|  |  |
| --- | --- |
|  | Chapter 11 |
| Bagels |

Topics Covered In This Chapter:

* Augmented Assignment Operators, +=, -=, \*=, /=
* The random.shuffle() Function
* The sort() and join() List Methods
* String Interpolation (also called String Formatting)
* Conversion Specifier %s
* Nested Loops

In this chapter, you’ll learn a few new methods and functions that come with Python. You’ll also learn about augmented assignment operators and string interpolation. These things don’t let you do anything you couldn't do before, but they are nice shortcuts to make coding easier.

Bagels is a deduction game you can play with a friend. Your friend thinks up a random 3-digit number with no repeating digits, and you try to guess what the number is. After each guess, your friend gives you three types of clues:

* **Bagels** – None of the three digits you guessed is in the secret number.
* **Pico** – One of the digits is in the secret number, but your guess has the digit in the wrong place.
* **Fermi** – Your guess has a correct digit in the correct place.

You can get multiple clues after each guess. If the secret number is 456 and your guess is 546 the clues would be “fermi pico pico”. The 6 provides “fermi” and the 5 and 4 provide “pico pico”.

## Sample Run of Bagels

I am thinking of a 3-digit number. Try to guess what it is.

Here are some clues:

When I say: That means:

Pico One digit is correct but in the wrong position.

Fermi One digit is correct and in the right position.

Bagels No digit is correct.

I have thought up a number. You have 10 guesses to get it.

Guess #1:

123

Fermi

Guess #2:

453

Pico

Guess #3:

425

Fermi

Guess #4:

326

Bagels

Guess #5:

489

Bagels

Guess #6:

075

Fermi Fermi

Guess #7:

015

Fermi Pico

Guess #8:

175

You got it!

Do you want to play again? (yes or no)

no

## Source Code of Bagels

If you get errors after typing this code in, compare the code you typed to the book’s code with the online diff tool at http://invpy.com/diff/bagels.

bagels.py

1. import random

2. def getSecretNum(numDigits):

3. # Returns a string that is numDigits long, made up of unique random digits.

4. numbers = list(range(10))

5. random.shuffle(numbers)

6. secretNum = ''

7. for i in range(numDigits):

8. secretNum += str(numbers[i])

9. return secretNum

10.

11. def getClues(guess, secretNum):

12. # Returns a string with the pico, fermi, bagels clues to the user.

13. if guess == secretNum:

14. return 'You got it!'

15.

16. clue = []

17.

18. for i in range(len(guess)):

19. if guess[i] == secretNum[i]:

20. clue.append('Fermi')

21. elif guess[i] in secretNum:

22. clue.append('Pico')

23. if len(clue) == 0:

24. return 'Bagels'

25.

26. clue.sort()

27. return ' '.join(clue)

28.

29. def isOnlyDigits(num):

30. # Returns True if num is a string made up only of digits. Otherwise returns False.

31. if num == '':

32. return False

33.

34. for i in num:

35. if i not in '0 1 2 3 4 5 6 7 8 9'.split():

36. return False

37.

38. return True

39.

40. def playAgain():

41. # This function returns True if the player wants to play again, otherwise it returns False.

42. print('Do you want to play again? (yes or no)')

43. return input().lower().startswith('y')

44.

45. NUMDIGITS = 3

46. MAXGUESS = 10

47.

48. print('I am thinking of a %s-digit number. Try to guess what it is.' % (NUMDIGITS))

49. print('Here are some clues:')

50. print('When I say: That means:')

51. print(' Pico One digit is correct but in the wrong position.')

52. print(' Fermi One digit is correct and in the right position.')

53. print(' Bagels No digit is correct.')

54.

55. while True:

56. secretNum = getSecretNum(NUMDIGITS)

57. print('I have thought up a number. You have %s guesses to get it.' % (MAXGUESS))

58.

59. numGuesses = 1

60. while numGuesses <= MAXGUESS:

61. guess = ''

62. while len(guess) != NUMDIGITS or not isOnlyDigits(guess):

63. print('Guess #%s: ' % (numGuesses))

64. guess = input()

65.

66. clue = getClues(guess, secretNum)

67. print(clue)

68. numGuesses += 1

69.

70. if guess == secretNum:

71. break

72. if numGuesses > MAXGUESS:

73. print('You ran out of guesses. The answer was %s.' % (secretNum))

74.

75. if not playAgain():

76. break

Designing the Program

The flow chart in Figure 11-1 describes what happens in this game, and in what order they can happen.

How the Code Works

1. import random

2. def getSecretNum(numDigits):

3. # Returns a string that is numDigits long, made up of unique random digits.

At the start of the program, import the random module. Then define a function named getSecretNum(). The function makes a secret number that has only unique digits in it. Instead of only 3-digit secret numbers, the numDigits parameter lets the function make a secret number with any number of digits. For example, you can make a secret number of four or six digits by passing 4 or 6 for numDigits.

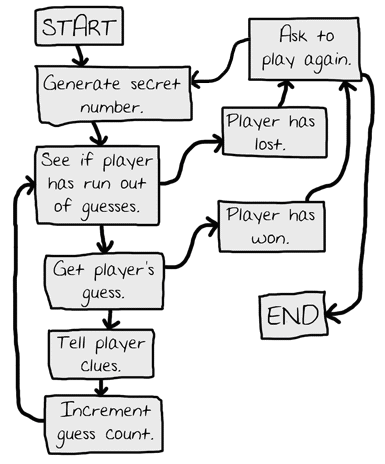


Figure 11-1: Flow chart for the Bagels game.

Shuffling a Unique Set of Digits

4. numbers = list(range(10))

5. random.shuffle(numbers)

Line 4’s list(range(10)) always evaluate to [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]. It’s just easier to type list(range(10)). The numbers variable contains a list of all ten digits.

## The random.shuffle() Function

The random.shuffle() function randomly changes the order of a list’s items. This function doesn’t return a value, but rather modifies the list you pass it “in place”. This is similar to the way the makeMove() function in the Tic Tac Toe chapter modified the list it was passed in place, rather than return a new list with the change. This is why you do **not** write code like numbers = random.shuffle(numbers).

Try experimenting with the random.shuffle() function by entering the following code into the interactive shell:

>>> import random

>>> spam = list(range(10))

>>> print(spam)

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> random.shuffle(spam)

>>> print(spam)

[3, 0, 5, 9, 6, 8, 2, 4, 1, 7]

>>> random.shuffle(spam)

>>> print(spam)

[1, 2, 5, 9, 4, 7, 0, 3, 6, 8]

>>> random.shuffle(spam)

>>> print(spam)

[9, 8, 3, 5, 4, 7, 1, 2, 0, 6]

You want the secret number in Bagels to have unique digits. The Bagels game is much more fun if you don’t have duplicate digits in the secret number, such as '244' or '333'. The shuffle() function will help you do this.

Getting the Secret Number from the Shuffled Digits

6. secretNum = ''

7. for i in range(numDigits):

8. secretNum += str(numbers[i])

9. return secretNum

The secret number will be a string of the first numDigits digits of the shuffled list of integers. For example, if the shuffled list in numbers is [9, 8, 3, 5, 4, 7, 1, 2, 0, 6] and numDigits was 3, then you’d want the string returned by getSecretNum() to be '983'.

To do this, the secretNum variable starts out as a blank string. The for loop on line 7 iterates numDigits number of times. On each iteration through the loop, the integer at index i is pulled from the shuffled list, converted to a string, and concatenated to the end of secretNum.

For example, if numbers refers to the list [9, 8, 3, 5, 4, 7, 1, 2, 0, 6], then on the first iteration, numbers[0] (that is, 9) will be passed to str(), which in turn returns '9' which is concatenated to the end of secretNum. On the second iteration, the same happens with numbers[1] (that is, 8) and on the third iteration the same happens with numbers[2] (that is, 3). The final value of secretNum that is returned is '983'.

Notice that secretNum in this function contains a string, not an integer. This may seem odd, but remember that you cannot concatenate integers. The expression 9 + 8 + 3 evaluates to 20, but what you want is '9' + '8' + '3', which evaluates to '983'.

## Augmented Assignment Operators

The += operator on line 8 is one of the augmented assignment operators. Normally, if you wanted to add or concatenate a value to a variable, you would use code that looked like this:

spam = 42

spam = spam + 10

eggs = 'Hello '

eggs = eggs + 'world!'

The augmented assignment operators are a shortcut that frees you from retyping the variable name. The following code does the same thing as the above code:

spam = 42

spam += 10 # Like spam = spam + 10

eggs = 'Hello '

eggs += 'world!' # Like eggs = eggs + 'world!'

There are other augmented assignment operators as well. Try entering the following into the interactive shell:

>>> spam = 42

>>> spam -= 2

>>> spam

40

>>> spam \*= 3

>>> spam

120

>>> spam /= 10

>>> spam

12.0

Calculating the Clues to Give

11. def getClues(guess, secretNum):

12. # Returns a string with the pico, fermi, bagels clues to the user.

13. if guess == secretNum:

14. return 'You got it!'

The getClues() function will return a string with the fermi, pico, and bagels clues depending on the guess and secretNum parameters. The most obvious and easiest step is to check if the guess is the same as the secret number. In that case, line 14 returns 'You got it!'.

16. clue = []

17.

18. for i in range(len(guess)):

19. if guess[i] == secretNum[i]:

20. clue.append('Fermi')

21. elif guess[i] in secretNum:

22. clue.append('Pico')

If the guess isn’t the same as the secret number, the code must figure out what clues to give the player. The list in clue will start empty and have 'Fermi' and 'Pico' strings added as needed.

Do this by looping through each possible index in guess and secretNum. The strings in both variables will be the same length, so the line 18 could have used either len(guess) or len(secretNum) and work the same. As the value of i changes from 0 to 1 to 2, and so on, line 19 checks if the first, second, third, etc. letter of guess is the same as the number in the same index of secretNum. If so, line 20 will add a string 'Fermi' to clue.

Otherwise, line 21 will check if the number at the ith position in guess exists anywhere in secretNum. If so, you know that the number is somewhere in the secret number but not in the same position. Line 22 will then add 'Pico' to clue.

23. if len(clue) == 0:

24. return 'Bagels'

If the clue list is empty after the loop, then you know that there are no correct digits at all in guess. In this case, line 24 returns the string 'Bagels' as the only clue.

## The sort() List Method

26. clue.sort()

Lists have a method named sort() that rearranges the items in the list to be in alphabetical or numerical order. Try entering the following into the interactive shell:

>>> spam = ['cat', 'dog', 'bat', 'anteater']

>>> spam.sort()

>>> spam

['anteater', 'bat', 'cat', 'dog']

>>> spam = [9, 8, 3, 5, 4, 7, 1, 2, 0, 6]

>>> spam.sort()

>>> spam

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

The sort() method doesn’t return a sorted list, but rather sorts the list it is called on “in place”. This is just like how the reverse() method works.

You would never want to use this line of code: return spam.sort() because that would return the value None (which is what sort() returns). Instead you would want a separate line spam.sort() and then the line return spam.

The reason you want to sort the clue list is to get rid of extra information based on the order of the clues. If clue was ['Pico', 'Fermi', 'Pico'], then that would tell the player that the center digit of the guess is in the correct position. Since the other two clues are both Pico, the player would know that all they have to do is swap the first and third digit to get the secret number.

If the clues are always sorted in alphabetical order, the player can’t be sure which number the Fermi clue refers. This is what we want for the game.

## The join() String Method

27. return ' '.join(clue)

The join() string method returns a list of strings as a single string joined together. The string that the method is called on (on line 27, this is a single space, ' ') appears between each string in the list. For an example, enter the following into the interactive shell:

>>> ' '.join(['My', 'name', 'is', 'Zophie'])

'My name is Zophie'

>>> ', '.join(['Life', 'the Universe', 'and Everything'])

'Life, the Universe, and Everything'

So the string that is returned on line 27 is each string in clue combined together with a single space between each string. The join() string method is sort of like the opposite of the split() string method. While split() returns a list from a split up string, join() returns a string from a combined list.

Checking if a String Has Only Numbers

29. def isOnlyDigits(num):

30. # Returns True if num is a string made up only of digits. Otherwise returns False.

31. if num == '':

32. return False

The isOnlyDigits() helps determine if the player entered a valid guess. Line 31 checks if num is the blank string, and if so, returns False.

34. for i in num:

35. if i not in '0 1 2 3 4 5 6 7 8 9'.split():

36. return False

37.

38. return True

The for loop iterates over each character in the string num. The value of i will have a single character on each iteration. Inside the for-block, the code checks if i doesn’t exist in the list returned by '0 1 2 3 4 5 6 7 8 9'.split(). (The return value from split() is equivalent to ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'] but is easier to type.) If it doesn’t, you know there’s a non-digit character in num. In that case, line 36 returns False.

If execution continues past the for loop, then you know that every character in num is a digit. In that case, line 38 returns True.

Finding out if the Player Wants to Play Again

40. def playAgain():

41. # This function returns True if the player wants to play again, otherwise it returns False.

42. print('Do you want to play again? (yes or no)')

43. return input().lower().startswith('y')

The playAgain() function is the same one you used in Hangman and Tic Tac Toe. The long expression on line 43 evaluates to either True or False based on the answer given by the player.

The Start of the Game

45. NUMDIGITS = 3

46. MAXGUESS = 10

47.

48. print('I am thinking of a %s-digit number. Try to guess what it is.' % (NUMDIGITS))

49. print('Here are some clues:')

50. print('When I say: That means:')

51. print(' Pico One digit is correct but in the wrong position.')

52. print(' Fermi One digit is correct and in the right position.')

53. print(' Bagels No digit is correct.')

After all of the function definitions, this is the actual start of the program. Instead of using the integer 3 in our program for the number of answer has, use the constant variable NUMDIGITS. The same goes for using the constant variable MAXGUESS instead of the integer 10 for the number of guesses the player gets. Now it will be easy to change the number of guesses or secret number digits. Just change line 45 or 46 and the rest of the program will still work without any more changes.

The print() function calls will tell the player the rules of the game and what the Pico, Fermi, and Bagels clues mean. Line 48's print() call has % (NUMDIGITS) added to the end and %s inside the string. This is a technique known as string interpolation.

## String Interpolation

String interpolation is a coding shortcut. Normally, if you want to use the string values inside variables in another string, you have to use the + concatenation operator:

>>> name = 'Alice'

>>> event = 'party'

>>> where = 'the pool'

>>> day = 'Saturday'

>>> time = '6:00pm'

>>> print('Hello, ' + name + '. Will you go to the ' + event + ' at ' + where + ' this ' + day + ' at ' + time + '?')

Hello, Alice. Will you go to the party at the pool this Saturday at 6:00pm?

As you can see, it can be hard to type a line that concatenates several strings. Instead, you can use string interpolation, which lets you put placeholders like %s. These placeholders are called conversion specifiers. Then put all the variable names at the end. Each %s is replaced with a variable at the end of the line. For example, the following code does the same thing as the previous code:

>>> name = 'Alice'

>>> event = 'party'

>>> where = 'the pool'

>>> day = 'Saturday'

>>> time = '6:00pm'

>>> print('Hello, %s. Will you go to the %s at %s this %s at %s?' % (name, event, where, day, time))

Hello, Alice. Will you go to the party at the pool this Saturday at 6:00pm?

String interpolation can make your code much easier to type. The first variable name is used for the first %s, the second variable with the second %s and so on. You must have the same number of %s conversion specifiers as you have variables.

Another benefit of using string interpolation instead of string concatenation is interpolation works with any data type, not just strings. All values are automatically converted to the string data type. If you concatenated an integer to a string, you’d get this error:

>>> spam = 42

>>> print('Spam == ' + spam)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: Can't convert 'int' object to str implicitly

String concatenation can only combine two strings, but spam is an integer. You would have to remember to put str(spam) instead of spam. But with string interpolation, this conversion to strings is done for you. Try entering this into the interactive shell:

>>> spam = 42

>>> print('Spam is %s' % (spam))

Spam is 42

String interpolation is also known as string formatting.

Creating the Secret Number

55. while True:

56. secretNum = getSecretNum(NUMDIGITS)

57. print('I have thought up a number. You have %s guesses to get it.' % (MAXGUESS))

58.

59. numGuesses = 1

60. while numGuesses <= MAXGUESS:

Line 55 is an infinite while loop that has a condition of True so it will loop forever until a break statement is executed. Inside the infinite loop, you get a secret number from the getSecretNum() function, passing it NUMDIGITS to tell how many digits you want the secret number to have. This secret number is assigned to secretNum. Remember, the value in secretNum is a string not an integer.

Line 57 tells the player how many digits is in the secret number by using string interpolation instead of string concatenation. Line 59 sets variable numGuesses to 1 to mark this is as the first guess. Then line 60 has a new while loop that loops as long as numGuesses is less than or equal to MAXGUESS.

Getting the Player’s Guess

61. guess = ''

62. while len(guess) != NUMDIGITS or not isOnlyDigits(guess):

63. print('Guess #%s: ' % (numGuesses))

64. guess = input()

The guess variable will hold the player’s guess returned from input(). The code keeps looping and asking the player for a guess until the player enters a valid guess. A valid guess has only digits and the same number of digits as the secret number. This is what the while loop that starts on line 62 is for.

The guess variable is set to the blank string on line 61 so the while loop’s condition is False the first time it is checked, ensuring the execution enters the loop.

Getting the Clues for the Player’s Guess

66. clue = getClues(guess, secretNum)

67. print(clue)

68. numGuesses += 1

After execution gets past the while loop that started on line 62, guess contains a valid guess. Pass this and secretNum to the getClues() function. It returns a string of the clues, which are displayed to the player on line 67. Line 68 increments numGuesses using the augmented assignment operator for addition.

Checking if the Player Won or Lost

Notice that this second while loop on line 60 is inside another while loop that started on line 55. These loops-inside-loops are called nested loops. Any break or continue statements will only break or continue out of the innermost loop, and not any of the outer loop.

70. if guess == secretNum:

71. break

72. if numGuesses > MAXGUESS:

73. print('You ran out of guesses. The answer was %s.' % (secretNum))

If guess is the same value as secretNum, the player has correctly guessed the secret number and line 71 breaks out of the while loop that was started on line 60.

If not, then execution continues to line 72, where it checks if the player ran out of guesses. If so, the program tells the player they’ve lost.

At this point, execution jumps back to the while loop on line 60 where it lets the player have another guess. If the player ran out of guesses (or it broke out of the loop with the break statement on line 71), then execution would proceed past the loop and to line 75.

Asking the Player to Play Again

75. if not playAgain():

76. break

Line 75 asks the player if they want to play again by calling the playAgain() function. If playAgain() returns False, break out of the while loop that started on line 55. Since there’s no more code after this loop, the program terminates.

If playAgain() returned True, then the execution would not execute the break statement and execution would jump back to line 55. The program generates a new secret number so the player can play a new game.

Summary

Bagels is a simple game to program but can be difficult to win at. But if you keep playing, you’ll eventually discover better ways to guess and make use of the clues the game gives you. This is much like how you’ll get better at programming you more you keep at it.

This chapter introduced a few new functions and methods (random.shuffle(), sort(), and join()), along with a couple handy shortcuts. An augmented assignment operators involve less typing when you want to change a variable’s relative value such as in spam = spam + 1, which can be shortened to spam += 1. String interpolation can make your code much more readable by placing %s (called a conversion specifier) inside the string instead of using many string concatenation operations.

The next chapter isn’t about programming directly, but will be necessary for the games we want to create in the later chapters of this book. We will learn about the math concepts of Cartesian coordinates and negative numbers. These are used in the Sonar, Reversi, and Dodger games, but Cartesian coordinates and negative numbers are used in many games. If you already know about these concepts, give the next chapter a brief read anyway to refresh yourself.