Weather: 98% chance of weather today
Sports: Grandmother of nine makes hole in one

**News Briefs**

- Spam proven to be even healthier than chocolate! (page 42)
- For loops shown to be secret of all happiness (page 42)
- Page 42 no longer printed, says editor of the Gazette. “Nobody reads it.” (page 42)

**The CS 5 Green Gazette**

**Computer Scientists Unveil New High Energy “Bit Accelerator”**

(Claremont, AP): A team of CS professors at Harvey Mudd College have announced that they have built a new high energy “bit accelerator” that promises to shed light on the hypothesized existence of sub-bit particles. The accelerator is a Python program that spins a bit (a 0 or a 1) in for a loop at very high speeds. The bit is then bombarded by docstrings, causing it break apart into its constituent sub-bit particles. “Although 0’s probably contain nothing, we believe that 1’s may contain fract-ions,” said one excited CS professor. Some researchers feel that the fract-ions may themselves be made of even smaller particles called numerators and denominators, but this is highly speculative and may require many more years of experimental and theoretical research. “This theory of numerators and denominators is not a rational theory,” said one expert. Other experts disagreed. “I think that there is a 1 in 2 chance that there is something to this notion of numerators and denominators,” said one expert, Professor Ray Sheeyo of the Pasadena Institute of Technology.
A word on grading

- How it works: auto-grading coupled with human grading
- Style points
- Your part: check your scores every week, and contact Kevin if there are problems (kherrera@hmc.edu)
Due to historical horizontal transfer events, the GC content inside pathogenicity islands sometimes differs from the rest of the genome.

Biology note: pathogenicity islands and GC content
Review: the virtues of negative thinking!

```
myList = [42, 47, 23, [3.141, 2.718], 5]

>>> myList[len(myList)-3]
23

>>> myList[-3]
23

>>> myList[-2:]
[[3.141, 2.718], 5]
```
Review: the virtues of negative thinking!

myString = "I luv spam"

>>> myString[::1]
'I luv spa'

>>> myString[-3:]
'pam'
Review: two types of for loop

def spamify(L):
    '''Add "n spam" to every string in list L.'''
    newL = []
    for s in L:
        newL.append(s + "n spam")
    return newL

>>> spamify(["eggs","sausage","oatmeal"])
['eggsn spam', 'sausagen spam', 'oatmealn spam']

For loop goes directly over the list L

Green eggs n spam!
def spamCount(S):
    '''Count occurrences of "spam" in input S.'''
    counter = 0
    for i in range(len(S)):
        if S[i:i+4] == "spam":
            counter = counter + 1
    return counter

>>> spamCount("gspamtspammspamn")
3
An alternate use of \texttt{in}

```python
>>> for num \texttt{in} \texttt{range}(1, 100): # what we've seen before
... 

>>> 42 \texttt{in} \[3, 67, 42, 18, 2001\] # new use
True

>>> 42 \texttt{in} \[13, 33, 300\]
False

>>> \texttt{food} = ["carrots","coffee","arugula","spam"]
>>> if "spam" \texttt{in} \texttt{food}: \texttt{print}("Yay!!!")
... 
Yay!!!

>>> "bio" \texttt{in} "symbiont"
True
```
Displaying output for the user: the print function

def verbose():
    print('Prof Wu likes spam. ')
    print('How much does she like spam? ')
    print(42)

>>> verbose()
Prof Wu likes spam.
How much does she like spam?
42
def printCodons(DNAstring):
    for i in range(0, len(DNAstring), 3):
        print("Next codon: ", DNAstring[i:i+3])
# no return statement necessary!

>>> printCodons("AAATTTGGGC")
Next codon: AAA
Next codon: TTT
Next codon: GGG
Next codon: C

What colorful codons you have!
return vs print...

def dbl(x):
    return 2 * x

def happy(input):
    y = dbl(input)
    return 2 * y

>>> happy(4)
16

def trbl(x):
    print(2 * x)
    return

def sad(input):
    y = trbl(input)
    return 2 * y

>>> sad(4)
8
TypeError: unsupported operand type(s) for *: 'int' and 'NoneType'
Professor P.I. Pette: calculate median genome size in intracellular pathogens vs. free livers

<table>
<thead>
<tr>
<th>speciesList</th>
<th>habitatList</th>
<th>genomeSizeList</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Agrobacterium tumefaciens C58&quot;</td>
<td>[&quot;free living&quot;]</td>
<td>5100</td>
</tr>
<tr>
<td>&quot;Chlamidia psittaci 1H&quot;</td>
<td>[&quot;intracellular&quot;]</td>
<td>1450</td>
</tr>
<tr>
<td>&quot;Salmonella typhimurium&quot;</td>
<td>[&quot;intracellular&quot;, &quot;free living&quot;]</td>
<td>4780</td>
</tr>
<tr>
<td>&quot;Lactococcus lactis C2&quot;</td>
<td>[&quot;free living&quot;]</td>
<td>2500</td>
</tr>
<tr>
<td>&quot;Escherichia coli K12&quot;</td>
<td>[&quot;free living&quot;, &quot;commensal&quot;]</td>
<td>4670</td>
</tr>
</tbody>
</table>

http://www.genomesize.com/prokaryotes/table1/
The data (in Python)

```python
habitatList = [["free living"],
               ["intracellular"],
               ["intracellular", "free living"],
               ["free living"],
               ["free living", "commensal"],
               ["intracellular"],
               ["free living", "commensal"],
               ["free living"],
               ["free living"],
               ["free living"],
               ["free living", "commensal"],
               ["free living"],
               ["intracellular"],
               ["free living", "commensal"],
               ["free living", "commensal"],
               ["free living", "commensal"]]

genomeSizeList = [5100,
                   1450,
                   4780,
                   2500,
                   4670,
                   1600,
                   4140,
                   4330,
                   6400,
                   6700,
                   4592,
                   3100,
                   1120,
                   7490,
                   4200]
```
Nick’s code (calculates median genome size in intracellular pathogens vs. free livers)

```python
from bacGenData import *

def gsize(habitatList, genomeSizeList):
    a = []
    for i in range(len(genomeSizeList)):
        if 'intracellular' in habitatList[i]:
            a.append(genomeSizeList[i])
    a.sort()
    if len(a) % 2 != 0:
        midInd = int((len(a) - 1) / 2)
        print('intracellular', 1.0 * a[midInd])
    else:
        y = int((len(a) / 2) - 1)
        b = int(len(a) / 2)
        print('intracellular', (a[y] + a[b]) / 2.0)
    z = []
    for i in range(len(genomeSizeList)):
        if 'free living' in habitatList[i]:
            z.append(genomeSizeList[i])
    z.sort()
    if len(z) % 2 != 0:
        midInd = int((len(z) - 1) / 2)
        print('free living', 1.0 * z[midInd])
    else:
        y = int((len(z) / 2) - 1)
        b = int(len(z) / 2)
        print('free living', (z[y] + z[b]) / 2.0)
```
Prof Pette now wants to look at **exclusively** intracellular or free living species. But Nick is out of town...
def gsize(habitatList, genomeSizeList):
    '''Calculate and print median genome size in intracellular pathogens and also in free livers.'''

    intraSizes=findSizes("intracellular",habitatList,genomeSizeList)
    freeSizes=findSizes("free living",habitatList,genomeSizeList)

    print("intracellular", median(intraSizes))
    print("free living", median(freeSizes))
def median(L):
    '''Calculates the median of list L.''
    L.sort()
    if len(L) % 2 != 0:
        # odd num elements, take middle one
        midInd = int((len(L) - 1) / 2)
        return 1.0 * L[midInd]
    else:
        # even num elements
        midInd1 = int((len(L) / 2) - 1)
        midInd2 = int(len(L) / 2)
        return (L[midInd1] + L[midInd2]) / 2.0

[2, 7, 18, 23, 500]  [2, 7, 18, 23]

>>> median([500, 18, 2, 23, 7])
18.0
def findSizes(habitat, habitatList, genomeSizeList):
    '''Find all species which live in habitat, and return a list of their genome sizes.'''
    outList = []

    return outList

>>> findSizes("commensal", habitatList, genomeSizeList)
[4670, 4140, 4592, 7490, 4200]
def findSizes(habitat, habitatList, genomeSizeList):
    '''Find all species which live in habitat, and return a list of their genome sizes.''

    outList = []

    for i in range(len(genomeSizeList)):
        if habitat in habitatList[i]:
            outList.append(genomeSizeList[i])

    return outList

>>> findSizes("commensal", habitatList, genomeSizeList)
[4670, 4140, 4592, 7490, 4200]
Advantages of modular programming

• Simpler to understand
• Less opportunity for errors
• Easier to modify in future
The central dogma in a nutshell

Protein
RNA
ATG
TAG
DNA
Promoter
S
Finding open reading frames (ORFs): check all 3 reading frames

ATGCCCTAACATGAAAATGACTTTAGG

ATGCCCTAACATGAAAATGACTTTAGG

ATGCCCTAACATGAAAATGACTTTAGG
Finding extremz!

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

dictionary = ['abdomen',
              'abdominal',
              'abduct',
              'abduction',
              'aberration',
              'abet',
              'abhor',
              'abhorrence',
              'abhorrent',
              'abide',
              'abiding',
              'ability',
              'abject',
              'ablate',
              ...
              ...
              etc.
              ...
              ...]
def extremz(wordList):
    '''Find and return the word with the most z's'''
def extremz(wordList):
    '''Find and return the word with the most z's'''
    bestCount=0
    bestWord=""
    for word in wordList:
        count=z(word)
        if count>bestCount:
            bestCount=count
            bestWord=word
    return bestWord

This is a common “programming motif”. 