1. The spatial analyst tools have built in classes such as “neighborhood” and “remap” because they allow similar functions and variables to be grouped together and organized. According to ESRI, the Neighborhood class of tools “create output values for each cell location based on the location value and the values identified in a specific neighborhood” (*An overview of neighborhood classes,* ArcGIS Pro, <https://pro.arcgis.com/en/pro-app/arcpy/spatial-analyst/an-overview-of-neighborhood-classes.htm>) while “Remap objects are used to define how the data will be reclassified. Remap objects are lists within lists that are constructed to assign input cell values to new output cell values” (*An overview of remap classes,* ArcMap, <https://pro.arcgis.com/en/pro-app/arcpy/spatial-analyst/an-overview-of-remap-classes.htm>).
2. Temporary rasters are held in memory (RAM or cache) and are likely store the output of intermediate steps in longer scripts and are therefore useful when not required as part of the final product. They can be saved to persistent memory (hard drive) to become “permanent” and reusable.
3. Functions are useful because they allow blocks of code to be reused in different places and from different contexts (even other .py files or modules). Syntax is as follows:

def <function name>(<arguments>):

<function body>

<return value (optional in some cases)>

1. Classes are useful because they allow for functions and data to be easily grouped together. Classes can be instantiated as objects which can hold object-specific data while containing the same methods as other objects in the class. For example a class Car might have objects Ferrari and Honda, which have different speeds but share the method drive().
2. Modules are stored as .py files, where everything (excluding absolute path) preceding “.py” is the module’s name, which can be imported elsewhere.
3. The following code

if \_\_name\_\_ == “\_\_main\_\_”:

is often included in ‘main’ files - .py files that are the main point of execution or contain the bulk of the code – and is used to trigger an overarching function or block of code when the script containing it is run. For example, if a script contains multiple function definitions, code calling those functions might be includes after the above line in order to get triggered when the file is run directly (but not when imported, as would happen if the lines were at the ‘top level’ of a module.