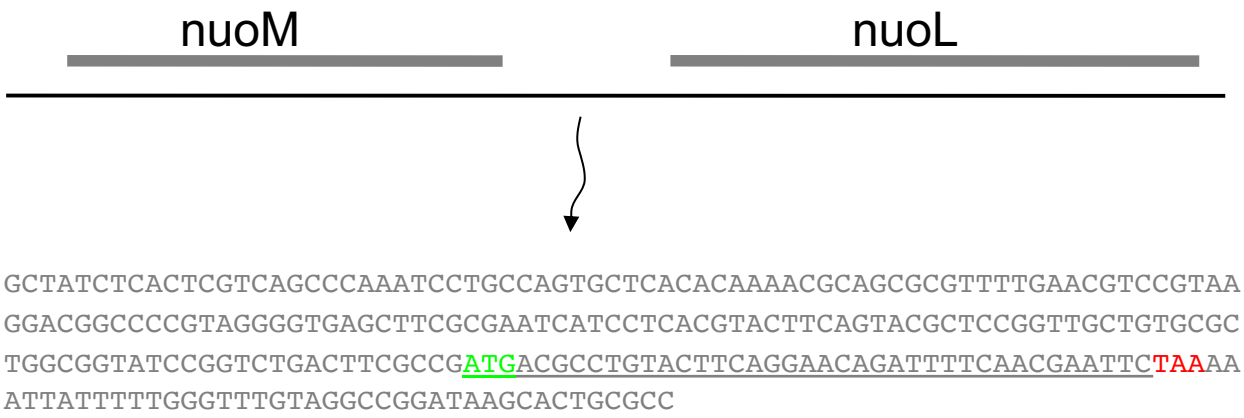
Gene 1: coding strand is on top

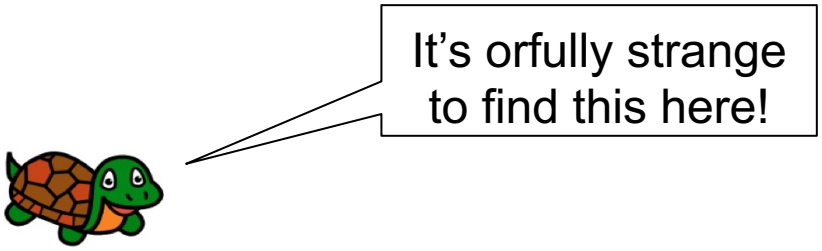
```
5' – AATGCCGTGCTTGTAGACGTAGGCTTAGATCGTCATGGG – 3'
3' – TTACGGCACGAACATCTGCATCCGAATCTAGCAGTACCC – 5'
```

Gene 2: coding strand is on bottom

```
5' – AATGCCGTGCTTGTAGACGTAGGCTTAGATCGTCATGGG – 3’

Reverse:    GGGTACTGCTAGATTCGGATGCAGATGTTCGTGCCGTAA

Complement: CCCATGACGATCTAAGCCTACGTCTACAAGCACGGCATT
```

Noncoding sequence between Salmonella genes





The unfortunate truth

- Not every ATG is a start codon
- Not every ORF is a gene

How can we separate gene ORFs from ORFs due to random chance?

https://memegenerator.net/instance/75366649/

A simple gene finding strategy

A sequence of interest

```
AATGGGCCGACCAAGGCGACATAGACGCGAATCGGACCAGACGCCGGCTCACCTGTTCATCTACCTTTCTG
CGTTGGCGCTAAAAGTTAACGATCGGGCCCTGCGCCGAAACGAAACGTCAGGAATCGACAAATACCAAGTA
TCTAAGCTACGGGATAAGCCCCCCCTCGCGAGAGAGGGGAAGGGGTCAATATTTCCCTGGCCGACTGACAA
TGGAGTGTACTTACCGGTATACAGTTTGTACTCTACAGCCATCGCTGTCTTACGACGTATTCGGGGCATTT
CAACATGCTGTCTCTCAGGAGTTTTCGCGCGCTGAAAGAACTCCCATCTAAACCCTG
```

ORF:
318 nucs

Randomly shuffled versions of this sequence

```
CGCTAGGCACGAAAGGATGGCGTCCCAACATATCAACGAGGTACGTTTGTGGAAGGCCCCGTATTACCGTC
AGAGACCTGGTACGAGGGTGACTATTTAACGCGGGAGCCCCAAGGAAAACTCAAGCAAAGCCGGCATCTTT
GTCGGGTACAGCGCTGTAACTGCGACCGTGATTGGAATCAACACGGATGACCGTGAAGGCGTGTTTGCCAG
CTACCACCCCTGATCCCCGGTCTCTCTTTGCCTGGGTTTATAGCTCAAAACTGTATCACGCGTTTAAAAGC
AACAACTGTAACGGCATACCCCCGAATTCCCCGTACCAGACGACGTTAATGCTTTCC
```

Longest ORF:
153 nucs

```
CTGCCCTCGCGACGTAAAGGCCTACCCTTATTCGGGCGCTGGTGTCTGGTGTCTGCACCTTGTACGATTTA
ATCCGTCTTACGCACCGCGGGTGTCAGTGCAAAACGACTTGGGCTTACAGACATGAATCACGGAACTCTGA
AGTATGGGTCGACCAGTCCACATTATGGAGGGAGCCAGAGTCCAACCCGGGAGGCGGGGCACCACACGCGG
TATTTTAAGAGGAACCACGCTTGATCACCAACGGAAAGTAGCCGCTAAATTATCGTCAATCTACCCTCAAA
CACAAAACCTCGGCTGAACGTCATATTCGAAAAGCTCTACATTTCGGGTTCAGGCCC
```

Longest ORF:
156 nucs

Don't forget to look at the reverse complements!

Modules and the `import` statement

```
>>> L = ['A','C','G','G','T','C','A’]

>>> L
['A', 'C', 'G', 'G', 'T', 'C', 'A']

>>> import random

>>> random.shuffle(L)

>>> L
['C', 'T', 'A', 'A', 'C', 'G', 'G’]
```

Homework bonus: look and say

- The look-and-say sequence…

1

11

21

1211

111221

What's next!?

Homework bonus: average length to ATG vs. AAA

'CGAGGCGCGGATATCTGGTTTACCCGTACATACTACATTG**ATG**TTGTA...'

Name _____

Worksheet

A Prime Example of Looping!

```
>>> this_many_primes(5)
[2, 3, 5, 7, 11]

def prime(k):
    '''Returns True if k is prime and False otherwise'''
    for d in range(2, k):
        if k % d == 0:
            return False
    return True

def this_many_primes(n):
    '''Returns a list of the first n primes.'''
```