PLTW COMPUTER SCIENCE

Activity 1.3.8

While Loops

Introduction

A `while` loop is another way to iterate code, as you discovered while using the Scratch™ programming language. Unlike `for` loops, which are used to iterate across a collection of a known length, a `while` loop is controlled by a conditional expression that might be less predictable. The conditional expression is evaluated once before an iteration through the `while` block and again before each additional iteration. If the conditional expression is `False` when evaluated (always at the beginning/end of an iteration), execution jumps to the code after the `while` block.

Materials

- Computer with Enthought Canopy distribution of *Python*® programming language

Resources

1.3.8 sourceFiles.zip

Procedure

1. Form pairs as directed by your teacher. Meet or greet each other to practice professional skills and establish norms.

2. Launch Canopy and open an editor window.

3. If your teacher directs you to turn in your work with an IPython log, set the working directory for the IPython session and turn on session logging.

```
In []: %logstart -ort JDoeBSmith1_3_8.log
In []: # Jane Doe 1.3.8 IPython log
```

4. Start a new program in the code editor by choosing **File > New > Python file**. Save the file as

```
JDoe_BSmith_1_3_8.py
```

5. Any Boolean operator, like `==`, `!=`, or `<=`, can be used in the conditional expression for a `while` loop. One useful Boolean expression for validating input is the `in` operator. Validating input means checking that the input is what you expected. Try the following code:

```
 def validate():
     guess = '1' # initialization with a bad guess
     answer = 'hangman word'
     while guess not in answer:
         guess = raw_input('Name a letter in \'' + answer + '\': ')
     print('Thank you!')
```

   Why is line 2 necessary? Try commenting it out or changing `'1'` to `'a'`.

6. The following code is poorly annotated. Read and discuss it with your partner. Replace the comments on lines 5-8, 13, 16, 18, and 21 with your own. Then try it and modify your comments if needed.

```
from __future__ import print_function
import random

def guess_winner(players=('Amy', 'Bill', 'Cathy', 'Dale')):
    '''Summarize the function in this docstring.

    Provide descriptions for the arguments and say what type each one
    Describe the type and meaning of the value returned.
    '''
    winner = random.choice(players)

    ####
    # Summarize the following section of code here
    ####
    print('Guess which of these people won the lottery: ',end='')
    for p in players[:len(players)-1]: # explain index here
        print(p+', ', end='')
    print(players[len(players)-1]) # explain this line here

    ####
    # Summarize the following section of code here
    ####
    guesses = 1
    while raw_input() != winner:
        print('Guess again!')
        guesses += 1
    print('You guessed in', guesses, 'guesses!')
    return guesses
```

7. It is often useful to consider how many times a `while` loop will execute. If `guess_game(options)` is called with a list of 100 options, how many times might the user have to guess?

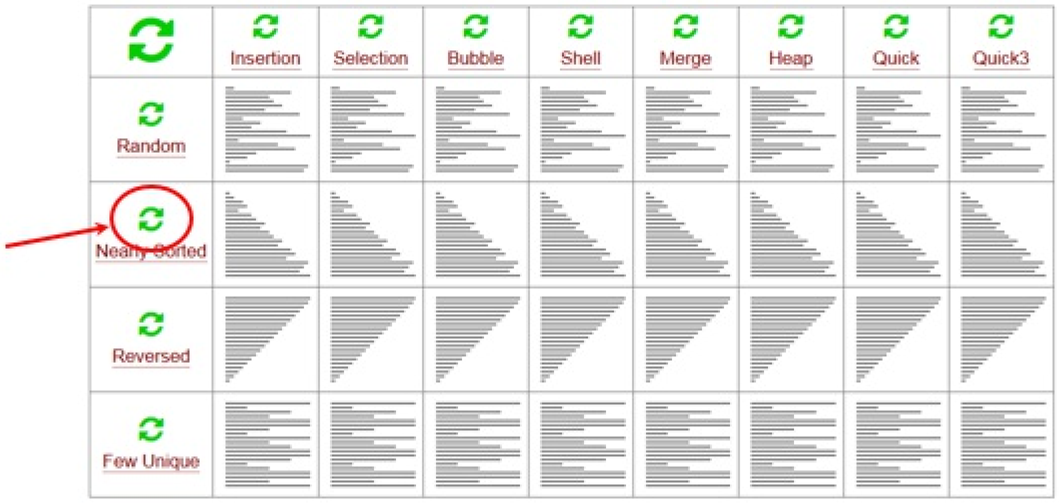
8. Define a function `goguess()` that implements a number guessing game as described below.

Strategize as pair programmers. Then iteratively code and test.

```
In []: goguess()
I have a number between 1 and 20 inclusive.
Guess: 6
6 is too low.
Guess: 10
10 is too high.
Guess: 9
Right! My number is 9! You guessed in 3 guesses!
```

9. If you change `between 1 and 20` from the previous program to `between 1 and 6000`, how many guesses will you need to guarantee that you have the right answer? Explain.

10. Sorting and searching are two important algorithmic problems. You can compare different sorting algorithms (insertion sort, selection sort, bubble sort, etc.) at http://www.sorting-algorithms.com/ . Which algorithm is fastest when a list is nearly sorted? Compare the algorithms empirically as shown below.



11. The most important solutions to the problem of searching are linear search (checking one at a time, potentially going through a whole list) and binary search (checking against the middle term in a list to narrow the search down to half the items with each comparison.)

   - Pair up with someone whose birthday you do not know. Find out their birthday using binary search, asking whether their birthday is earlier or later in the year than each of your guesses. How many guesses might be required to guarantee you find out their birthday?

   - Sorting a list helps you search the list. Explain why an unsorted list cannot be searched with binary search.

   - Sorting consumes a lot of processor cycles. It's not worth it if you're only going to search a list once, but definitely worth the time to sort if you're going to search the list a lot. Explain how you know whether Google keeps its inventory of webpage content sorted.

Conclusion

1. Describe the difference between a `while` loop and a `for` loop.
2. Reflect on how effectively you and your partner worked together. Individually, describe the strengths of your process and style, and describe what could be improved and how.