Learning Objectives
Students will be able to:

Content:
- Define a **keyword** and **positional parameters**
- Identify **variadic arguments**

Process:
- Write functions that use both positional and keyword parameters
- Write code that unpacks elements from variadic arguments.

Prior Knowledge
- Python concepts from Activities 1-27.

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 below. The code’s output is shown to the right.

```
Sample Code

0  >>> mys = "hello, there, world!"
1  >>> mys.split(',
2       ['hello', 'there', 'world']
3  >>> mys.split(',, maxsplit=1)
4       ['hello', ' there, world!']
```

   a. How many commas are in the string, mys? ____________________________

   b. How many commas in total are in the strings returned by the first `split(..)` function call on line 1? ____________________________

   c. What does the first parameter of the `split(..)` method represent? ____________________________

   d. How many commas in total are in the strings returned by the second `split(..)` function call on line 3? ____________________________

   e. What might the second parameter of the `split(..)` method represent? ____________________________

   f. What happens when no `maxsplit` parameter is passed to the `mys.split(..)` method? ____________________________

   g. What might the parameters of the `split(..)` function look like in the function header? ____________________________
2. Examine the sample function definition below. It takes in rankings from an athletic competition and prints them out, in order, as a list.

```python
def printRanks(frst, sec, thrd=None, frth=None):
    rank = []
    rank[1], rank[2] = frst, sec
    rank[3] = thrd if thrd is not None else 'Nobody'
    rank[4] = frth if frth is not None else 'Nobody'
    print(rank)
```

a. Which arguments of `printRanks(..)` are positional args? __________________________

b. Which arguments of `printRanks(..)` are keyword args? __________________________

c. How does the way first and second place are processed and placed into the rank list differ from the way third and fourth places are processed?

______________________________

d. If we called `printRanks(..)` with the following function call, what would we expect the output to be? `printRanks('Amy', 'Beth', thrd='Carol', frth='Danni')`

____________________________________________________________________

e. If we called `printRanks(..)` with the following function call, what would we expect the output to be? `printRanks('Amy', 'Beth')`

____________________________________________________________________

f. If we called `printRanks(..)` with the following function call, what would we expect the output to be? `printRanks('Amy', 'Beth', frth='Eliza')`

____________________________________________________________________

g. Write some lines of code to modify `printRanks(..)` so it can take a fifth-place ranking:

____________________________________________________________________
____________________________________________________________________
____________________________________________________________________

FYI: Functions in python can accept **positional arguments** or **keyword arguments**. Positional arguments are specified by their position in the function definition, while keyword arguments are specified by their identifier. Keyword arguments can have a default value specified in the function definition.

FYI: Keyword arguments can be omitted, but positional arguments cannot. Because of this, positional arguments must be placed before keyword arguments. Hence, a function definition such as `def printRanks(frst, sec=None, thrd, frth=None)` would generate a SyntaxError: non-default argument follows default argument.
3. Examine the sample code below. The code’s output is shown to the right.

<table>
<thead>
<tr>
<th>Sample Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>print(&quot;hello&quot;)</code>       hello</td>
</tr>
<tr>
<td><code>print(&quot;hello&quot;, 1, [0.1, 0.2])</code>       hello 1 [0.1, 0.2]</td>
</tr>
</tbody>
</table>

a. How many parameters does `print(..)` accept? ______________________

b. What might the function header of `print(..)` look like?

4. Examine the sample code below. This function replicates some of the behavior of `print(..)`.

```
0 def stringToPrint(*args):
1   s = ""
2   for item in args:
3       s += str(item) + " "
4   return s

5 >>> stringToPrint(('hello'))
6   'hello'
7 >>> stringToPrint(('hello', 1, [0.1, 0.2]))
8   'hello 1 [0.1, 0.2]'  
```

a. How many parameters did we pass `stringToPrint(..)` on line 5? ______________

b. How many parameters did we pass `stringToPrint(..)` on line 7? ______________

c. How many parameters does `stringToPrint(..)` have in its function header? _____

d. What does the `*args` parameter represent? _____________________________________________

e. Inside `stringToPrint(..)`, what might `args[0]` be? ____________________________________________

f. Rewrite `printRanks(..)` from the previous page so it can take any number of ranks:

```

```
5. Examine the rewritten `printRanks(..)` function below:

```python
0 def printRanks(**kwargs):
1 print(kwargs)
2 >>>
3 printRanks(frst='Amy', sec='Beth', thrd='Carol')
4 {'frst': 'Amy', 'sec': 'Beth', 'thrd': 'Carol'}
```

a. What kind of argument does `printRanks(..)` take? ___________________

b. What type of data structure is `**kwargs` represented by? ___________________

c. Write some lines of code to add to `printRanks(..)` that will print all the data passed in its `**kwargs` parameter in the following example format: “In first place is Amy.”

```python
def printRanks(**kwargs):
    ________________________________________
    ________________________________________
    ________________________________________
    ________________________________________
    ________________________________________
    ________________________________________
    ________________________________________
    ________________________________________
```

6. The following code is from a program, `q20.py`. Examine this commonly used code for obtaining filenames from the command line:

```
q20.py
0 if __name__ == '__main__':
1     from sys import argv
2     print(argv) # printing for learning purposes only!
3     filename = argv[1] if len(argv) > 1 else "default.txt"
4     f = open(filename)
```

a. From what module do we import `argv`? ___________________

b. What is printed to the terminal at line 2? ___________________

c. How does the output differ when the program is called on line 5, versus line 7?

```
IN THE TERMINAL
5 > python3 q20.py whatever
6 ['q20.py', 'whatever']
7 > python3 q20.py whatever another
8 ['q20.py', 'whatever', 'another']
```

d. What kind of argument might `argv` be? ___________________

e. After line 8, what is stored in `argv[0]`? ___________________

f. What is always the first element of `argv`? ___________________

g. Where does `argv` obtain its elements from? ___________________
Rewrite the single line if statement on line 3 into a multi-line if.. statement:

What ends up being stored in `filename` if we call the program with `python3 q20.py`?

Rewrite the sample code to look for a `filename` in the first command line argument, and store everything that follows that first argument into a list called `leftovers`. Remember to handle the case where < 2 command line arguments are passed!

```python
if __name__ == "__main__":

```

Examine this code for unpacking argument lists:

```python
mytuple = tuple(range(7))

# TUPLE ASSIGNMENT REFRESHER
0 >>> (a, b) = (1,2) # a now is 1, b is 2
creating a tuple from a range
1 >>> t = tuple(range(10))
2 >>> t
3 (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
4 >>> (first, *rest) = t
5 >>> first
6 0
7 >>> rest
8 [1, 2, 3, 4, 5, 6, 7, 8, 9]
9 >>> (first, sec, third, *rest) = t
```

a. What kind of assignment are we using on line 4?

b. On line 4, what are we storing in `first`?

c. On line 4, what are we storing in `*rest`?

d. What kind of argument is `*rest`?

e. On line 9, what is stored in `sec` and `third`? `sec = _______` `third = _______`

f. On line 9, what is stored in `*rest`?

g. Write a line of code to store only the last two elements of `mytuple` in `*rest`

```python
mytuple = tuple(range(7))
```