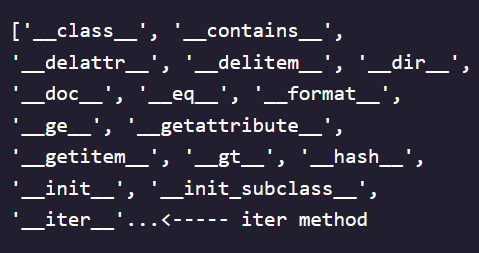
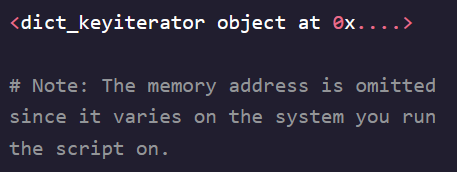
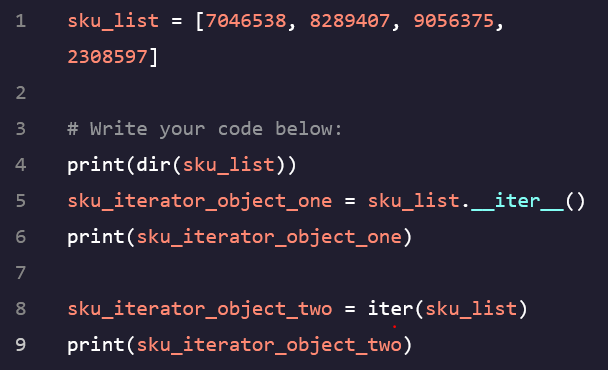
**Iterator Objects I:**

- ***\_\_iter\_\_()*** – A method that is called by the *iter()* function. All iterables have this built-in method pre-defined  
 

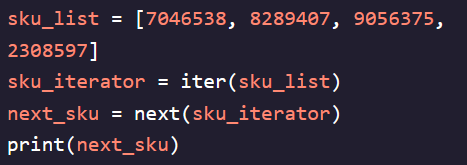
- **iter()** – For loops convert lists and dictionaries into *iterator objects* that are special objects representing a stream of data that can be operated on

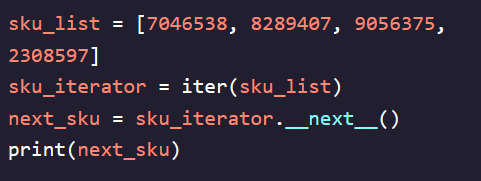
  
  


- Calling *dog\_foods.\_\_iter\_\_()*  is the same as calling *iter(dog\_foods)*

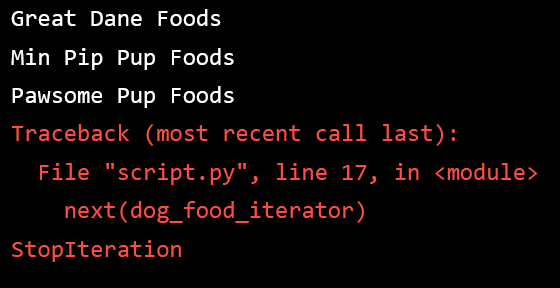
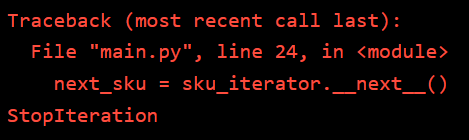
**Iterator Objects II:**

- ***\_\_next\_\_()***  - Each iterator object has a method called *\_\_next\_\_()* that retrieves the iterator’s next value in a *for loop*

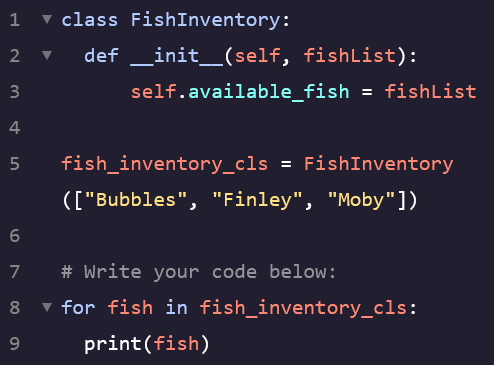
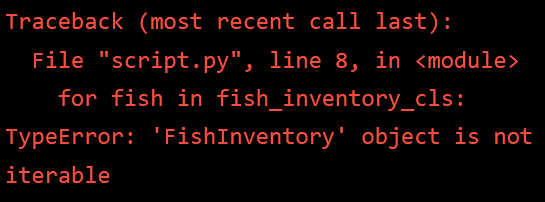
  

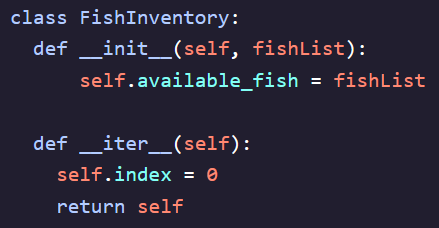
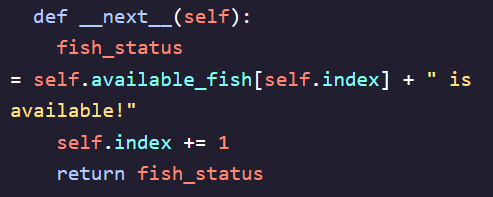

  


- ***next()*** – Similar to *iter()* it calls the iterators object’s, *\_\_next\_\_()* method

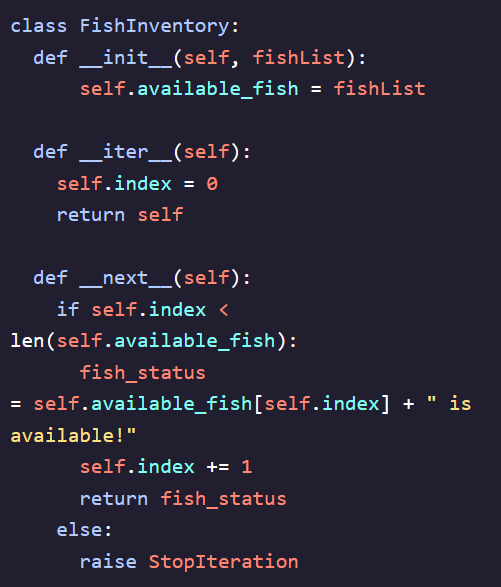
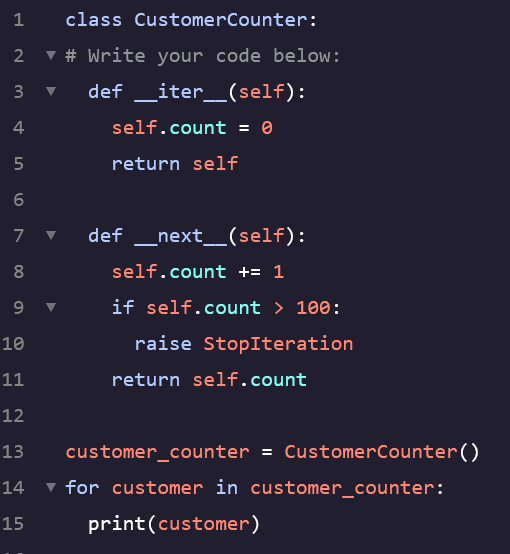
- These methods will call until all items are iterated through and then it raises an exception *StopIteration*

**Custom Iterators:**

- ***Iterator Protocol*** – The implementation of the *\_\_iter\_\_()* and *\_\_next\_\_()* methods that must be implemented for an object to be considered an iterator object  
- To create our own custom iterator, we need a class that defines at a minimum the *\_\_iter\_\_()* and *\_\_next\_\_()* methods  
- By default, custom classes we create are not iterable and we cannot use *for loops* with them  
   
  
- The *\_\_iter\_\_()* methodmust always return the iterator object by itself. This is accomplished by returning *self*  
- The *\_\_next\_\_()* method must either return the next value or raise the *StopIteration* exception.

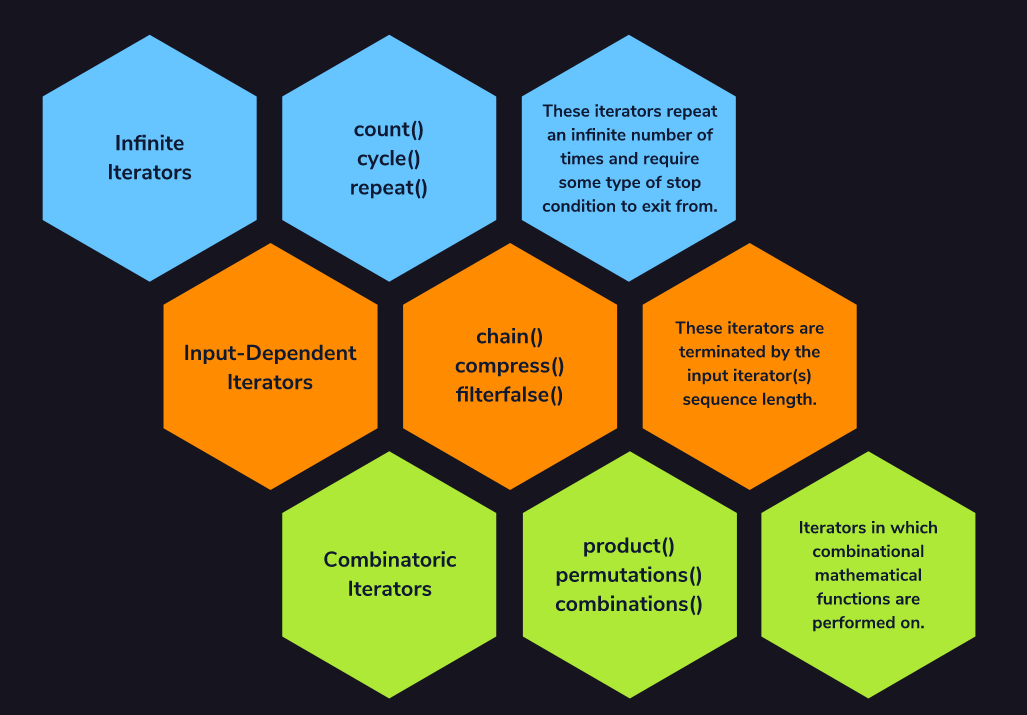
 

We can initialize a class member within the \_\_iter\_\_() method called index that will help us track the current position we’re in within the self.available\_fish list.

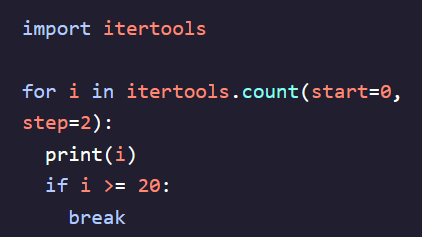
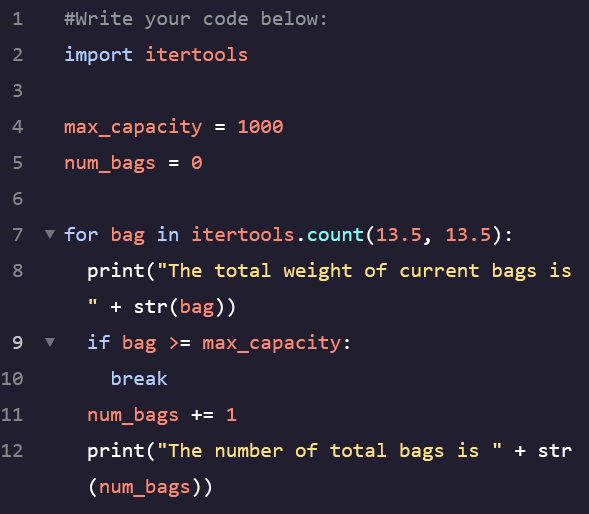
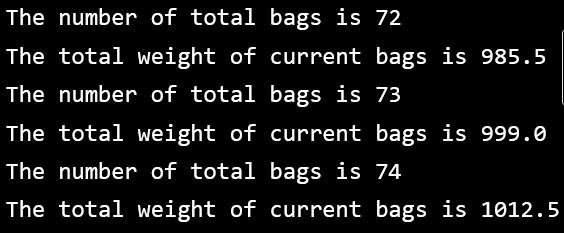
 

Iterating over this class object will eventually error out since we fail to do any checking of our index value against the length of the self.available\_fish list. We can avoid this and cleanly stop the iterator by raising the StopIteration exception in our \_\_next\_\_() method. Here, we’ll modify our \_\_next\_\_() method to raise StopIteration if index exceeds the length of available\_fish.

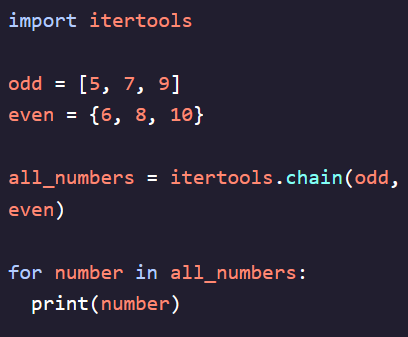
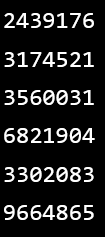
**Python’s Built-In Iterators:**

- While building custom iterator classes can be useful, Python also offers a convenient built-in module called *itertools* that provides the ability to create complex iterator manipulations  
- These operations can either input a single iterable or a combination of them  
  
- Can use by importing the *itertools* module  
  
- Three Categories:  
1. **Infinite** – These will repeat an infinite number of times unless a stop condition is written  
2. **Input-Dependant** – These are terminated by the input iterable(s) sequence length  
3. **Combinatoric** – Combinational mathematical functions are performed on the input iterable(s)

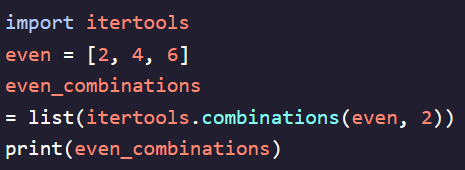
**Infinite Iterator – *count()*:**

- Will repeat an infinite number of times with no endpoint until a *StopIteration* exception is raised  
- *count()* – Will count from a first value until a stop condition is presented  
  
- The first argument is the value we are starting the count from  
- The second argument is an optional step that will return *current value + step*.   
 - This value can be positive, negative, an integer, or a float. Will default to 1 if none provided  
     
 

**Input-Dependent Iterator – *chain()*:**

- Terminates based on the length of one or more input values  
- *chain()* – Takes in one or more iterables and combines them into a single iterator  
  
- The input should be one or more iterables of the same type of varying iterable types (can use to combine a list and set into one iterator)  
- *Set* - An unordered collection of elements without any duplicates (A *list* is an ordered collection and can contain duplicates)  
   

**Combinatoric Iterator – *combinations()*:**

- Will perform a set of statistical or mathematical operations on an input iterable  
- *combinations()* – Will produce an iterator of tuples that contain combinations of all elements in the input  
  
- First argument is an iterable  
- Second argument is a value *r* that represenst the length of each combination tuple  
- The return of *combinations()* is an iterator that can be used in a *for loop* or converted into an iterable type using *list()* or *set()*

The resulting list of 2 member tuples are the combinations of all 3 members of even:  


Suppose we have a list of even numbers and we want all possible combinations of 2 even numbers:

