“Dodds and Medero to be Replaced by Two Penguins,” says HMC Dean

Claremont (AP): Beginning next week, CS 5 will be taught by a pair of penguins, announced an HMC dean. “Penguins are very smart,” said the dean, “and they are also quite adorable, which is more than we can say about the CS 5 faculty. Plus, we won’t need to heat their offices in the winter.”
This Week

Homework 2:
• Reading on AI and Jeopardy
• Lab: Fractal art
• Problem 2: Higher Order Functions
• Problem 3: Sequence Alignment
• Problem 4: RNA Folding

More practice with use-it-or-lose-it!
A note on Getting Help…

So many places to get help on CS5 homework!
And another note on Illness…

Please e-mail the prof as soon as you feel bad!
Comparing DNA via Longest Common Subsequence (LCS)

AGGACAT
ATTACGAT

>>> LCS("AGGACAT", "ATTACGAT")
5

>>> LCS("can", "man!")
2
Recursive Approach...

```python
def LCS(s1, s2):
    if BASE CASE:
        ???
    else:
        LCS("spam", "sam!")
```

Try this in your notes!
Solution follows
Recursive Approach...

def LCS(s1, s2):
    if s1 == "" or s2 == ""
        return 0
    else:
        LCS("spam", "sam!")
Recursive Approach...

def LCS(s1, s2):
    if s1 == "" or s2 == ""
        return 0
    else:
        if s1[0] == s2[0]:  # DO THE FIRST SYMBOLS MATCH?
            return 1 + ???
        else:

LCS("spam", "sam!")
Recursive Approach...

def LCS(s1, s2):
    if s1 == "" or s2 == "":
        return 0
    else:
        if s1[0] == s2[0]:  # DO THE FIRST SYMBOLS MATCH?
            return 1 + LCS(s1[1:], s2[1:])
        else:
            return 0

LCS("spam", "sam!")
Recursive Approach...

def LCS(s1, s2):
    if s1 == "" or s2 == "":
        return 0
    else:
        if s1[0] == s2[0]:  # DO THE FIRST SYMBOLS MATCH?
            return 1 + LCS(s1[1:], s2[1:])
        else:
            return max(LCS(s1, s2[1:]), LCS(s1[1:], s2))

LCS("spam", "sam!")

What does it actually do when it runs?
Turtle Graphics

forward(100)
right(90)
Etch-a-Sketch amazingness…

No way this is real…
except that it is!
Turtle graphics are built into Python!

```python
>>> import turtle
>>> turtle.forward(50)
>>> turtle.right(90)
>>> turtle.backward(50)
```

Problem 2 has a link to the turtle documentation.
Fractals
“I Wonder About Trees” – Robert Frost

>>> svTree(128, 6)

“We wonder about Robert Frost” - Trees
```python
>>> svTree(100, 3)
```

- **trunk length**
- **recursion level**
>>> svTree(100, 3)

100 long!
```python
>>> svTree(100, 3)

svTree(50, 2)

100 long!
```
>>> svTree(100, 3)

svTree(50, 2)

100 long!
>>> svTree(100, 3)

svTree(25, 1)

svTree(50, 2)

100 long!
>>> svTree(100, 3)

svTree(25, 1)

svTree(50, 2)

100 long!
>>> `svTree(100, 3)`

`svTree(25, 1)`

`svTree(50, 2)`

100 long!
>>> svTree(100, 3)

svTree(25, 1)

svTree(50, 2)

100 long!
>>> svTree(100, 3)

svTree(25, 1)

svTree(50, 2)

100 long!

trunk length
recursion level
>>> svTree(100, 3)

100 long!

svTree(25, 1)

svTree(50, 2)

trunk length

recursion level
>>> svTree(100, 3)

svTree(25, 1)

svTree(50, 2)

100 long!
>>> `svTree(100, 3)`

```
svTree(50, 2)
```

```
svTree(25, 1)
```

100 long!
>>> svTree(100, 3)

svTree(25, 1)

svTree(50, 2)

100 long!
>>> svTree(100, 3)

svTree(25, 1)

svTree(50, 2)

100 long!
>>> svTree(100, 3)

svTree(25, 1)

svTree(50, 2)

100 long!
SnowFlake Fractals

The Koch Snowflake Fractal:

level 0  level 1  level 2  level 3  level 4  level 5
Snowflake Fractals

The Koch Snowflake Fractal:

level 0

level 1

level 2

level 3

level 4

level 5

level 0

level 1
And now for something completely different…

(…or at least it will seem different until we see that it’s not!)
Tuples ("Immutable Lists")

```python
>>> foo = (42, 'hello', (5, 'spam'), 'penguin')
>>> foo
(42, 'hello', (5, 'spam'), 'penguin')
>>> foo[0]
42
>>> foo[-1]
'penguin'
>>> foo[0:2]
(42, 'hello')
>>> foo[0:1]
(42,)
>>> foo[0] = 100
BARF!!!
```
"Julie", "Zach", and "Alien" are called the "keys" in the dictionary. Any \texttt{immutable} object can be a key.
Dictionaries

>>> D = {}
>>> D["Julie"] = "chocolate"
>>> D["Zach"] = "coffee"
>>> D["Alien"] = 42
>>> D["Julie"]
'chocolate'
>>> D["Alien"]
42
>>> D["Suicide Squad"]
BARF!

>>> D
{ 'Julie': 'chocolate', 'Zach': 'coffee', 'Alien': 42 }
Sometimes We Need to Make More Than 2 Recursive Calls!
Giigle maps

Inf = float("inf")

FiveCities = ["A", "B", "C", "D", "E"]

FiveDists = {("A","A") : 0, ("A","B") : 1, ("A", "C") : 3, ("A", "D") : 7 , ("A", "E") : Inf,
            ("C", "A") : Inf, ("C", "B") : Inf, ("C", "C") : 0, ("C", "D") : 2, ("C", "E") : 13,
            ("D", "A") : Inf, ("D", "B") : Inf, ("D", "C") : Inf, ("D", "D") : 0, ("D", "E") : 5,

>>> FiveDists[ ("B", "C") ]
42
Finding Shortest Paths
Giigle maps

All roads point east!

Shortest path?
Is greed good?
How does the use-it-or-lose-it idea get used here?
Inf = float("inf")

FiveCities = ["A", "B", "C", "D", "E"]

FiveDists = {
    ("C", "A") : Inf, ("C", "B") : Inf, ("C", "C") : 0, ("C", "D") : 2, ("C", "E") : 13,
    ("D", "A") : Inf, ("D", "B") : Inf, ("D", "C") : Inf, ("D", "D") : 0, ("D", "E") : 5,
}

>>> shortestPath (FiveCities, FiveDists)
10

>>> shortestPath (["C", "D", "E"], FiveDists)
7

>>> shortestPath (["E"], FiveDists)
0
def shortestPath (Cities, Distances):
    '''Returns the length of the shortest path
    from the leftmost to the rightmost city in
    the Cities list.'''

    if BLAH:
        return BLAH BLAH
    else:
        return BLAH BLAH BLAH BLAH

Inf = float("inf")

FiveCities = ["A", "B", "C", "D", "E"]

FiveDist = {
    ("C", "A") : Inf, ("C", "B") : Inf, ("C", "C") : 0, ("C", "D") : 2, ("C", "E") : 13,
    ("D", "A") : Inf, ("D", "B") : Inf, ("D", "C") : Inf, ("D", "D") : 0, ("D", "E") : 5,
}