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.

<table>
<thead>
<tr>
<th>Sample Code</th>
</tr>
</thead>
</table>
| 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.

<table>
<thead>
<tr>
<th>Sample Code</th>
</tr>
</thead>
</table>
| 0  >>> def countEvens(n):
1    ...   i = 0
2    ...   while i <= n:
3    ...       yield i
4    ...       i += 2
5  >>> g = countEvens(3)
6  >>> print(next(g))
7    0
8  >>> print(next(g))
9    2
10  >>> print(next(g)) |

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

b. If we replace line 5 with g = countEvens(6), what will the output of line 10 be?

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

d. With line 5 as g = countEvens(3), what might the output of line 10 be?

e. With g = countEvens(3), line 10 will produce a StopIteration exception. Why might that be?

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

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 code from interactive Python:

```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
start:________________________
step:_______________________
```

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

________________________

```
```

d. What is the output of the `count(..)` function?

______________ ___________ ___________ ___________ ___________
5. Examine the following code from interactive Python:

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

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

<table>
<thead>
<tr>
<th>a</th>
<th>b</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():

```

_____________________________________________________________________________
_____________________________________________________________________________
_____________________________________________________________________________
_____________________________________________________________________________
_____________________________________________________________________________
_____________________________________________________________________________

4. Review lab assignments and Homeworks for more applications of generators.