Developers Institute

Python Course

Week 1

Day 2

Exercises

Exercise 1 – Sequences

1. Create a list containing the following sequences. Don’t hard-code the sequences.
   1. 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
   2. 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.
   3. -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4.
   4. 10, 9, 8, 7, 6, 5
   5. 1, 3, 5, 7, 9, 11, 13
   6. (Bonus: 2, 4, 8, 16, 32, 64, 128)
   7. (Is this possible? 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5)
2. Choose a motivational message (like ‘Keep Calm and Carry On’). Print it 3 times – use a loop.
3. Extend the motivational message to begin with the number of times it has been printed, eg. ‘1. Keep Calm and Carry On’, ‘2. Keep Calm and Carry On’, etc.

Exercise 2 – Sequences of floating-point numbers

1. Recap – What is a float? What is the difference between an integer and a float?
2. Earlier, we tried to create a sequence of floats. Did it work?
3. Can you think of another way of generating a sequence of floats?
4. Create a list containing the sequence 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5 **without hard-coding** the sequence.

Exercise 3 – Math on a List

We are given a list of 10 integers to analyze.
Repeat the questions below with the following lists of numbers:

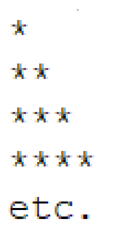
- [3, 47, 99, -80, 22, 97, 54, -23, 5, 7]
- [44, 9l, 8, 24, -6, 0, 56, 8, 100, 2]
- [3, 21, 76, 53, 9, -82, -3, 49, 1, 76]
- [18, 19, 2, 56, 33, 17, 41, -63, -82, 1]

1. Store the list of numbers in a variable.
2. We will now print some information about the list of numbers. Each time, print the answer together with a **helpful message** that tells the user what you are printing.
3. Print the following information:
   1. The list of numbers – printed in a single line
   2. The list of numbers – sorted in descending order (largest to smallest)
   3. The sum of all the numbers

4. A list containing the first and the last numbers only
5. A list of all the numbers greater than 50
6. A list of all the numbers smaller than 10
7. A list of all of the numbers **squared** – eg. for [1, 2, 3] you would print “1 4 9”
8. The numbers without any **duplicates** – also print how many numbers are in the new list.
9. (Bonus: The **average** of all the numbers)
10. (Bonus: The **largest** number)
11. (Bonus: The **smallest** number)
12. (Extra bonus: instead of using pre-defined lists of numbers, ask the user for 10 numbers between -100 and 100.
    1. Ask the user for an integer between -100 and 100 – repeat this question 10 times. Each answer from the user should be added onto a list variable that you created earlier.
    2. After asking the user 10 times, you should now have a list of integers.
    3. Print a line to separate the input section (getting the numbers from the user) from the output section).

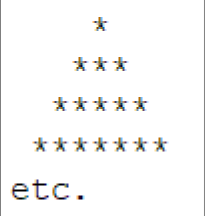
Exercise 3 – Stars

1. Today we learned how to repeat code multiple times using a loop. Making use of **loops**, print the following shapes:
    1. A triangle of stars 10 lines tall, with the base at the bottom (see Fig. 1)
    2. A triangle of stars 10 lines tall, with the base at the top.
    3. A ‘regular’ triangle, 10 lines tall (see Fig. 2).
    4. The mirror-image of 3.1.1 above – instead of ‘leaning’ against the left-hand side, the triangle should ‘lean’ against the right-hand side.
    5. (Bonus: Two triangles of stars 10 lines tall – facing each other.)
    6. (Bonus: A diamond of stars, 21 lines tall.)



```
*
**
***
****
etc.
```

Fig. 1: Triangle with base at bottom



```
   *
  ***
 *****
*******
etc.
```

Fig. 2: 'Regular' triangle

Exercise 4 – The Guessing Game

1. *Basics*
    1. Create a ‘secret’ number – a random integer between 1 and 100. Store it in a variable with an appropriate name.
    2. Ask the user to guess the number.
    3. If the user gets the correct answer, print a message of congratulations
    4. If the user gets the wrong answer, print an insult (or just a message saying that s/he got it wrong…) Tell the user what the secret number was before the program exits.
2. *Improvements*
    1. Let’s be fair – the chances that the user will get the correct answer are low. So let’s allow the user to have a few tries to guess the secret number.
    2. Using loops, keep asking the user to guess the secret number until they guess right. (Remember not to tell the user what the number is!)
    3. To help the user along, tell them if the secret number is **higher** or **lower** than the number they guessed.
3. **Test your code**: Does your code actually do what it’s supposed to do? It’s hard to tell when you’re generating random numbers, because you don’t actually know what the number is! Try the following two different ways of testing your code:

    1. **Hard-code** the secret number instead of generating it at random (just for your test). Run your program, and check that your program runs as it should for different inputs (correct answer, incorrect answer...). Now change it back to a random number.
    2. **Debug** by printing out the secret number as soon as it’s been generated. Now you can examine the program’s output and easily tell if the program has been working as it should.
4. (Bonus: choose how many tries the user should be allowed to have. Hard-code this as a variable at the top of your script. Each time you prompt the user for an answer, also inform them how many tries they have left to guess the correct number. If the user doesn’t guess the correct number after that many tries, stop asking. Simply tell them that they have lost, and tell them what the secret number was before exiting.)