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 Melinda if there are problems (mhlim@g.hmc.edu)
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 **in**

```python
>>> for num in range(1, 100): # what we’ve seen before
...

>>> 42 in [3, 67, 42, 18, 2001] # new use
True

>>> 42 in [13, 33, 300]
False

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

>>> "bio" in "symbiont"
True
```
Displaying output for the user: the `print` function

def verbose():
    print('Prof Bush likes spam.')
    print('How much does he like spam?')
    x = 42
    print('His favorite number: ', x, '!!!')

>>> verbose()
Prof Bush likes spam.
How much does he like spam?
His favorite number: 42!!!
Displaying output for the user: the print function

def printCodons(DNAstring):
    for i in range(0, len(DNAstring), 3):
        print("Next codon: ", DNAstring[i:i+3])
    # no return statement necessary!

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

What colorful codons you have!
def \texttt{dbl}(x): \\
\quad \texttt{return } 2 \times x \\

def \texttt{happy}(\texttt{input}): \\
\quad y = \texttt{dbl}(\texttt{input}) \\
\quad \texttt{return } 2 \times y \\

>>> \texttt{happy}(4) \\
16 \\

def \texttt{trbl}(x): \\
\quad \texttt{print}(2 \times x) \\
\quad \texttt{return} \\

def \texttt{sad}(\texttt{input}): \\
\quad y = \texttt{trbl}(\texttt{input}) \\
\quad \texttt{return } 2 \times y \\

>>> \texttt{sad}(4) \\
8 \\
\texttt{TypeError: unsupported operand type(s) for *: 'int' and 'NoneType' \n}
Advantages of modular programming
Advantages of modular programming

• Simpler to read understand
• Easier to test and debug
• Easier to modify in future
• Easier to reuse parts of the code
Pi.py
(Modularity and top-down design!)

- “Write a program to compute an approximation of pi using the Monte Carlo method. The program should take the number of darts as input, report the approximation of pi, and draw the board showing the dart throws.”
- What are parts of this problem which can/should each have their own functions?
- Let’s write the function signatures and docstrings!
def main(numDarts):
    """ Takes a number numDarts as input and returns an approximation of pi using a monte carlo simulation with that number of darts. """

def throw(n):
    """ Takes a number darts to be thrown using random uniform coordinates in a 2x2 box with x ranging from -1 to 1 and y from -1 to 1. Returns a list of (x, y) coordinates of darts. """

def numInsideCircle(dList):
    """ Takes a list of dart coordinates as input and returns the # that landed on or inside the circle. """

def drawPicture(dartList):
    """ Draws the dartboard and the darts """
Monty Hall

*Let’s make a deal* 1963-1986

inspiring the “Monty Hall paradox”
```python
import random

def monty():
    '''Simulates one round of the Monty Hall game without switching.'''
cardoor = random.choice([1, 2, 3])
guessNum = int(input("Which door would you like me to open? "))
if guessNum == cardoor:
    print("You've won a CAR!")
else:
    print("You've won a goat!")
```

DEMO!
```python
def niceMonty():
    """Simulates one round of the Monty Hall game with switching."
    cardoor = random.choice([1, 2, 3])
    guessNum = int(input("Which door would you like me to open? "))
    if cardoor == 1:
        if guessNum == 1:
            open = random.choice([2, 3])
        elif guessNum == 2:
            open = 3
        elif guessNum == 3:
            open = 2
    elif cardoor == 2:
        if guessNum == 1:
            open = 3
        elif guessNum == 2:
            open = random.choice([1, 3])
        elif guessNum == 3:
            open = 1
    elif cardoor == 3:
        if guessNum == 1:
            open = 2
        elif guessNum == 2:
            open = 1
        elif guessNum == 3:
            open = random.choice([1, 2])
    print("Look, there's nothing behind door ", open)
    response = input("Would you like to switch doors? (y or n) ")
    if response == 'y':
        guessNum = int(input("Which door would you like? "))
    if guessNum == cardoor:
        print("You've won a CAR!")
    else:
        print("You've won a goat!")
```
import random

def noswitch():
    cardoor = random.choice([1, 2, 3])
    guessNum = random.choice([1, 2, 3])
    if guessNum == cardoor:
        return 1  # We won 1 car
    else:
        return 0  # We won 0 cars

def switch():
    cardoor = random.choice([1, 2, 3])
    guessNum = random.choice([1, 2, 3])
    if cardoor == guessNum:
        # We guessed correctly but we are switching!
        return 0
    else:
        # We guessed a door X, the car is behind Y, so Monty opens Z
        # and we switch to Y
        return 1

def marilyn(games):
    print("No switching strategy...")
    cars = 0
    for game in range(0, games):
        cars = cars + noswitch()
    print("You won a car this many times: ", cars, " out of ", games, " games")
    print("The win-lose ratio was: ", 1.0*cars/games)
    print("Switching strategy...")
    cars = 0
    for game in range(0, games):
        cars = cars + switch()
    print("You won a car this many times: ", cars, " out of ", games, " games")
    print("The win-lose ratio was: ", 1.0*cars/games)
What’s the explanation?
“I can understand how a flower and a bee might slowly become, either simultaneously or one after the other, modified and adapted in the most perfect manner to each other, by the continued preservation of individuals presenting mutual and slightly favourable deviations of structure.”

Charles Darwin, The Origin of Species
Fig-Wasp Mutualism

From A. P. Jackson, “Cophylogeny of the Ficus Microcosm,” Biological Reviews, 79,
Reconciling gene and species trees

(a) Gene tree in the species tree
(b) The species tree
(c) The gene tree

From Yufeng Yu, University of Connecticut
Why should we care about reconciling evolutionary trees?

• Key fundamental discoveries in evolution
• New insights into origins of diseases
• Discoveries about the origins of parasites and impact on crops
**Phylogenetic Trees**

**Input:** DNA or protein sequences for each species

**Output:** Species tree

Boletus violaceofuscus AF457403
Boletus separans AF457404
Boletus gertrudiae AF457407
Strobilomyces sp. EU685109
Strobilomyces floccopus AY684155
Strobilomyces sp. DQ534627
Spongiforma thailandica EU685108
Spongiforma squarepantsii HQ724510
Spongiforma squarepantsii HQ724509
Porphyrellus sordidus DQ534644
Porphyrellus porphyrosporus DQ534643
Porphyrellus pseudopersicus EU685108
Jane (HMC 2010-18)
tchhikers guide to the Galápagos: co-logeography of Galápagos mockingbirds and parasites

Paquita EA Hoeck³, Lukas F Keller³ and Vincent S Smith¹

Parasites are evolutionary hitchhikers whose phylogenies often track the evolutionary history of their hosts. Congruence in the evolutionary history of closely associated lineages can be explained through a va
Jane 3 is Popular!

About 977,000,000 results (0.25 seconds)
The “great?” $\times$scape