Learning Objectives
Students will be able to:

Content:
- Define a generator
- Explain the difference between the `yield` and `return` keywords

Process:
- Write code that creates a generator.
- Write code that uses a generator via `next(..)` and a `for..loop`.

Prior Knowledge
- Python concepts from Activities 1-20.

Folks, this is a brand new activity. If you encounter any issues/typos, please let Iris know!

Critical Thinking Questions:

1. Examine the sample code from interactive python, below.

   ```python
   0 >>> def countEvens(n):
   1     i = 0
   2     while i <= n:
   3         print(i)
   4     i += 2
   5 >>> countEvens(3)
   ```

   a. When does the `while` loop on line 2 stop?
      _______________________
   
   b. If the parameter `n` was 3, how many times through the loop would we go? _________
      _______________________
   
   c. What is the output from calling `countEvens`, on line 5?
      _______________________

   FYI: A generator is an object that constructs a (possibly infinite) stream of values on demand.
2. Examine the sample code from interactive python, below.

```python
0  >>> def countEvens(n):
1     ... i = 0
2     ... while i <= n:
3         ... yield i
4     ... i += 2
5  >>> g = countEvens(10)
6  >>> next(g) 0
7  >>> next(g) 2
```

a. How does the function `countEvens(n)` differ from the previous `countEvens(n)`?

b. Write a line of code to print the next value yielded by `g`.

c. If we replace line 5 with `g = countEvens(10)`, what will the first five calls of `next(g)` generate? ______ ______ ______ ______ ______

d. Write a new function, `reverseGen(..)`, that takes a list and yields values from the list from the end to the beginning:

```python
def reverseGen(mylist):
    ...
    ...
    ...
    ...
    ...
```

FYI: `Yield` is a keyword like return, but instead of returning a value, it surrenders a generator object. The special method `next(..)` is required to retrieve the value that was yielded.

3. The following code occurs in interactive Python:

```python
0  >>> def countEvens(n):
1     ... i = 0
2     ... while i <= n:
3         ... yield i
4     ... i += 2
5  >>> for num in countEvens(3):
6     ... print(num)
```
a. The output from this sample code is the same as the output from Question 1. What might the `for` loop be doing in order to make this possible?

b. What will this code output?

c. Write a couple lines of code to use your `reverseGen(..)` generator from the previous question, using a `for` loop:

FYI: A more efficient mechanism for using generators is by using a for loop.

4. Examine the following Python code:

```python
0 def count(start = 0, step = 1):
1   i = start
2   while True:
3     yield i
4     i += step
```

a. How do the parameters of this `count(..)` function differ from those of `countEvens(..)`?

b. If we wanted to replicate the behavior of `countEvens(..)` with the `count(..)` function, what would our `start` and `step` values be?

```python
0 >>> def letters(word, n):
1   ...     i = 0
2   ...     while i < n:
3   ...         yield word[i]
4   ...     i += 1
5 >>> g=letters(‘good’, 3)
6 >>> next(g)
7 ‘g’
8 >>> next(g)
9 ‘o’
10 >>> next(g)
11 ‘o’
12 >>> next(g)
```

da. Write a few lines of code to output the first four multiples of the number three using `count(..)`:

```
```

c. When does the while loop on line 2 end?

5. Examine the following code from interactive python:

```
```
b. What are the values of the arguments passed to `letters(...)` on line 5? _______ 

c. What does the calls to `next(g)` do on lines 7, 9, and 11?  
_____________________________________________________________________________ 

d. Why might an error have been thrown by the `next(g)` call on line 12?  
_____________________________________________________________________________ 

e. What would happen if we replaced line 5 with `g=letters('good',4)`?  
_____________________________________________________________________________ 

f. What might happen if we replaced line 5 with `g=letters('bye',4)`?  
_____________________________________________________________________________ 

6. Examine the following Python code: 

```python
def mystery(a = 0, b = 1):
    yield a
    yield b
    while True:
        a, b = b, a+b
        yield b
```

a. Use the following table to step-through what this function is doing:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Yield Statement</th>
<th>Yielded</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>yield a</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>yield b</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>yield b (2)</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>yield b (2)</td>
<td></td>
</tr>
</tbody>
</table>

b. If we were to rename this function to something more meaningful, what would we name it to?  
_____________________________________________________________________________ 

Application Questions: Use the Python Interpreter to check your work

1. Write a function that uses the yield statement to infinitely generate all the odd numbers:

```python
def oddNum():
    ... 
```