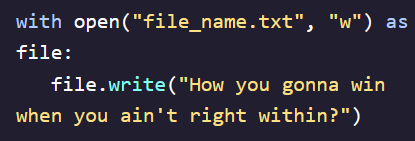
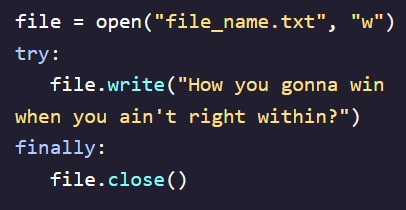
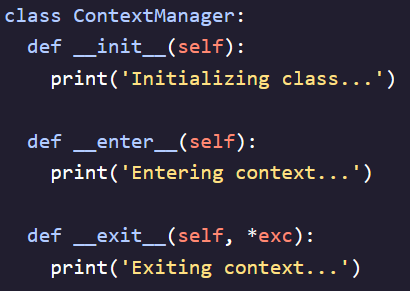
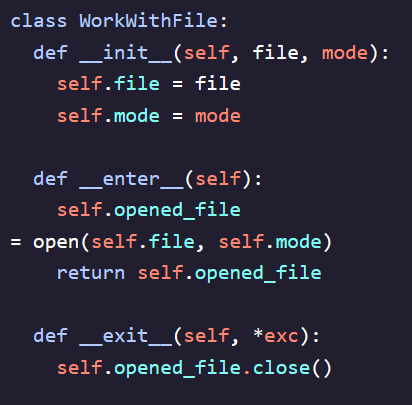
**Resource Management:**

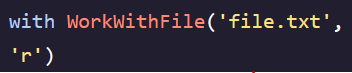
- The resources that need to be managed on a computer come in the form of memory, storage, or power  
- The best way to manage these resources is through the use of ***context managers*** – an object that takes care of assigning and releasing resources (files, database connections, etc)  
- Using these will prevent resource leaks, crashes, program slow-downs, and decrease the vulnerability of data

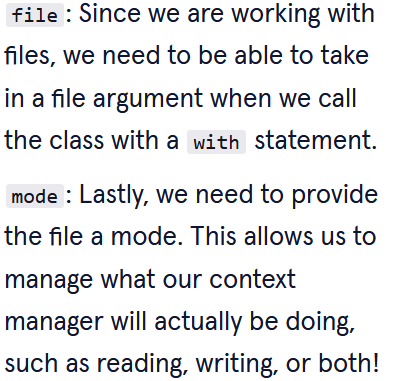
**With Statement:**

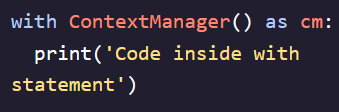
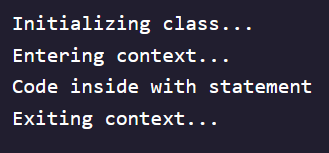
- Using ***with*** is the most common and Pythonic way of invoking context managers  
- Using ***with*** prevents us from having to write code to manually *open*() and *close*() the file, it does it automatically  
 

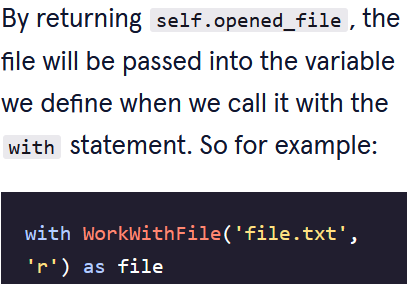
**Class Based Context Managers:**

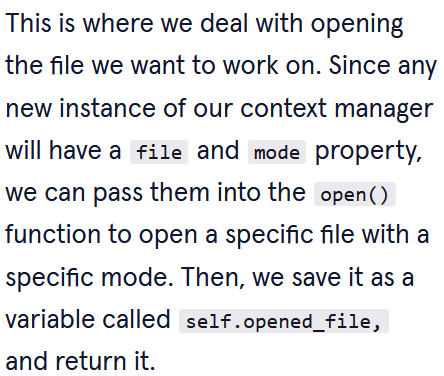
- This approach requires explicitly defining and implementing the ***\_\_enter\_\_()***  and ***\_\_exit\_\_()*** within a class  
- The ***\_\_enter\_\_()***  method allows for the setup of context managers. This method commonly takes care of opening resources (like files)  
 - It also begins what is known as the runtime context - the period of time in which a script runs  
- The ***\_\_exit\_\_()*** method ensures the breakdown of the context manager by closing open resources that are no longer in use  
- After defining, a context manger class can be invoked using a ***with*** statement  
   




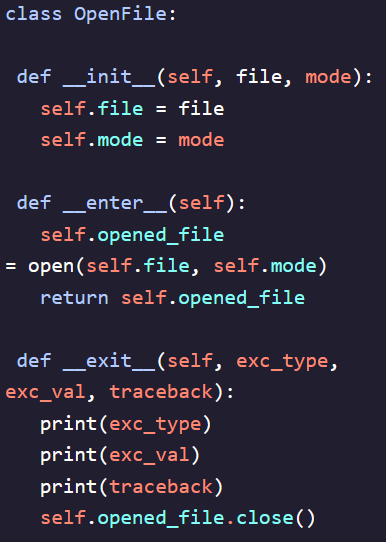


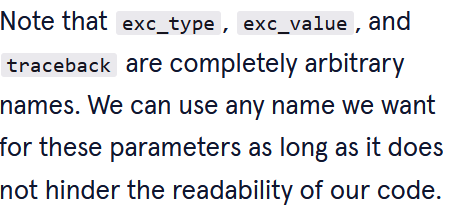
  


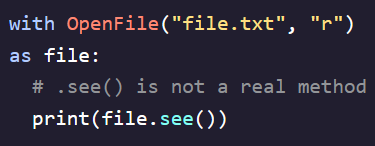
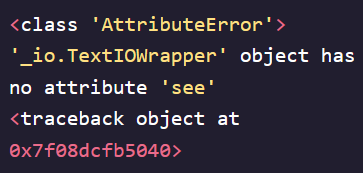
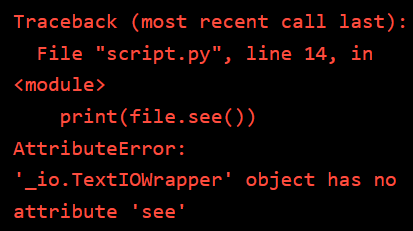


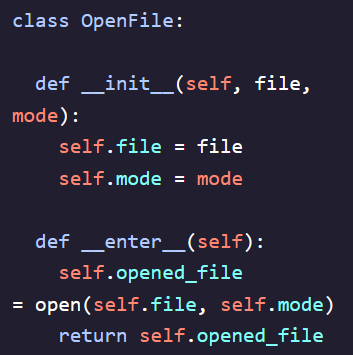
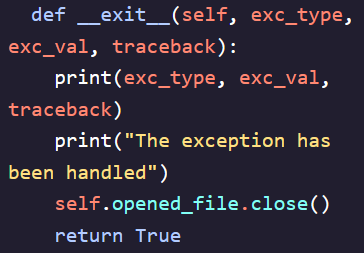


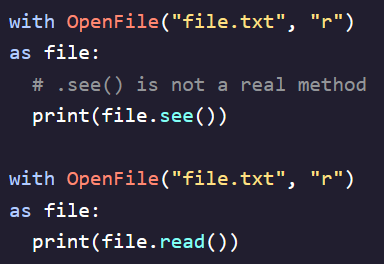
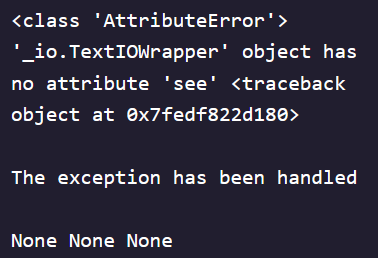
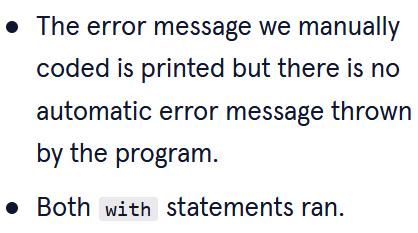
**Handling Exceptions:**

- The ***\_\_exit\_\_*** method is responsible for dealing with any exceptions in code  
- This method requires *4 positional arguments* to execute properly (using \* allowed us to bypass this previously)  
1. self  
2. An exception type which indicates the class of exception (AttributeError or NameError)  
3. An exception value – the actual value of the error  
4. A traceback – a report detailing the sequence of steps that caused the error and all the details needed to fix the error  


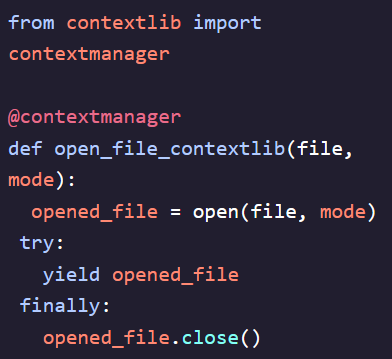
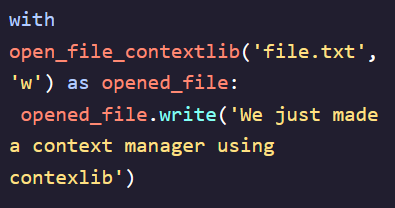


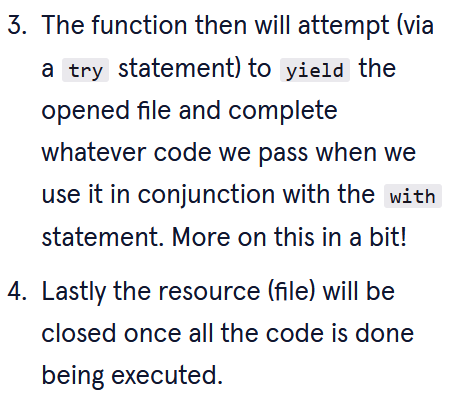
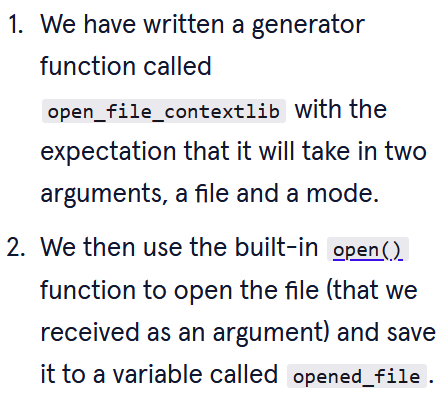
  
  


- If we want to throw an error when an error occurs we can: - If we want to suppress the error we can:  
1. Return ***False*** after the ***.close()*** method 1. Return ***True*** after the ***.close()*** method  
 

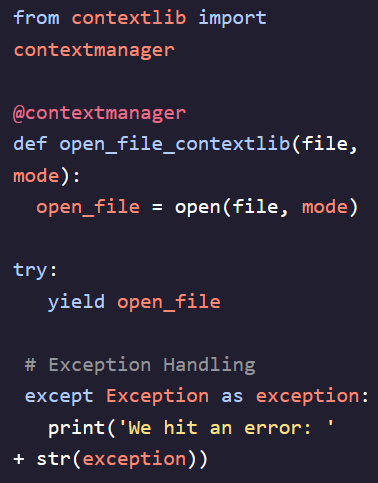
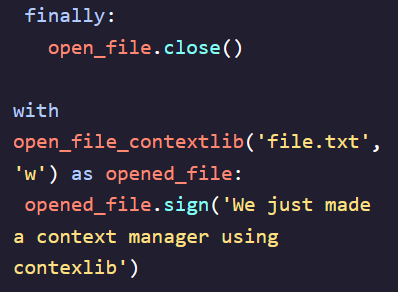
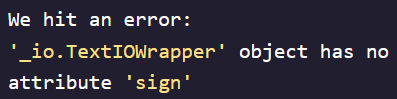
  

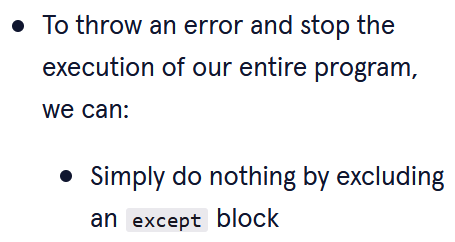
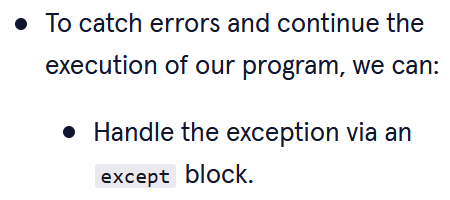
**Contextlib:**

- Allows for the creation of a context manager with the use of a generator function (*yield* instead of *return*) and a contexlib decorator - *@contextmanager*- Need to import contextlib module and then wrap generator function with contextlib decorator  
  




**Contextlib Error Handling:**

- Errors within contextlib are handled by an *except* block built ontop of the *try/finally* block  
  

**Nested Context Managers:**

- There may be a need to use context managers that include working with multiple files (work with information, copy the same information to multiple, copy information from one to another)  
- Context managers can be nested together with a *with* statement to manage multiple resources simultaneously  
