CS 5 Black: Loops at last!
Recursive optometry?
Recursion is fantastic...
... but loops are nice too!
What’s up for the second half of the course?

- Writing some “bigger” programs
  - Games
  - Data compression
- Object-oriented programming
  - AI applications!
- Fundamental limits of computing!
- Final project
def hanoi(Disks, FromPeg, ToPeg):
    if Disks == 1:
        print FromPeg
        print ToPeg
        return
    else:
        OtherPeg = 6 - FromPeg - ToPeg
        hanoi(Disks - 1, FromPeg, OtherPeg)
        hanoi(1, FromPeg, ToPeg)
        hanoi(Disks - 1, OtherPeg, ToPeg)
        return

it's a Hmmm-burger!
def hanoi(Disks, FromPeg, ToPeg):
    if Disks == 1:
        print FromPeg
        print ToPeg
        return
    else:
        OtherPeg = 6 - FromPeg - ToPeg
        hanoi(Disks - 1, FromPeg, OtherPeg)
        hanoi(1, FromPeg, ToPeg)
        hanoi(Disks - 1, OtherPeg, ToPeg)
        return

Two Hmmm-burgers!
def hanoi(Disks, FromPeg, ToPeg):
    if Disks == 1:
        print FromPeg
        print ToPeg
        return
    else:
        OtherPeg = 6 - FromPeg - ToPeg
        hanoi(Disks - 1, FromPeg, OtherPeg)
        hanoi(1, FromPeg, ToPeg)
        hanoi(Disks - 1, OtherPeg, ToPeg)
        return

What do you call an error in a Hmmm program?
def hanoi(Disks, FromPeg, ToPeg):
    if Disks == 1:
        print FromPeg
        print ToPeg
        return
    else:
        OtherPeg = 6 - FromPeg - ToPeg
        hanoi(Disks - 1, FromPeg, OtherPeg)
        hanoi(1, FromPeg, ToPeg)
        hanoi(Disks - 1, OtherPeg, ToPeg)
        return

Notice: 5-eyed Aliens have a strange sense of Hmmm-our
Who uses assembly language!?

Other than students in CS 5, CS 105, Engineering 85, and Engineering 155?!
def leppard(inputString):
    outputString = ""
    for symbol in inputString:
        if symbol == "o":
            outputString = outputString + "ooo"
        else:
            outputString = outputString + symbol
    print(outputString)

>>> leppard("hello")

>>> leppard("hello to you")
Mystery 2

I love a good mystery!

```python
vowels = ['a', 'e', 'i', 'o', 'u']

def spamify(word):
    for i in range(len(word)):
        if word[i] not in vowels:
            return word[0:i] + "spam" + word[i+1:]
    return word  # is this where you want it?

>>> spamify("oui")

>>> spamify("hello")

>>> spamify("aardvark")
```
vowels = ['a', 'e', 'i', 'o', 'u']

def spamify(word):
    for i in range(len(word)):
        if word[i] not in vowels:
            return word[0:i] + "spam" + word[i+1:]
    return word

>>> spamify("oui")

>>> spamify("hello")

>>> spamify("aardvark")

How many times does the return happen?
for <variable> in <iterable>:
    Do stuff!

for symbol in "blahblahblah":

for element in [1, 2, 3, 4]:

for index in range(42):

Three uses of for!

I’d like to see four uses of three!
Many languages have no loops!
More Mysteries!

def mystery1(n):
    for k in range(1, n):
        if k*k == n: return True
    return False

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

Scary! Does that modify the n in range(1, n) in the loop?
z detector

>>> z("I like zyzzyvas!")
3
>>> z("I am opposed to the letter after y")
0

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

def z(input):
    counter = 0
    for i in range(len(input)):
        if input[i] == 'z':
            counter = counter + 1
    return counter

The “direct” method

The “indirect” or “index” method
Spam counter!

```python
def spamCount(input):
    counter = 0
    for letter in input:
        ???
```
Spam counter!

```python
def spamCount(input):
    counter = 0
    for i in range(len(input)):
        spamCount("I like spam with spamspamspam!")
        4
              012345678911111111
              0123456

>>> spamCount("spamityspampampam!")
2

```
Spam counter!

```python
def spamCount(input):
    counter = 0
    for i in range(len(input)):
        if input[i:i+4] == "spam":
            counter += 1
```

And, look no else is just fine!
Functions that return lists

>>> squares(5)
[1, 4, 9, 16, 25]

def squares(n):
    output = []
    for x in range(1, n+1):
        output = output + [x*x]  # upgrade to listhood!
    return output

def squares(n):
    output = []
    for x in range(1, n+1):
        output.append(x*x)
    return output
Spam finder!

```python
def spamFinder(input):
    # Try this in your worksheet!
```

```python
>>> spamFinder("spamspamity")
[0, 4]
```

```python
>>> spamFinder("ssspam!")
[2]
```
Spam finder!

01234567891
    0

>>> spamFinder("spamspamity")
[0, 4]

>>> spamFinder("ssspam!")
[2]

def spamFinder(input):
    output = []
    for i in range(len(input)):
        if input[i:i+4] == "spam":
            output.append(i)
    return output

Try this in your worksheet!
def returnCodons(DNAstring):
    codonList = []
    for i in range(0, len(DNAstring), 3):
        codonList.append(DNAstring[i:i+3])
    return codonList

>>> returnCodons("AAATTTTG GCC")
[“AAA”, “TTT”, “GGG”, “C”]
Breaking out of a for loop!

for item in myList:
    # code here!
# more code here!

for item in myList:
    if item == 42: break
    else: print("Shucks!")
# more code here

for item in myList:
    if item == 42: return
    else: print("Shucks!")
# more code here
A subtle issue...

```python
def main():
    L = list(range(5))  # [0, 1, 2, 3, 4]
    L = incl1(L)
    print(L)

def incl1(myList):
    for item in myList:
        item += 1  # item = item + 1
    return myList
```
A subtle issue on mutability

def main():
    L = list(range(5))  # [0, 1, 2, 3, 4]
    L = inc2(L)
    print(L)

def inc2(myList):
    for i in range(len(myList)):
        myList[i] += 1
    return myList
Tuples are immutable!

def main():
    L = (0, 1, 2, 3, 4)
    L = inc2(L)
    print(L)

def inc2(myList):
    for i in range(len(myList)):
        myList[i] += 1
    return myList

What happens when we run this?!
Returning a shiny new list...

def main():
    L = list(range(5))  # [0, 1, 2, 3, 4]
    M = inc3(L)
    print(L)
    print(M)

def inc3(myList):
    newList = []
    for item in myList:
        newList.append(item+1)
    return newList
for loops versus higher-order functions...

# myList is a list of numbers

def inc(myList):
    newList = []
    for item in myList:
        newList.append(item+1)
    return newList

def incA(myList):
    return list(map(lambda X: X+1, myList))

def incB(myList):
    return [X+1 for X in myList]
While loops!

```python
def collatz(n):
    """Compute the collatz function on n """
    if n % 2 == 0:
        return n // 2
    else:
        return 3 * n + 1

def hmt(n):
    """Returns the number of iterations required to collatz n until it gets to 1. """
    counter = 0
    while n != 1:
        n = collatz(n)
        counter += 1
    return counter

>>> hmtList(10)
[0, 1, 7, 2, 5, 8, 16, 3, 19, 6]

def hmtList(n):
    """Returns a list of the number of times that the number 1, 2, ..., n need to be Collatzed to get to 1. """
```

Write this function and keep it short and sweet!
While loops!

def collatz(n):
    """ Compute the collatz function on n """
    if n % 2 == 0:
        return n // 2
    else:
        return 3 * n + 1

def hmt(n):
    """ Returns the number of iterations required to collatz n until it gets to 1. """
    counter = 0
    while n != 1:
        n = collatz(n)
        counter += 1
    return counter

>>> hmtList(10)
[0, 1, 7, 2, 5, 8, 16, 3, 19, 6]

def hmtList(n):
    """ Returns a list of the number of times that the number 1, 2, ..., n need to be Collatzed to get to 1. """
    return list(map(lambda X: hmt(X), range(1, n+1)))

DEMO!!
Paul Erdos

“Mathematics is not yet ready for such problems.”

10 → 5 → 16 → 8 → 4 → 2 → 1

Paul Erdős, 1913-1996
import random

def play():
    print("Welcome!")
    secret = random.choice(range(1, 100))
    numGuesses = 0
    userGuess = 0
    while userGuess != secret:
        userGuess = int(input("Enter your guess: "))
        numGuesses += 1
        if userGuess > secret:
            print("Too high")
        elif userGuess < secret:
            print("Too low")
    print("You got", secret, "in", numGuesses, "guesses!")
    print("Thanks for playing!")
for or while?

• Use a for loop when you know how many iterations you’ll need...

```python
for i in range(len(myList)):
    blah, blah, blah
```

• Use a while loop when you can’t know in advance how many iterations you’ll need...

```python
while userInput != 42:
    blah, blah, blah
```
Try-als and Tribulations

def safeDivide(x, y):
    try:
        return x / y
    except:
        print("Don't DO that!")
        return float("inf")

def saferDivide(x, y):
    try:
        return x / y
    except ZeroDivisionError:
        print("Don't DO that!")
        return float("inf")
An Exceptional Program

def safestDivide(x, y):
    try:
        return x / y
    except ZeroDivisionError:
        print("Division by zero!")
        return float("inf")  # Infinity
    except TypeError:
        print("Oops! Something here wasn't a number!")
        return float("nan")  # Not a number
    except:
        print("Something bad happened here. I'm clueless!")
        raise  # "Throw" an exception

Demo!
def index(myList, myIndex):
    try:
        return myList[myIndex]
    except IndexError:
        print("Oops, index out of bounds!")
        return None
    except TypeError:
        print("Oops! Something was of the wrong type!")
        return None
    except:
        print("Utter panic!")
        raise

Demo!
def foo(myList):
    """ Takes a list of positive numbers as input and returns their sum """
    sum = 0
    for item in myList:
        assert (item > 0)
        sum += item
    return sum

Demo!
def dumbgame():
    while True:
        userInput = input("Enter a positive integer: ")
        try:
            userNum = int(userInput)
            assert(userNum > 0)
        except (ValueError, AssertionError):
            print("Oops! Try again!")
            continue
        print("Good job!", userNum, "is a lovely positive integer!")

Input Validation

Demo!
def get():
    while True:
        userInput = input("Enter a positive integer: ")
        try:
            userNum = int(userInput)
            assert(userNum > 0)
            break
        except (ValueError, AssertionError):
            print("Oops! Try again!")
            continue
    print("Done with loop")
    return userNum
5. Built-in Exceptions

In Python, all exceptions must be instances of a class that derives with an `except` clause that mentions a particular class, that clause and from that class (but not exception classes from which it is derived). ed via subclassing are never equivalent, even if they have the same

The built-in exceptions listed below can be generated by the interpreter. The exceptions are unordered according to their 

- **exception `AssertionError`**
  Raised when an `assert` statement fails.

- **exception `AttributeError`**
  Raised when an attribute reference (see not support attribute references or attri:

- **exception `EOFError`**
  Raised when the `input()` function hits a `io.IOBase.read()` and `io.IOBase.read`

- **exception `FloatingPointError`**
  Raised when a floating point operation when Python is configured with the `--w` fined in the `pyconfig.h` file.
The Game of Nim!
Good Design

Programs must be written for people to read, and only incidentally for machines to execute. - Abelson and Sussman

1. Design your program “on paper” first. Identify the separate logical parts and the arguments and return value for each part.

2. Once your design is established, write the function “signatures” (function name, arguments) and docstrings.

3. Fill in the code for a function, test that function carefully, and proceed only when you are convinced that the function works correctly.

4. Use descriptive function and variable names (how about x, stuff, florg, jimbob?).

5. Don’t replicate functionality. Break out repeated code into helper functions.

6. Keep your code readable and use comments to help! # Here’s one now!

7. Use whitespace liberally.

8. Avoid global variables unless absolutely necessary! Instead, pass each function just what it needs.

9. Use recursion, list comprehension, and functional constructs (e.g. map, reduce, filter, lambda) where appropriate.
Objective: Write a tic-tac-toe program that lets two human players play, and stops when a player has won.

Functions:

main(): Welcomes user, plays a game, asks if we want to play again
welcome(): Prints the welcome message
playGame(): Maintains a board and plays one game
getMove(board, player): Queries the player (1 or 2) for her/his move and changes the board accordingly
printBoard(board): Takes a board as argument and displays it
gameOver(board): Evaluates a board to see if game over
debug = True

def main():
    """Play tic-tac-toe with a human""
    welcome()
    while True:
        if debug: print("About to enter playGame")
        playGame()
        response = input("Would you like to play again? (y or n): ")
        if response not in ["y", "Y", "yes", "Yes", "Yup", "si", "oui", "youbetcha"]:  
            print("Bye")
            return

def welcome():
    """Prints the welcome message for the game.
    We might also print the rules for the game and any other
    information that the user might need to know.""
    print("Welcome to tic-tac-toe!")

def playGame():
    """Play one game of tic-tac-toe""
    if debug: print("Entering the playGame function")
    board = [[" ", " ", " "], [[" ", " ", " "], [[" ", " ", " "]]
    player = 1
    while not gameOver(board):
        print("The board looks like this:")
        printBoard(board)
        getMove(board, player)
        if player == 1:  
            player = 2  
        else:
            player = 1
# Tic-tac-toe by Ran Libeskind-Hadas

def main():
    """Play tic-tac-toe with a human""
    welcome()
    while True:
        if debug: print("About to enter playGame")
        playGame()
        response = input("Would you like to play again? (y or n): ")
        if response not in ["y", "Y", "yes", "Yes", "Yup", "si", "oui", "youbetcha"]:
            print("Bye")
            return

    def welcome():
        """Prints the welcome message for the game.
        We might also print the rules for the game and any other
        information that the user might need to know.""
        print("Welcome to tic-tac-toe!")

    def playGame():
        """Play one game of tic-tac-toe""
        if debug: print("Entering the playGame function")
        board = [[" ", " ", " "], [" ", " ", " "], [" ", " ", " "]]
        player = 1
        while not gameOver(board):
            print("The board looks like this:")
            printBoard(board)
            getMove(board, player)
            if player == 1:
                player = 2
            else:
                player = 1

        print("Game over!")

        return

        def printBoard(board):
            for row in board:
                print("| " + "| " + "| " + "| ")

        def getMove(board, player):
            print("Player " + str(player) + ":")
            row = input("Enter row (0-2): ")
            column = input("Enter column (0-2): ")
            if row.isdigit() and column.isdigit() and int(row) in range(3) and int(column) in range(3):
                row = int(row)
                column = int(column)
                board[row][column] = "X" if player == 1 else "O"
            else:
                print("Invalid move!")

        def gameOver(board):
            for row in board:
                if row.count("X") == 3 or row.count("O") == 3:
                    return True
            for col in range(3):
                if board[0][col] == board[1][col] == board[2][col] and board[0][col] in ["X", "O"]:
                    return True
            for row in range(3):
                if board[row][0] == board[row][1] == board[row][2] and board[row][0] in ["X", "O"]:
                    return True
            for row in range(3):
                for col in range(3):
                    if board[row][col] == ":":
                        board[row][col] = " 
            if board[0][0] == board[1][1] == board[2][2] and board[0][0] in ["X", "O"]:
                return True
            if board[0][2] == board[1][1] == board[2][0] and board[0][2] in ["X", "O"]:
                return True
            return False

What’s this?!
def gameOver(board):
    """Returns False if the game is NOT over. Otherwise, prints a message indicating which player has won and then returns True indicating that the game is over. THIS FUNCTION IS NOT IMPLEMENTED CORRECTLY!"""
    return False

def getMove(board, player):
    """Takes the board and the current player (1 or 2) as arguments. Asks the player for a move. If it's legitimate (position exists and is empty) updates the board. Otherwise, the player is queried again until a valid move is provided."""
    print("Player " + str(player) + "'s turn")
    while True:
        try:
            row = int(input("Enter the row: "))
            column = int(input("Enter the column: "))
        except:
            print("You must enter numbers from 0 to 2 for the row and column")
            continue
        if not 0 <= row <= 2 or not 0 <= column <= 2:
            print("That's not a valid location on the board! Try again.")
        elif board[row][column] != " ":
            print("That cell is already taken! Try again.")
        else:
            board[row][column] = str(player)
            break