

Anagram Advent Calendar - Python

Step 1 - Build the Scramble Word Function

This step will build the function that will scramble words.

We will require the random library from Python, this will be used to shuffle a list of indexes. Those indexes will be used to pick the letters in a random order.

- * import random at the top of the code.
- * define the scramble word function, with a single parameter which I have called word.
- * generate a list of letter indexes by using the range function, wrapped in the list function.
- * shuffle the list
- * create the variable that will hold the scrambled word
- * iterate through the letter indexes adding the letters to our scrambled word.
- * return the scrambled word.

When you have done this, the code should look like this

```
1  import random
2
3  # Define a separate function for scrambling a word
4  def scrambleWord(word):
5      # Generate a list of indexes, they are generated in order
6      letterIndexes = list(range(0, len(word)))
7
8      # Shuffle the indexes so they can be used to generate anagram
9      random.shuffle(letterIndexes)
10
11     # Start the scrambled value as empty string
12     scrambled = ""
13
14     # Work through the shuffled indexes, picking letters to append to our puzzle
15     for x in letterIndexes:
16         scrambled += word[x]
17
18     return scrambled
19
```

To test the function, call it a couple of times and print the results, for example.

```
print("Scrambled monkeys " + scrambleWord("monkeys"))
print("Scrambled turkey " + scrambleWord("turkey"))
```

```
Python 3.4.0 (default, Apr 11 2014, 13:05:11)
[GCC 4.8.2] on linux
```

```
Scrambled monkeys emsonky
Scrambled turkey eyutkr
```

Step 2 - Build Player Interaction

This step will show you how to ask the player for an answer and then check their answer.

- * Define a new function called testUser, it should have one parameter called puzzle
- * Ask the user for their attempt using the input function.
- * Check that the answer given equals the puzzle
- * Print congratulations or commiserations as appropriate.
- * Return True for success, return False for failure. This will be required later
- * Call the testUser function, passing in a fixed puzzle
- * Run the code to try it out

```
30 def testUser(puzzle):
31     answerGiven = input("Please type in the unscrambled " + scrambleWord(puzzle) + ": ")
32
33     # Check the answer
34     if puzzle == answerGiven:
35         print("Correct, please award yourself one gift")
36         return True
37     else:
38         print("Incorrect, sorry, please try again")
39         return False
40
41 testUser("strawberries")
42
```

Once you have done this, comment out the call to testUser() using the # symbol

```
Please type in the unscrambled atrbwisresre: strawberries
Correct, please award yourself one gift
sh-4.3$
```

Step 3 - Create Puzzle List

In order to provide a different puzzle for each day, we should create a list with many puzzles in. The following code will create a list variable with a set of festive words in.

```
35 # Compose the list of puzzles
36 puzzles=[ 'snowball',
37           'santa',
38           'holly',
39           'reindeer',
40           'presents',
41           'turkey',
42           'tinsel',
43           'robin',
44           'carols',
45           'pudding',
46           'wreath',
47           'stocking',
48           'star',
49           'angle',
50           'cracker',
51           'nativity',
52           'bauble',
53           'sheperd',
54           'stuffing',
55           'tree',
56           'sprouts',
57           'sleigh',
58
```

Step 4 - Solve Today's Puzzle

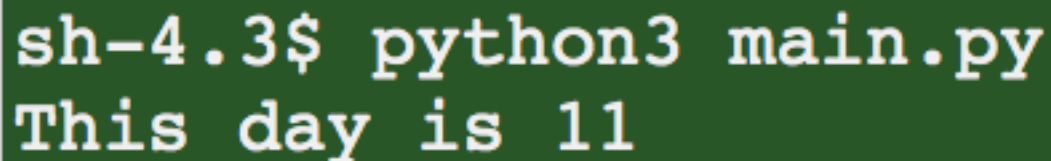
In order to get the current day, we will need the datetime library. We can then use this library to access the current day.

- * Add the import date/time library call to the top of the python code
- * Just underneath call the datetime.datetime.now() function, this returns a time structure with various fields required to define the current date and time.
- * Create a new variable called thisDay and use the value of timeNow.day.
- * Put a print statement in to check that it is working.

The top of the code should look like this

```
1 import random
2 import datetime
3
4 # Get hold of the day within the month
5 timeNow = datetime.datetime.now()
6 thisDay = timeNow.day
7
8 print("This day is " + str(thisDay))
9
```

Run the code and the following should be printed



```
sh-4.3$ python3 main.py
This day is 11
```

Now we need to present today's puzzle to the user. We need to add the following steps at the bottom of the code (after the puzzles list has been defined).

```
67
68 testUser(puzzles[thisDay])
69
```

Run the code again and you should be presented with a puzzle for the current day. If you run this code tomorrow it will present a different puzzle.

As before, comment out this line in preparation for the next stage. You can always comment it back in to retest things quickly.

Step 5 - Print Text Based Calendar

Now that we can present the user with a puzzle selected from our list, it would be nice to present all of the puzzles as hashed out values, the user can then pick their way through the ones they want, effectively 'opening' the doors.

- * In order to remember if the player has solved a particular day, a new list variable will be created containing boolean values (true/false). Create an empty list called solved.
- * Populate this list by appending False for each value of puzzle
- * Define a new function to print the puzzles in their current state.
- * In this function, loop round the puzzles by index
- * Compose a new variable called toPrint which is set to a row of hashes.
- * If the solved indicator shows the user has solved that puzzle, change the value of toPrint to the solved puzzle.
- * Print the toPrint value
- * Outside of the function definition, add a couple of lines just to test it, run the test and see what is printed.

The function should look like this

```
71
72 # Seed the solved indicators with False for all puzzles
73 solved = []
74 for puzzle in puzzles:
75     solved.append(False)
76
77 def printPuzzles():
78     for x in range(0, len(puzzles)):
79         toPrint = "#####"
80         if solved[x]:
81             toPrint = puzzles[x]
82         print("Puzzle for " + str(x + 1) + ":\t" + toPrint)
83
84 solved[23] = True
85 printPuzzles()
86
87 whichDay = 1
```

Comment out the test lines in preparation for the next and final step.

Step 6 - Loop Puzzles with User

Now that we have our print function, we should continue calling out, allowing the user to solve as many puzzles as they want to.

- * Define a variable called whichDay, this will be used to store the day the user has selected to solve.
- * Start a while loop which runs forever
- * Prompt the user for a day selection using the input function
- * If the user types in zero, break out of the loop
- * Check the day is valid, then present the puzzle for the selected day
- * If the user correctly solves the puzzle, the testUser function will return true and we should mark it as solved.
- * After the user has solved/not solved the puzzle, reprint the calendar.

Once completed, this interaction loop code should look like this

```
84 printPuzzles()
85
86 whichDay = 1
87 while True:
88     whichDay = int(input("Please type in which day to solve (type 0 to quit): "))
89     if (whichDay == 0):
90         break
91     if (whichDay <= thisDay):
92         if testUser(puzzles[whichDay]):
93             solved[whichDay] = True
94             printPuzzles()
95     else:
96         print("You aren't allowed to solve this one yet")
97
98
```

CONGRATULATIONS YOU HAVE NOW COMPLETED THE ADVENT CALENDAR

My own implementation can be found at <http://goo.gl/D0PdYD>