|  |  |
| --- | --- |
|  | Chapter 9 ½ |
| Extending Hangman |

Topics Covered In This Chapter:

* The dictionary data type
* Key-value pairs
* The keys() and values() dictionary methods
* Multiple variable assignment

The Hangman is much bigger than the Dragon Realm program, but it’s is also more sophisticated. It really helps to make a flow chart or small sketch to remember how you want everything to work. Now that you’ve created a basic Hangman game, let’s look at some ways you can extend it with new features.

After you’ve played Hangman a few times, you might think that six guesses aren't enough to get many of the words. You can easily give the player more guesses by adding more multi-line strings to the HANGMANPICS list.

Save your hangman.py program as hangman2.py, then add the following instructions:

58. ==========''', '''

59. +----+

60. | |

61. [O |

62. /|\ |

63. / \ |

64. |

65. ==========''', '''

66. +----+

67. | |

68. [O] |

69. /|\ |

70. / \ |

71. |

72. ==========''']

There are two new multi-line strings to the HANGMANPICS list, one with the hangman's left ear drawn, and the other with both ears drawn. Because the program will tell the player they have lost on line134 based on len(missedLetters) == len(HANGMANPICS) - 1, this is the only change you must make. The rest of the program works with the new HANGMANPICS list just fine.

You can also change the list of words by changing the words on line 59. Instead of animals, you could have colors:

59. words = 'red orange yellow green blue indigo violet white black brown'.split()

Or shapes:

59. words = 'square triangle rectangle circle ellipse rhombus trapezoid chevron pentagon hexagon septagon octagon'.split()

Or fruits:

59. words = 'apple orange lemon lime pear watermelon grape grapefruit cherry banana cantaloupe mango strawberry tomato'.split()

## Dictionaries

With some modification, you can change the code so that the Hangman game uses sets of words, such as animal, color, shape, or fruit. The program can tell the player which set (animal, color, shape, or fruit) the secret word is from.

To make this change, you will need a new data type called a dictionary. A dictionary is a collection of values like a list is. But instead of accessing the items in the dictionary with an integer index, you can access them with an index of any data type. For dictionaries, these indexes are called keys.

Dictionaries use { and } curly braces instead of [ and ] square brackets. Try entering the following into the interactive shell:

>>> spam = {'hello':'Hello there, how are you?', 4:'bacon', 'eggs':9999 }

The values between the curly braces are key-value pairs. The keys are on the left of the colon and the key’s values are on the right. You can access the values like items in lists by using the key. Try entering the following into the interactive shell:

>>> spam = {'hello':'Hello there, how are you?', 4:'bacon', 'eggs':9999}

>>> spam['hello']

'Hello there, how are you?'

>>> spam[4]

'bacon'

>>> spam[eggs]

9999

Instead of putting an integer between the square brackets, you can use, say, a string key. This will evaluate to the value for that key.

Getting the Size of Dictionaries with len()

You can get the number of key-value pairs in the dictionary with the len() function. Try entering the following into the interactive shell:

>>> stuff = {'hello':'Hello there, how are you?', 4:'bacon', 'spam':9999}

>>> len(stuff)

3

The Difference Between Dictionaries and Lists

Dictionaries can have keys of any data type, not just strings. But remember, because 0 and '0' are different values, they will be different keys. Try entering this into the interactive shell:

>>> spam = {'0':'a string', 0:'an integer'}

>>> spam[0]

'an integer'

>>> spam['0']

'a string'

The keys in dictionaries can also be looped over using a for loop. Try entering the following into the interactive shell.

>>> favorites = {'fruit':'apples', 'animal':'cats', 'number':42}

>>> for k in favorites:

... print(k)

fruit

number

animal

>>> for k in favorites:

... print(favorites[k])

apples

42

cats

Dictionaries are different from lists because the values inside them are unordered. The first item in a list named listStuff would be listStuff[0]. But there’s no “first” item in a dictionary, because dictionaries do not have any sort of order. Try entering the following into the interactive shell:

>>> favorites1 = {'fruit':'apples', 'number':42, 'animal':'cats'}

>>> favorites2 = {'animal':'cats', 'number':42, 'fruit':'apples'}

>>> favorites1 == favorites2

True

The expression favorites1 == favorites2 evaluates to True because dictionaries are unordered and considered equal if they have the same key-value pairs in them. Meanwhile, lists are ordered, so two lists with the same values in a different order are not equal to each other. Try entering this into the interactive shell:

>>> listFavs1 = ['apples', 'cats', 42]

>>> listFavs2 = ['cats', 42, 'apples']

>>> listFavs1 == listFavs2

False

Dictionaries have two useful methods, keys() and values(). These will return values of a type called dict\_keys and dict\_values, respectively. Much like range objects, values of those data types are returned by the list() function. Try entering the following into the interactive shell:

>>> favorites = {'fruit':'apples', 'animal':'cats', 'number':42}

>>> list(favorites.keys())

['fruit', 'number', 'animal']

>>> list(favorites.values())

['apples', 42, 'cats']

Sets of Words for Hangman

Let’s change the code in the Hangman game to support different sets of secret words. First, replace the value assigned to words with a dictionary whose keys are strings and values are lists of strings. The string method split() will return a list of strings with one word each.

59. words = {'Colors':'red orange yellow green blue indigo violet white black brown'.split(),

60. 'Shapes':'square triangle rectangle circle ellipse rhombus trapezoid chevron pentagon hexagon septagon octagon'.split(),

61. 'Fruits':'apple orange lemon lime pear watermelon grape grapefruit cherry banana cantaloupe mango strawberry tomato'.split(),

62. 'Animals':'bat bear beaver cat cougar crab deer dog donkey duck eagle fish frog goat leech lion lizard monkey moose mouse otter owl panda python rabbit rat shark sheep skunk squid tiger turkey turtle weasel whale wolf wombat zebra'.split()}

Lines 59 to 62 are across multiple lines in the source code, but they are still one assignment statement. The instruction doesn’t end until the final } curly brace on line 62.

## The random.choice() Function

The choice() function in the random module takes a list argument and returns a random value from it. This is similar to the what the previous getRandomWord() function did. You’ll use random.choice() in the new version of the getRandomWord() function.

To see how the random.choice() function works, try entering the following into the interactive shell:

>>> import random

>>> random.choice(['cat', 'dog', 'mouse'])

'mouse'

>>> random.choice(['cat', 'dog', 'mouse'])

'cat'

>>> random.choice([2, 22, 222, 223])

2

>>> random.choice([2, 22, 222, 223])

222

Change the getRandomWord() function so that its parameter will be a dictionary of lists of strings, instead of just a list of strings. Here is what the function originally looked like:

61. def getRandomWord(wordList):

62. # This function returns a random string from the passed list of strings.

63. wordIndex = random.randint(0, len(wordList) - 1)

64. return wordList[wordIndex]

Change the code in this function so that it looks like this:

64. def getRandomWord(wordDict):

65. # This function returns a random string from the passed dictionary of lists of strings, and the key also.

66. # First, randomly select a key from the dictionary:

67. wordKey = random.choice(list(wordDict.keys()))

68.

69. # Second, randomly select a word from the key's list in the dictionary:

70. wordIndex = random.randint(0, len(wordDict[wordKey]) - 1)

71.

72. return [wordDict[wordKey][wordIndex], wordKey]

The name of the wordList parameter is changed to wordDict to be more descriptive. Now instead of choosing a random word from a list of strings, first the function chooses a random key in the wordDict dictionary by calling random.choice().

And instead of returning the string wordList[wordIndex], the function returns a list with two items. The first item is wordDict[wordKey][wordIndex]. The second item is wordKey.

Evaluating a Dictionary of Lists

The wordDict[wordKey][wordIndex] expression on line 72 may look complicated, but it is just an expression you can evaluate one step at a time like anything else. First, imagine that wordKey had the value 'Fruits' (which was chosen on line 65) and wordIndex has the value 5 (chosen on line 68). Here is how wordDict[wordKey][wordIndex] would evaluate:

wordDict[wordKey][wordIndex]

▼

wordDict['Fruits'][wordIndex]

▼

['apple', 'orange', 'lemon', 'lime', 'pear', 'watermelon', 'grape', 'grapefruit', 'cherry', 'banana', 'cantaloupe', 'mango', 'strawberry', 'tomato'][wordIndex]

▼

['apple', 'orange', 'lemon', 'lime', 'pear', 'watermelon', 'grape', 'grapefruit', 'cherry', 'banana', 'cantaloupe', 'mango', 'strawberry', 'tomato'][5]

▼

'watermelon'

In the above case, the item in the list this function returns would be the string 'watermelon'. (Remember that indexes start at 0, so [5] refers to the 6 th item in the list, not the 5th.)

Because the getRandomWord() function now returns a list of two items instead of a string, secretWord will be assigned a list, not a string. You can assign these two items into two separate variables using multiple assignment. This is explained next.

## Multiple Assignment

Multiple assignment is a shortcut to specify multiple variables, separated by commas, on the left side of an assignment statement. Try entering the following into the interactive shell:

>>> a, b, c = ['apples', 'cats', 42]

>>> a

'apples'

>>> b

'cats'

>>> c

42

The above example is equivalent to the following assignment statements:

>>> a = ['apples', 'cats', 42][0]

>>> b = ['apples', 'cats', 42][1]

>>> c = ['apples', 'cats', 42][2]

You must put the same number of variables on the left side of the = assignment operator as there are items in the list on the right side. Python will automatically assign the first item's value in the list to the first variable, the second item's value to the second variable, and so on. But if you do not have the same number of variables and items, the Python interpreter will give you an error.

>>> a, b, c, d = ['apples', 'cats', 42, 10, 'hello']

Traceback (most recent call last):

File "<pyshell#8>", line 1, in <module>

a, b, c, d = ['apples', 'cats', 42, 10, 'hello']

ValueError: too many values to unpack

>>> a, b, c, d = ['apples', 'cats']

Traceback (most recent call last):

File "<pyshell#9>", line 1, in <module>

a, b, c = ['apples', 'cats']

ValueError: need more than 2 values to unpack

Change your code in Hangman on line 109 and 145 to use multiple assignment with the return value of getRandomWord():

108. correctLetters = ''

109. secretWord, secretKey = getRandomWord(words)

110. gameIsDone = False

...

144. gameIsDone = False

145. secretWord, secretKey = getRandomWord(words)

146. else:

Printing the Word Category for the Player

The last change you’ll make is to tell the player which set of words they are trying to guess. This way, when the player plays the game they will know if the secret word is an animal, color, shape, or fruit. Add this line of code after line 112. Here is the original code:

112. while True:

113. displayBoard(HANGMANPICS, missedLetters, correctLetters, secretWord)

Add the line so your program looks like this:

112. while True:

113. print('The secret word is in the set: ' + secretKey)

114. displayBoard(HANGMANPICS, missedLetters, correctLetters, secretWord)

Now you’re done with the changes to the Hangman program. Instead of just a single list of strings, the secret word is chosen from many different lists of strings. The program also tells the player which set of words the secret word is from. Try playing this new version. You can easily change the words dictionary on line 59 to include more sets of words.

Summary

We’re done with Hangman. Even after you’ve finished writing a game, you can always add more features after you learn more about Python programming.

Dictionaries are similar to lists except that they can use any type of value for an index, not just integers. The indexes in dictionaries are called keys.

Multiple assignment is a shortcut to assign multiple variables the values in a list.

Hangman was fairly advanced compared to the previous games in this book. But at this point, you know most of the basic concepts in writing programs: variables, loops, functions, and Python’s data types such as lists and dictionaries. The later programs in this book will still be a challenge to master, but you have finished the steepest part of the climb.