**Assignment 4/slot.py**

```
import os, sys
from time import sleep
from random import randint as rint

def clr():
    os.system('cls' if os.name == 'nt' else 'clear')
clr()

# Define some variables
global delay, fullRollovers

delay = 0.05            # next number delay
fullRollovers = 2       # how many full "rollovers" the "wheels" do before landing on the number
waitAfterRoll = True    # used for testing - eliminates delays after each "pull" of the "lever"


class NotPossibleError(Exception):
    pass


# Write some things down from previous programs to remember
# sys.stdout.flush()
# sys.stdout.write('\rSomething')


# When given the user's input (individual), a "yes" condition (individual or
# list), and a "no" condition (individual or list), the function will check if
# the input matches the "yes" condition(s), which will return True, or the "no"
# condition(s), which will return False. If neither matches, it will simply
# return the value.
def becomeBool(var, yes, no):
    def tests(var, yes, no):
        for value in var:
            if value in yes:
                return True
            elif value in no:
                return False
            else:
                return value

    if var == '':   # pressing enter returns an empty string, which, when "listified," will return
        var = '\n'  # an empty list. Which won't work right.


    # Convert everything to lists to make equality checks easier
    var = [var]
    yes = [b.lower() for b in yes]
    no = [c.lower() for c in no]
```

```
49
50      return tests(var, yes, no)
51
52
53  # Make the intro a function with a function inside that can be called to repeat
    the intro
54  def intro():
55      def askBegin():
56          begin = becomeBool(input('\nShall we begin? (Y/n) '), ['y', 'yes', '\n'],
    ['n', 'no'])
57
58          if begin == False:
59              exit(0)
60          elif begin == True:
61              return slot()
62          else:
63              print('Unknown option ' + str(begin) + '.')
64              askBegin()
65
66      print('\n             -  -  -  Welcome to the Python Slot Machine!  -  -  -
    \n')
67      print('This is a virtual slot machine written in Python.')
68      return askBegin()
69
70
71
72  # This function creates the virtual "wheels" to "spin". It repeats a full list
73  # of numbers, 0 thru 9, a certain number of times based on the integer stored
74  # in `rollovers`. It then appends more numbers to each "wheel" sequentially
75  # (starting at 0) until it reaches the randomly generated numbers.
76  def makeWheels(numbers, rollovers):
77      wheelTemplate = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
78
79      wheel1 = wheelTemplate * rollovers
80      wheel2 = wheelTemplate * rollovers
81      wheel3 = wheelTemplate * rollovers
82
83      for a in range(numbers[0] + 1):
84          wheel1.append(a)
85      for b in range(numbers[1] + 1):
86          wheel2.append(b)
87      for c in range(numbers[2] + 1):
88          wheel3.append(c)
89
90      return wheel1, wheel2, wheel3
91
92
93  # When passed a list of lists, it finds which sublist has the largest length,
94  # then returns that max length.
95  def biggestList(lists):
96      lens = []
97      big = 0
98
99      for l in lists:
```

```
        lens.append(len(l))

    for i in range(len(lens)):
        if lens[i] > big:
            big = lens[i]

    return big


# This function finds the number of equal numbers in the list of random
# numbers.
def match(nums):
    uniqueNums = set(nums)
    count = 0

    for num in uniqueNums:
        if nums.count(num) > 1:
            count += nums.count(num)

    return (count if count > 1 else 1)


# This is the actual slot machine program.
def slot():
    tries = 1
    goAgain = True
    wins = [0, 0, 0]    # tries, doubles, jackpots


    while goAgain:
        clr()
        print('\n              -  -  -  Welcome to the Python Slot Machine!  -  -  -             \n')      # Reprint the title bar for each run
        print(f'Try #{tries}: ({wins[1]} doubles, {wins[2]} jackpots) ($ in = {(tries - 1) * 0.05:.2f}, $ out = {((wins[1] * 2) + (wins[2] * 4)) * 0.05:.2f})')
        sys.stdout.flush()

        numbers = [rint(0,9), rint(0,9), rint(0,9)]     # make the random number list
        #numbers = [0, 4, 7]    # for testing only
        wheel1, wheel2, wheel3 = makeWheels(numbers, fullRollovers)     # fetches the "wheels"
        visibleNums = [0, 0, 0] # used to display the "visible" numbers on the "wheels"
        biggest = biggestList([wheel1, wheel2, wheel3])
        j = 0
        done = 0

        # iterates through each "wheel" until it reaches the last number (which should
        # be the randomly generated number), then uses `sys.stdout.write()` to
        # essentially reprint over the same line to produce a "rolling" effect.
        for i in range(biggest + 1):
            if i <= (fullRollovers * 10):
                visibleNums = [wheel1[i], wheel2[i], wheel3[i]]
```

```
149             else:
150                 visibleNums = [min(j, numbers[0]), min(j, numbers[1]), min(j,
    numbers[2])]
151                 j += 1
152             #print(visibleNums)
153             sys.stdout.write(f'\r{visibleNums[0]} {visibleNums[1]} {visibleNums[2]
    }')
154 
155             if visibleNums == numbers and done <= 2:    # This prevents obscenely
    long delays that sometimes appear after the wheels
156                 continue                                # stop. If the wheels are
    displaying the same numbers at least three times in a
157                                                         # row, the delay is
    skipped. This shouldn't be needed if things are working
158             sleep(delay)                                # properly.
159 
160 
161 
162         sleep(.25 if waitAfterRoll else 0); print()     # Delays (if allowed by
    `waitAfterRoll`), then prints a new line.
163 
164         if (matches := match(visibleNums)) == 1:    # Uses the `walrus` operator
    to store the value of `matches` from the return of
165             print('No matches.')                    #`match(visibleNums)`, then
    immediately tests based on `matches`.
166         elif matches == 2:
167             print('Double!')
168             wins[1] += 1
169         elif matches == 3:
170             print('Jackpot!!')
171             wins[2] += 1
172         else:
173             # This code should not ever run. It will only execute if either less
    than one or
174             # more than three matches are detected, which is not possible, hence
    the
175             # `NotPossibleError` as defined on line 17.
176             if matches < 1:
177                 raise NotPossibleError('Less than one match was detected.')
178             elif matches > 3:
179                 raise NotPossibleError('More than three matches were detected.')
180             else:
181                 raise Exception('There were between one and three matches, but
    something else went wrong.')
182 
183         tries += 1
184         sleep(.5 if waitAfterRoll else 0)
185         def again():
186             if (goAgain := becomeBool(input('\nGo again (enter), or exit (E)? '),
    '\n', 'E')) == False:
187                 return False
188             elif goAgain == True:
189                 return True
190             else:
191                 print('Unknown option ' + str(goAgain) + '.')
192                 again()
```

```
193         goAgain = again()
194 
195     wins[0] = tries - 1
196     return wins
197 
198 
199 wins = intro()
200 clr()
201 print('\n             -  -  -  Welcome to the Python Slot Machine!  -  -  -
    \n')    # Reprint the title bar for the totals
202 print('Thanks for playing! Your score was:')
203 print(f'Doubles =  {wins[1]} * 2')
204 print(f'Jackpots = {wins[2]} * 4')
205 print(f'\nTOTAL =    {(wins[1] * 2) + (wins[2] * 4)}')
206 print(f'\nIf each pull were a nickel, this game would have cost you ${wins[0] *
    0.05:.2f}.')
207 print(f'If this game rewarded you with nickels, you would have gotten ${((wins[1]
    * 2) + (wins[2] * 4)) * 0.05:.2f}.')
208 print('\n')
209 
210 input('Press any key to exit...')
211 
```