CS 5 Penguins Fail to Return from Fall Break in Arctic

Claremont (AP): The two official CS 5 “Black” penguins failed to return from their fall-break trip to the Arctic Circle. “They went up to Northern Alaska for a little R&R before the second half of CS 5,” sniffled one of the CS 5 professors. “They said they were going to just chill and maybe play a few practical jokes. I can’t imagine what could have happened to them!”

Last photo taken of the CS 5 Penguins before their mysterious disappearance
Recursion is fantastic…
And is often “handy”…
What’s Up Next…

- Loop structures: **for** and **while**
- Writing some “bigger” programs
  Secret Sharing (cryptography)
  Games (Nim, Mastermind)
- Data compression

That’ll keep us entertained for a few weeks!
Loops!
I love a good mystery!

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")
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")
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!
import random

def play():
    print("Welcome!")
    secret = random.choice(range(1, 100))
    numGuesses = 0
    userGuess = 0
    while userGuess != secret:
        userGuess = eval(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!")
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")
def safestDivide(x, y):
    try:
        return x / y
    except ZeroDivisionError:
        print("Don't DO that!")
        return float("inf")
    except (ValueError, TypeError):
        print("That's just silly")
        return float("nan")
    except:
        print("I don't know what happened")
    raise

An Exceptional Program
import random

def play():
    print("Welcome!")
    secret = random.choice(range(1, 100))
    numGuesses = 0
    userGuess = 0
    while userGuess != secret:
        userGuess = input("Enter your guess: ")
        try:
            userGuess = int(userGuess)
        except ValueError:
            print("Please enter a number")
            continue
        numGuesses += 1
        if userGuess > secret:
            # Rest of program is the same...
Lab Problem: The Mandelbrot Set

Consider some complex number C

\[ z_0 = 0 \]

\[ z_{n+1} = z_n^2 + C \]

For which values of C does this not diverge?
Consider some complex number $C$

\[ z_0 = 0 \]

\[ z_{n+1} = z_n^2 + C \]

For which values of $C$ does this *not* diverge?
Rooter: A Methodology for the Typical Unification of Access Points and Redundancy

Jeremy Stribling, Daniel Aguayo and Maxwell Krohn

ABSTRACT

Many physicists would agree that, had it not been for congestion control, the evaluation of web browsers might never have occurred. In fact, few hackers worldwide would disagree with the essential unification of voice-over-IP and public-private key pair. In order to solve this riddle, we confirm that SMPs can be made stochastic, cacheable, and interposable.

I. INTRODUCTION

Many scholars would agree that, had it not been for active networks, the simulation of Lamport clocks might never have occurred. The notion that end-users synchronize with the

The rest of this paper is organized as follows. For starters, we motivate the need for fiber-optic cables. We place our work in context with the prior work in this area. To address this obstacle, we disprove that even though the much-touted autonomous algorithm for the construction of digital-to-analog converters by Jones [10] is NP-complete, object-oriented languages can be made signed, decentralized, and signed. Along these same lines, to accomplish this mission, we concentrate our efforts on showing that the famous ubiquitous algorithm for the exploration of robots by Sato et al. runs in \( \Omega((n + \log n)) \) time [22]. In the end, we conclude.

II. ARCHITECTURE
Training File: "I like spam. I like toast and spam. I eat ben and jerry's ice cream too."

First order Markov Dictionary:
I : like, like, eat
like : spam., toast
spam. : I, I
and : spam, jerry's

Generating “random” text:
"I like spam. I like spam."
"I eat ben and spam. I like toast and jerry's ice cream too."
Training File: Wikipedia essay on Huffman Compression

First order Markov sentences generated…

"Huffman was a source symbol."

"Huffman became a known as a character in a particular symbol frequencies agree with those used for each possible value of Engineering."
Training File: "I like spam. I like toast and spam. I eat ben and jerry's ice cream too."

First order Markov Dictionary:
- I : like, like, eat
- like : spam, toast
- spam. : I, I
- and : spam, jerry's

Second order Markov Dictionary:
- I like : spam., toast
- like spam. : I
- spam. I : like, eat
"Huffman coding is such a code is not produced by Huffman's algorithm."

"Huffman was able to design the most common characters using shorter strings of bits than are used for lossless data compression."
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 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 their 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
# Tic-tac-toe by Ran Libeskind-Hadas

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:  # Can this be done with clever arithmetic?
            player = 2
        else:
            player = 1
# Tic-tac-toe by Ran Libeskind-Hadas

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

What's this?!

How 'bout:
    row = [" ", " ", " "]
    board = [row, row, row]

Or

    board = [[" ", " ", " "]]*3