T3: Iterators, Generators, and Swampy

This is a team assignment designed as an in-class activity. It should be completed in your design team.

**Directions for use:**

* To use this form effectively, sign into a Google account.
* Then under “File” choose “Make a Copy” in order to be able to edit.
* Share with all team members, but allow the Recorder to do the recording.
* Each yellow box should be filled with an appropriate team response.
* You are also going to write a Python program to verify some of the computations.
* Once complete, download as *yourteamname-T3.docx*.
* Upload thefollowing to Moodle:
  + *yourteamname-*T3.docx

**Member Roles**

* If you have only three people, also combine Recorder & Spokesperson
* If you have five people, add a Process Analyst
* Make up a team name which suits your team!

|  |  |
| --- | --- |
| **Team Roles** | **Member Name** |
| **Facilitator:** Reads the questions aloud, keeps track of time, and makes sure everyone contributes appropriately. | **Ishwar Agarwal** |
| **Recorder:** Records all answers and questions and uploads when complete. | **John Hellrung** |
| **Spokesperson:** Talks to the instructor(s), TA(s) and other teams. Compiles and runs programs when applicable. | **Ben Quesada** |
| **Quality Control Officer:** Considers how the answers could be deeper, and how the team could work and learn more effectively. | **Haleigh George** |
| **Process Analyst** | **William Mosier** |
| **Team Name:**  Make a name which is representative of your group (or just fun.) | **Team Awesome Lemmings Team** |

**Iteration**

According to Wikipedia, “**iteration** is the act of repeating a process with the aim of approaching a desired goal, target or result. Each repetition of the process is also called an ‘iteration’, and the results of one iteration are used as the starting point for the next iteration.” That sounds a bit like looping, but iteration is an abstraction while looping is an implementation. Iteration can be implemented via looping or via recursion.

One of the reasons why Python is so highly regarded as a rapid development language is because of how it handles its’ variables and compound data structures or containers. You might think of a container as an abstraction of a real life container: a bag, a box, a cookie jar, a house, etc. More technically, an object is a **container** when one can ask whether or not it contains a certain element. Examples of useful container types include strings, lists, tuples, and dictionaries.

Unlike many earlier languages, Python uses iterators in a truly innate fashion. In Python, one can easily iterate over a container using a loop.

|  |
| --- |
| **Consider each of the following loops** Beginning with the code: [loop-iterators.py](http://cs.berea.edu/courses/complexity/tasks/loop-iterators.py) |
| Write the output of the following:  list = [10, 20, 30, 40]  for num in list:     print(num) |
| 10 20 30 40 |
| Write the output of the following:  for letter in "string**"**:     print(letter) |
| “s”  “t”  “r”  “i”  “n”  “g” |
| Write the output of the following: (Be very careful here! Don’t just guess!)  month= {"Jan":1, "Feb" :2, "Mar":3, "Apr":4, "May":5, "Jun":6, "Jul":7,  "Aug":8, "Sep":9, "Oct":10,  "Nov":11,  "Dec":12}  for when in month:     print(when) |
| Feb  Aug  Jan  Dec  Oct  Mar  Sep  May  Jun  Jul  Apr  Nov |
| Was your group surprised by the output of any of these? Explain. |
| We were surprised by the dictionary output since we expected them to be printed in order but they were not. |

In each of these examples, an iterator is created internally. An **iterator** is an object that produces the next value when you call next() on it. Any object that has a \_\_next\_\_() method is therefore an iterator.

**The Iteration Protocol**

The built-in Python function iter takes an iterable object and returns an iterator. For example, assuming the following are executed in order:

|  |
| --- |
| **Assume the following are executed in consecutive order** Begin with the code:  [iterate.py](http://cs.berea.edu/courses/complexity/tasks/iterate.py) |
| Write the ***output*** of the following: (Hint: Be careful!)  x = iter(range(10)) |
| No output, because it is an expression not a statement. |
| Assuming the above statement was executed, write the ***output*** of the following:  print(x.next()) |
| 0 |
| Assuming the above statements were both executed, write the ***output*** of the following:  print(x.next()) |
| 1 |
| Assuming the above statements were all executed, write the ***output*** of the following:  print(x.next()) |
| 2 |

In each of these examples, an iterator is created internally. An **iterator** is an object that produces the next value when you call next() on it. Any object that has a \_\_next\_\_() method is therefore an iterator. Internally, iterators are implemented as classes in Python.

**Generators**

A generator is a function that produces a sequence of results instead of a single value, so generators simplify the creation of an iterator.

|  |
| --- |
| **Assume the following are executed in consecutive order** Begin with the code [generate.py](http://cs.berea.edu/courses/complexity/tasks/generate.py) |
| Write the ***output*** of the following: (Hint: Be careful!)  for letter in 'abcdefghijklmnopqrstuvwxyz'     yield letter |
| Nothing |
| Assuming the above statement was executed, write the ***output*** of the following:  iter = generate\_letters()  print(iter.next()) |
| a |
| Assuming the above statements were both executed, write the ***output*** of the following:  print(iter.next()) |
| b |
| Assuming the above statements were all executed, write the ***output*** of the following:  print(iter.next()) |
| c |

**Swampy**

Swampy is a suite of Python programs that support our textbook, [Think Complexity](http://www.greenteapress.com/complexity/) Follow the directions at <http://www.greenteapress.com/thinkpython/swampy/install.html> to install the Swampy library, being sure to download the suite which works in Python 2 rather than Python 3. Then download and run the following:

* [Graphy.py](http://cs.berea.edu/courses/complexity/tasks/Graph.py)
* [GraphWorld.py](http://cs.berea.edu/courses/complexity/tasks/GraphWorld.py)

|  |
| --- |
| After running the above code several times, in main() of  GraphWorld.py, change the number of nodes which are generated to take input from the user, and run again  Write the line or lines of code used to accomplish this: |
| **def main**(script, n='5', \*args):    # create n Vertices    n = int(n)    labels = string.ascii\_lowercase + string.ascii\_uppercase    vs = [Vertex(c) **for** c **in** labels[:n]] |

**Reaction**

|  |
| --- |
| Discuss the things you learned in this teamwork, and identify the most important thing you learned and why you think it is the most important. |
| In this teamwork we learned how iterators and generators work. We learned how an object containing \_\_next\_\_ method can produce next values during iteration. |

**Submission**

To submit:

* The recorder will download this Google Doc worksheet as:
  + *yourteamname-T3.docx* and upload to Moodle.
* All other team members should submit the name of your team and all team mates.