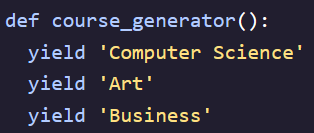
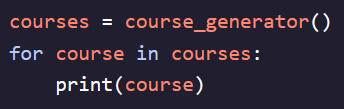
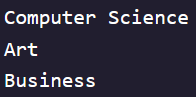
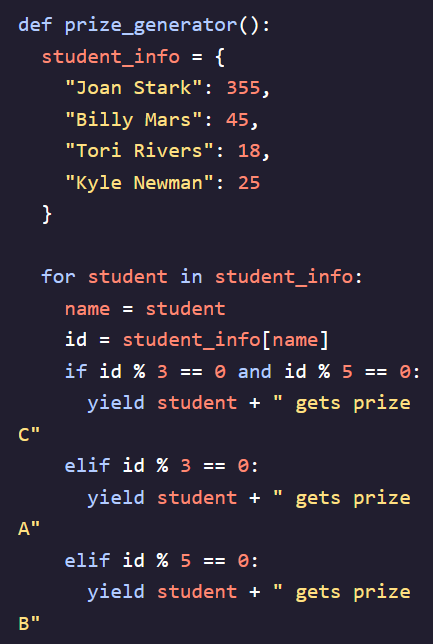
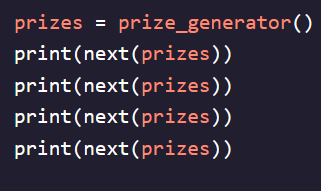
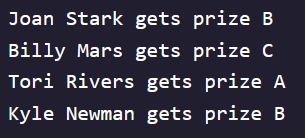
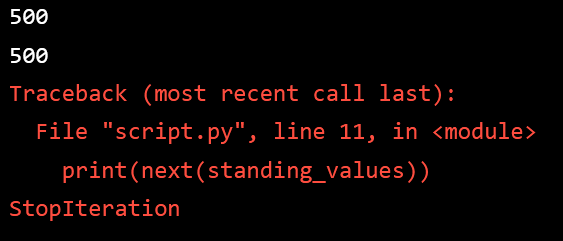
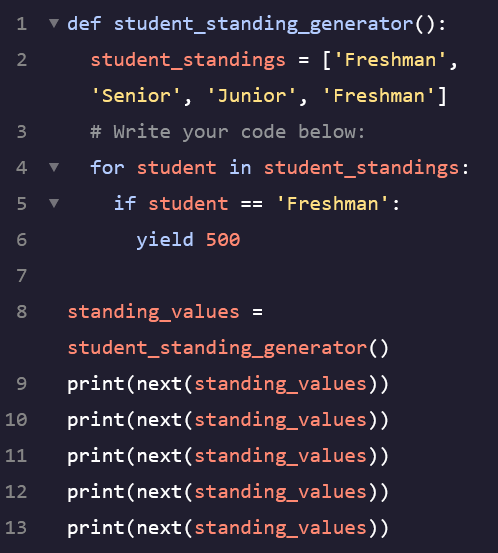
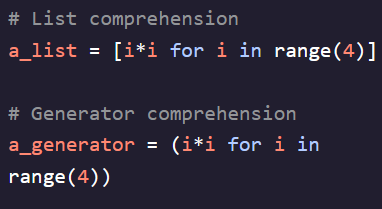
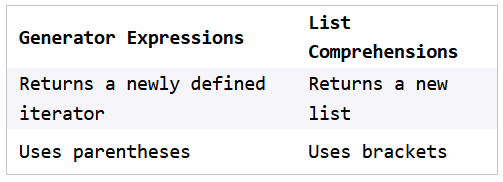
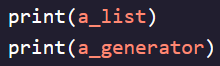
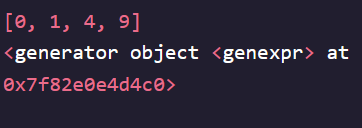
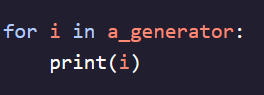
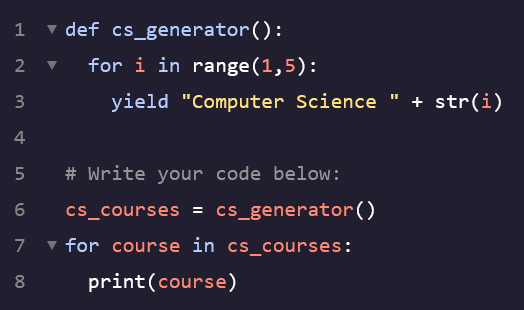
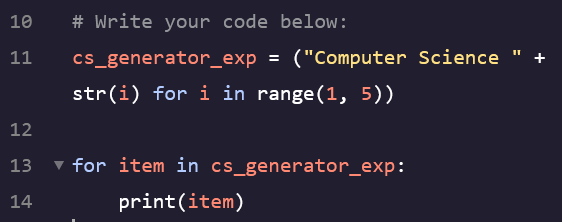
**Generators:**

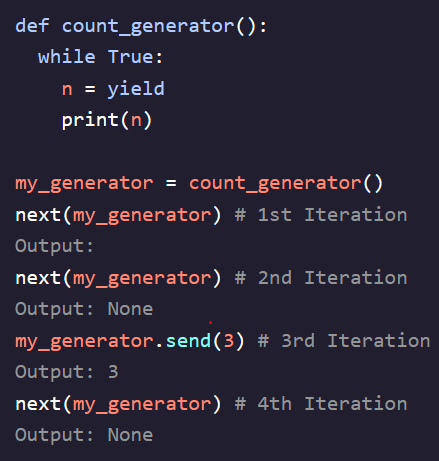
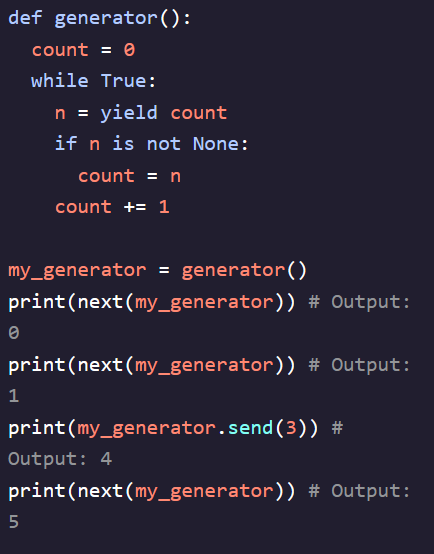
- A *generator* allows for the creation of iterators without having to implement *\_\_iter\_\_()* and *\_\_next\_\_()* methods  
- They improve code readability, save memory by allowing for iterative access of elements, and allow for traversal of infinite streams of data  
  
There are two types of generators in Python:  
- Both of these return a generator object that can be looped over similar to a list but the contents of the generator object are not stored in memory, allowing for complex and infinite iteration of data  
  
1. **Generator Functions** – Similar to regular functions except they must return an iterator but instead of using *return*  they use *yield* - Any code written after *yield* will execute on the next iteration of the iterator whereas code written after *return* will not  
 - *yield* will also suspend the execution of the function and preserve any local variables that exist, *return* will terminate the function immediately and return the results to the caller  
 - The *iterator object* returned by a generator function can be stored in a variable and used later, it can be iterated through as needed  
- *yield* uses the Python built-in function *\_\_next\_\_()* to retrieve the next value and automatically raises a *StopIteration* exception when it runs out of iterable objects  
  
    


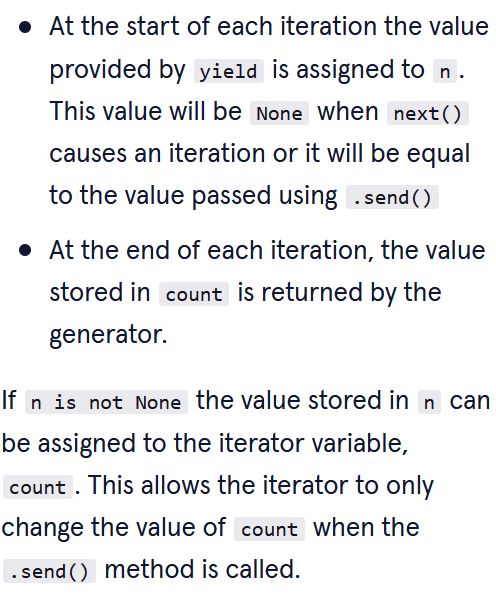
  


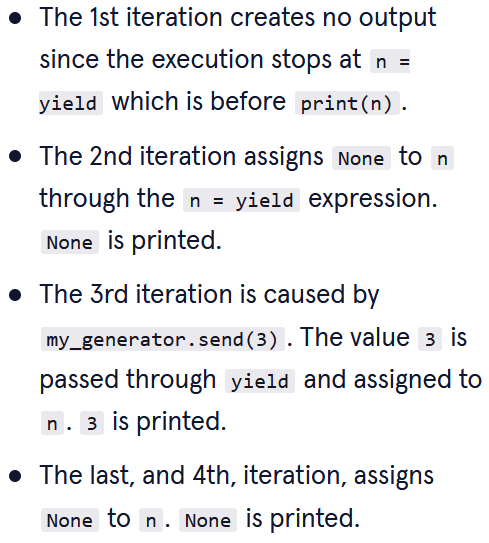


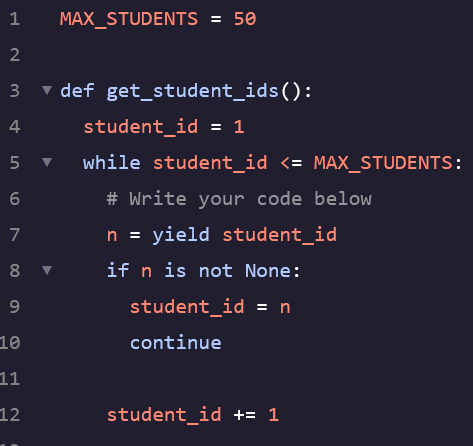
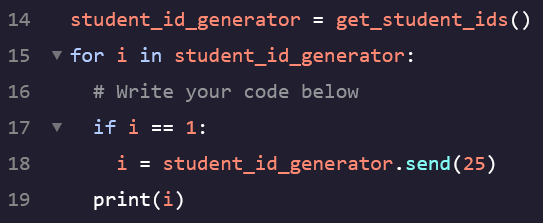
2. **Generator Expressions** – Allow for a clean, single line definition and creation of an iterator with no need to define a full generator function   
- Similar to List Comprehensions but differ in a few ways:  
  
- Cannot be accessed directly like a list comprehension and need to be iterated over  
     
  
 

**Generator Methods – .*send()*:**

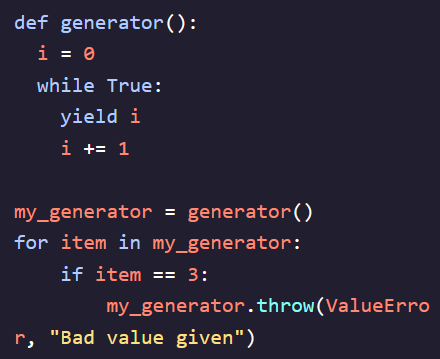
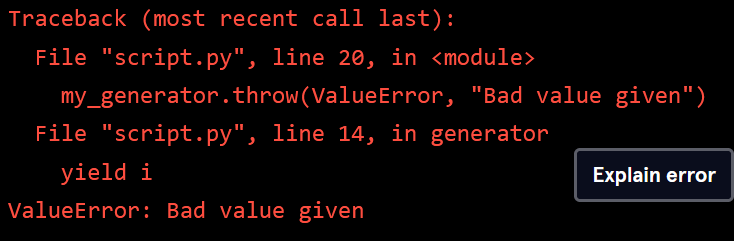
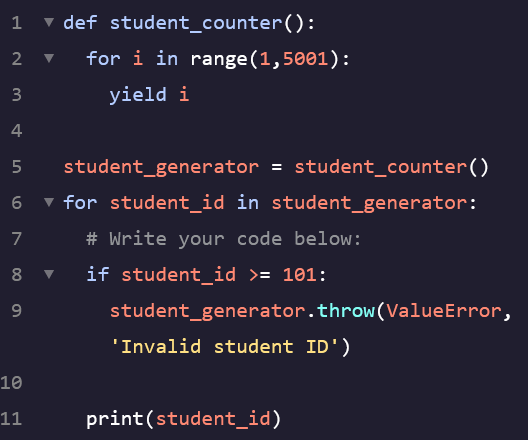
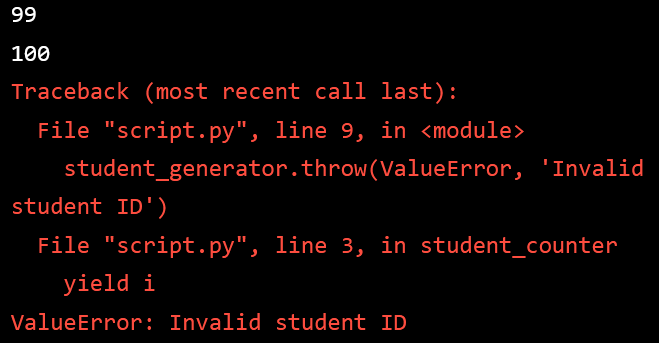
- Allows us to send a value to a generator using the *yield* expression. If you assign *yield* to a variable, the argument passed to the *.send()* method will be assigned to that variable  
  




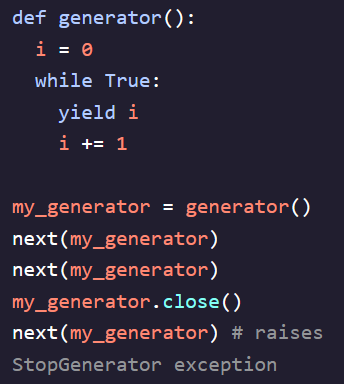
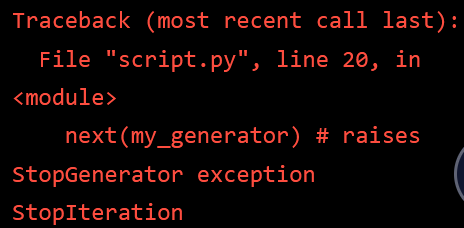
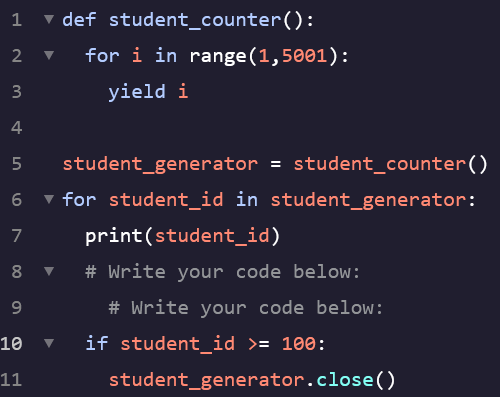


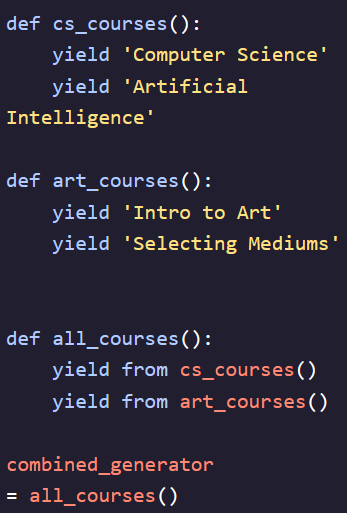
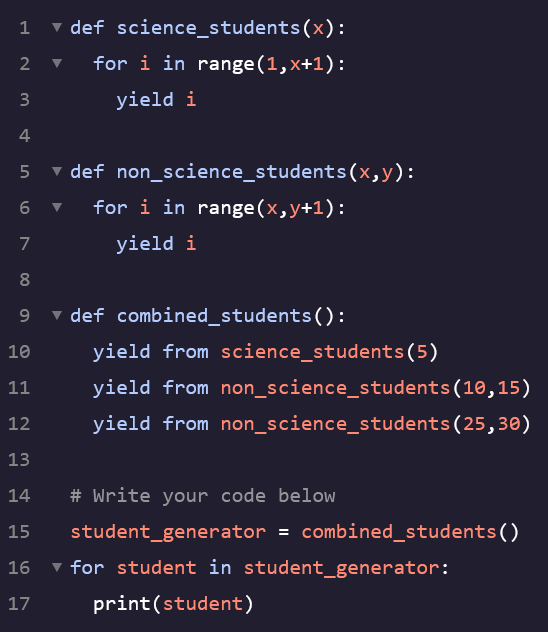
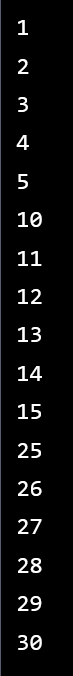
**Generator Methods – .*throw()*:**

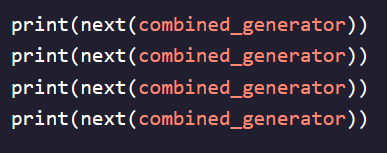
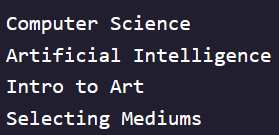
- Provides the ability to throw an exception inside the generator from the caller point, used when we want to end the generator after reaching a certain value or meeting a particular condition  
   
 

**Generator Methods – .*close()*:**

- Used to terminate a generator early, once called a generator is finished just like the end of a *for loop*. Any further iteration attempts will raise a StopIteration exception  
  

**Connection Generators:**

- Able to connect multiple generators into one allowing us to delegate the operations of one generator to another sub-generator, use the *yield from* statement  
- Similar to using the *itertools chain()* method  
   
   
 

**Pipelines Generators/Nested Generators:**

- Allow us to use multiple generators to perform a series of operations all within one expression  
- Used to break down complex operations into smaller, more manageable parts that can then be pipelined together to achieve the desired output  
- The output of one generator function can be the input of another one and so on and so on  
