CS 112, Lab 5 – Exercise – Loops

Due: Sat. 06/19, 11:59pm

Files:

- **download** tester file on the blackboard for testing.
- no template provided – create your file from scratch and include the functions defined below
- Grading is based on the fraction of passing tests. You can run it like this:
  `python3 testerL5E.py gmason76_230_L5.py`

As this is an **Exercise**, you can read any and all materials, ask us questions, talk with other students, and learn however you best learn in order to solve the task. Just create your own solution from those experiences, and turn in your work.

Loops allow us to run the same block of code multiple times in a row. When variables used in that block of code are changed between each iteration, we can accumulate effects and process larger batches of data, such as the elements in a list. There are problems that selection statements alone are insufficient to solve, yet loops add enough power to tackle them. We will use loops to solve various problems.

Notes

- We will fill out function definitions again. Remember, don't call `input()` or `print()` anywhere.
- A few of the tasks have individual restrictions about what may be called; note them, as we will remove points for any task that uses them.

Turning It In

Add a comment at the top of the file that indicates your name, userID, G#, lab section, a description of your collaboration partners, as well as any other details you feel like sharing. Please also mention what was most helpful for you. Once you are done, run the testing script once more to make sure you didn't break things while adding these comments. If all is well, go ahead and turn in just your one .py file you've been working on, named with our usual convention (see above), over on BlackBoard to the correct lab assignment. We have our own copy of the testing file that we'll use, so please don't turn that in (or any other extra files), as it will just slow us down.

Tasks

- `def location(xs, key):` We will implement our own function-version of the `.index()` method (which you may not call). Search through the list `xs`, and when you find a value equivalent to `key`, return that non-negative index. When it's not found, return the `None` value. The leftmost match (smallest non-negative index) is reported when multiple matches of the key are present.
  - assume that `xs` is a list.
  - Restriction: can't call `.index()` method – just loop through manually and seek the key
  - Examples:
    - `location( [5,10,15,20,25],    15)` → `2`
    - `location( [3,4,5,6,4,4,4,4,4], 4)` → `1`
    - `location( [100, 105, 110],   234)` → `None`

- `def fibonacci(n):` The fibonacci sequence begins with two `1`'s, and each successive spot is the sum of the previous two. If we were to store those values in a list, it would begin as:
  `fibs = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, … ]`
  Given an index into that list, `n`, report the `n`th fibonacci number.
  - Note: though there is a well-known recursive solution, it is well-known because it is pathologically slow! We have one larger test-case that will somewhat require that you use a loop-based solution. Restriction: no recursion. If you don't know what recursion is, I doubt you'll stumble into it! (don't call the fibonacci function on other inputs as your solution).
  - Examples:
    - `fibonacci(0)` → `1`
    - `fibonacci(7)` → `21`
    - `fibonacci(15)` → `987`

- `def int_sqrt(n):` Find the largest integer whose square isn't bigger than the parameter `n`.
  - assume that `n` is a positive integer.
  - Restriction: you may not import or use `math.sqrt()`
  - Examples:
    - `int_sqrt(9)` → `3` `# 3*3 happens to perfectly equal 9.`
    - `int_sqrt(25)` → `5` `# 5*5 is the largest square not larger than 25`
    - `int_sqrt(26)` → `5` `# n:25 --> 35 all answer with 5.`
    - `int_sqrt(36)` → `6` `# 6*6 happens to perfectly equal 36.`

- `def sum_evens_2d(xss):` Given a "two-dimensional list" of integers – that is, a list where each value in it is a list of integers – find all the even numbers and sum them together. You will need more than a single loop for this!
  - assume that `xss` is a list where each value is a list of integers.
  - Restriction: you may not call `sum()`. (It would have included the odds anyways).
  - Examples:
    - `sum_evens_2d([[1,2,3],[4,5,6]])` → `2+4+6 == 12`
    - `sum_evens_2d([[1,3,5],[7,9,11]])` → `0`
    - `sum_evens_2d([[1,2,3],[4,5],[6],[7,8,9]])` → `2+4+6+8 == 20`