



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
```

```
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
```

```
>>> mystery(1)
```

```
>>> mystery(6)
```

```
>>> mystery(16)
```

Repeat until this expression is no longer True



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
```

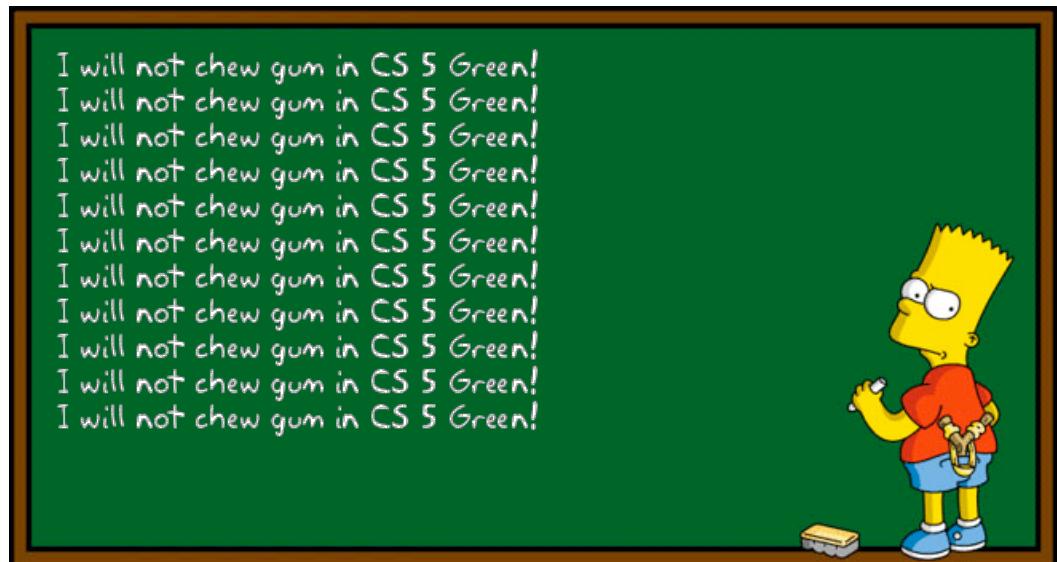


while loops

```
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

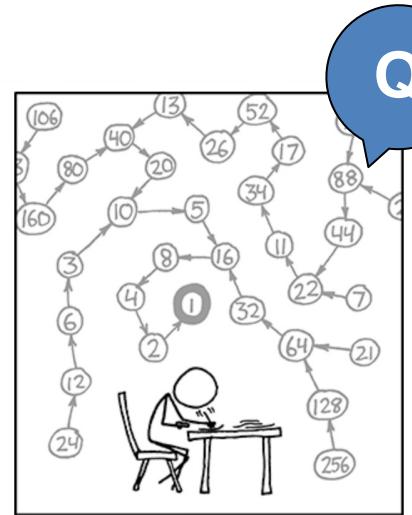
```
def collatz(n):
    '''Applies the collatz function to n.'''
    if n % 2 == 0:
        return n/2
    else:
        return 3*n +1
```

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

```
return counter
```

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

5 → 16 → 8 → 4 → 2 → 1



THE COLLATZ CONJECTURE STATES THAT IF YOU PICK A NUMBER, AND IF IT'S EVEN DIVIDE IT BY TWO AND IF IT'S ODD MULTIPLY IT BY THREE AND ADD ONE, AND YOU REPEAT THIS PROCEDURE LONG ENOUGH, EVENTUALLY YOUR FRIENDS WILL STOP CALLING TO SEE IF YOU WANT TO HANG OUT.

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

A Prime Example of Looping!

Worksheet

Q

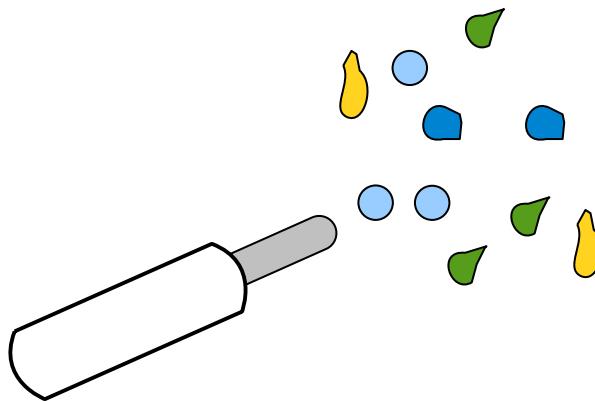
```
>>> this_many_primes(5)
[2, 3, 5, 7, 11]
```

```
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

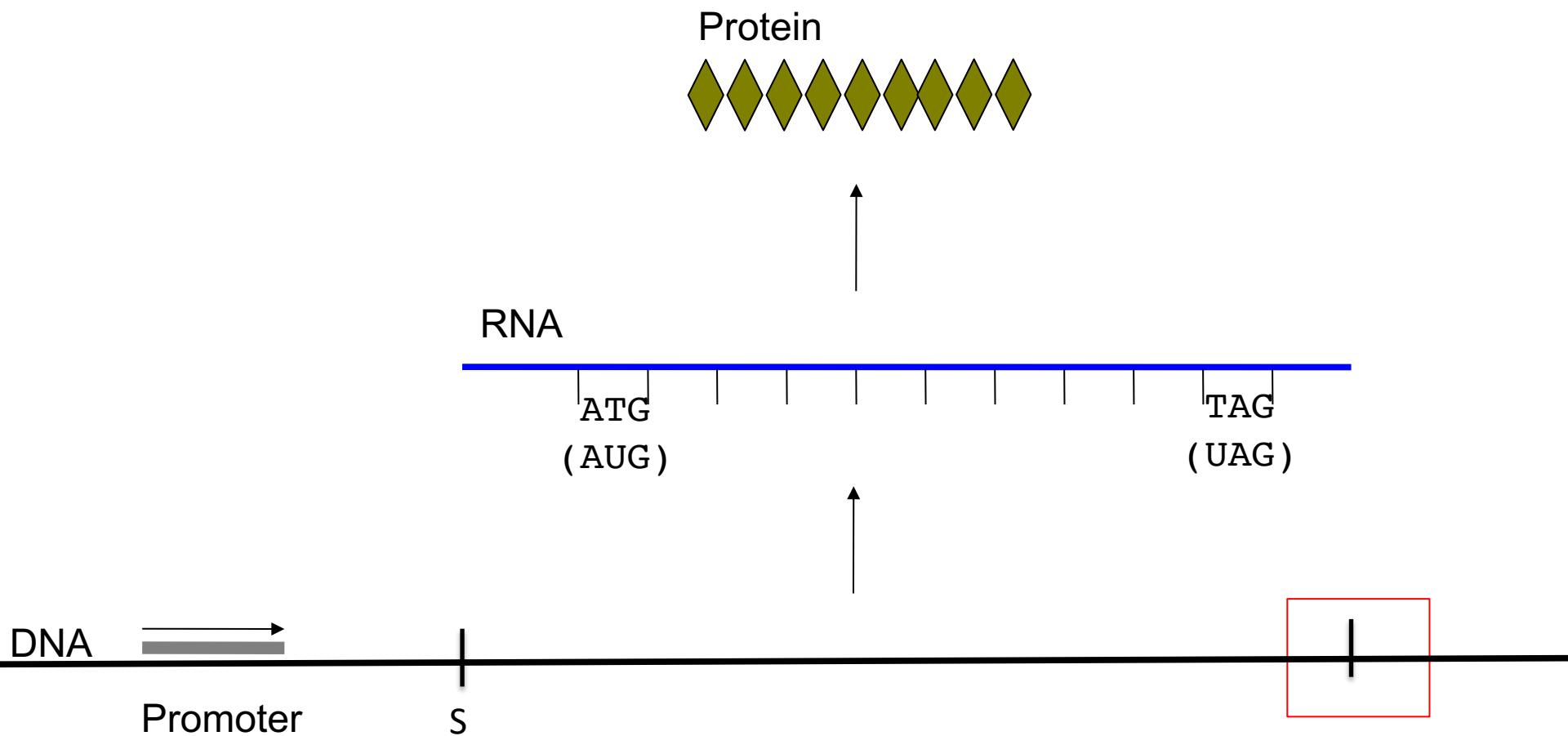


Salmonella pathogenicity island 1

The screenshot shows the National Library of Medicine's NCBI search interface. The search bar contains the identifier "X73525". Below the search bar, a COVID-19 information banner is visible. The main results section indicates "Results found in 4 databases". Under the "NUCLEOTIDE SEQUENCE" heading, the entry for "S.typhimurium genes for surface presentation of antigens" is listed. It specifies "Salmonella enterica subsp. enterica serovar Typhimurium", "6,387 bp genomic sequence", and "X73525.1". There are "FASTA" and "BLAST" links, and a "Download" button.



The central dogma in a nutshell



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

ATGCCCTAACATGAAAATGACTTAGG


ATGCCCTAACATGAAAATGACTTAGG


ATGCCCTAACATGAAAATGACTTAGG


Genes can occur on either strand

ATG = start codon

TGA, TAG, TAA = Stop codons

Gene 1: coding
strand is on top

5' - AATGCCGTGCTTG**TAG**ACGTAGGCTTAGATCGTCATGGG - 3'
3' - TTACGGCACGAACATCTGCATCCG**AAT**CTAGCA**GT**ACCC - 5'

Gene 2: coding
strand is on bottom

5' - AATGCCGTGCTTGTAGACGTAGGCTTAGATCGTCATGGG - 3'

Reverse: GGGTACTGCTAGATTGGATGCAGATGTTCGTGCCGTAA

Complement: CCC**ATG**ACGATC**TAAG**GCCTACGTCTACAAGCACGGCATT

Noncoding sequence between Salmonella genes



GCTATCTCACTCGTCAGCCAAATCCTGCCAGTGCTCACACAAAACGCAGCGCGTTGAACGTCCGTAA
GGACGGCCCCGTAGGGGTGAGCTTCGCGAATCATCCTCACGTACTTCAGTACGCTCCGGTTGCTGTGCGC
TGGCGGTATCCGGTCTGACTTCGCCGATGACGCCTGTACTTCAGGAACAGATTTCCAACGAATTTAAAA
ATTATTTTGGGTTGTAGGCCGGATAAGCACTGCGCC

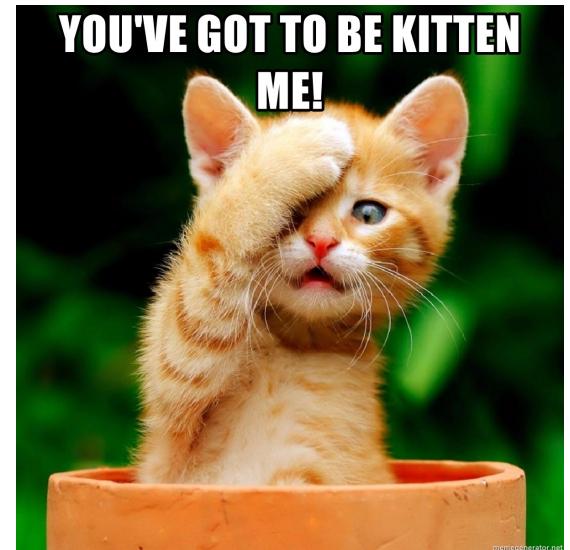


It's orfully strange
to find this here!

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?



A simple gene finding strategy

A sequence of interest

AATGGGCCGACCAAGGCGACATAGACGCGAATCGGACCAGACGCCGGCTCACCTGTTCATCTACCTTCTG
CGTTGGCGCTAAAAGTTAACGATCGGGCCCTGCGCCGAAACGAAACGTCAGGAATCGACAAATACCAAGTA
TCTAAGCTACGGGATAAGCCCCCCTCGCGAGAGAGGGGAAGGGGTCAATATTCCCTGGCCACTGACAA
TGGAGTGTACTTACCGGTATACAGTTGTACTCTACAGCCATCGCTGTCTACGACGTATTGGGGCATT
CAACATGCTGTCTCAGGAGTTTCGCGCGC TGA AAGAACTCCCCTAAACCCCTG

ORF:
318 nucs

Randomly shuffled versions of this sequence

CGCTAGGCACGAAAGGATGGCGTCCAACATATCAACGAGGTACGTTGTGGAAGGCCCCGTATTACCGTC
AGAGACCTGGTACGGAGGTGACTATTTAACGCGGGAGCCCCAAGGAAAACCTCAAGCAAAGCCGGCATCTT
GTCGGGTACAGCGCTGTAACCGTGATTGGAATCAACACGGATGACCGTGAAGGCGTGTGTTGCCAG
CTACCACCCCTGATCCCCGGTCTCTCTTGCCTGGTTATAGCTAAAACGTATCACGCGTTAAAAGC
AACAACTGTAACGGCATACCCCCGAATTCCCCGTACCGACGACGTTAATGCTTCC

Longest ORF:
153 nucs

CTGCCCTCGCGACGTAAAGGCCTACCCCTATTCGGCGCTGGTGTCTGGTGTCTGCACCTGTACGATT
ATCCGTCTTACGCACCGCGGGTGTCACTGCAAAACGACTTGGCTTACAGACATGAATCACGAACTCTGA
AGTATG GGT CGACCAGTCCACATTATGGAGGGAGCCAGAGTCCAACCCGGAGGCAGGGCACCACACCGCG
TATTTTAAGAGGAACCACGCTTGATCACCAACGGAAAGTAGCCGCTAAATTATCGTCAATCTACCCCTCAA
CACAAAACCTCGGC TGA AC GT CAT ATT CG AAA AG CT AC ATT CG GG TT CAGG C CC

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



'CGAGGCGCGGATATCTGGTTACCCGTACATACTACATTG**ATG**TTGTA...' '

Name _____

Worksheet

Q

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.'''

```