**Iterators and Generators**

This style of access is clear, concise, and convenient. The use of iterators pervades and unifies Python. Behind the scenes, the for statement calls iter() on the container object. The function returns an iterator object that defines the method \_\_next\_\_() which accesses elements in the container one at a time. When there are no more elements, \_\_next\_\_() raises a StopIteration exception which tells the for loop to terminate.

In programming, an iterator is an object that allows a programmer to traverse a container, particularly lists. Iterators provide a way to access elements of a collection sequentially without exposing the underlying details of the collection's representation.

In many programming languages, including Python, an iterator is an object that adheres to the iterator protocol.

The iterator protocol consists of two methods:

1. **\_\_iter\_\_**: Returns the iterator object itself. This method is required for an object to be considered an iterator.
2. **\_\_next\_\_**: Returns the next element from the iterator. When there are no more items to return, it should raise the **StopIteration** exception to signal the end of the iteration.

**Generators**

Generators are a simple and powerful tool for creating iterators. They are written like regular functions but use the yield statement whenever they want to return data. Each time next() is called on it, the generator resumes where it left off (it remembers all the data values and which statement was last executed).

Simple generators can be coded succinctly as expressions using a syntax similar to list comprehensions but with parentheses instead of square brackets. These expressions are designed for situations where the generator is used right away by an enclosing function. Generator expressions are more compact but less versatile than full generator definitions and tend to be more memory friendly than equivalent list comprehensions.

**Operating System Interface**

The os module provides dozens of functions for interacting with the operating system:

>>>

>>> import os

>>> os.getcwd()      # Return the current working directory

'C:\\Python37'

>>> os.chdir('/server/accesslogs')   # Change current working directory

>>> os.system('mkdir today')   # Run the command mkdir in the system shell.

The sys module also has attributes for stdin, stdout, and stderr. The latter is useful for emitting warnings and error messages to make them visible even when stdout has been redirected:

>>>

>>> sys.stderr.write('Warning, log file not found starting a new one\n')

Warning, log file not found starting a new one.

The most direct way to terminate a script is to use sys.exit().

**String Pattern Matching**

The re module provides regular expression tools for advanced string processing. For complex matching and manipulation, regular expressions offer succinct, optimized solutions:

>>>

>>> import re

>>> re.findall(r'\bf[a-z]\*', 'which foot or hand fell fastest')

['foot', 'fell', 'fastest']

>>> re.sub(r'(\b[a-z]+) \1', r'\1', 'cat in the the hat')

'cat in the hat'

When only simple capabilities are needed, string methods are preferred because they are easier to read and debug:

>>>

>>> 'tea for too'.replace('too', 'two')

'tea for two'

**Internet Access**

There are a number of modules for accessing the internet and processing internet protocols. Two of the simplest are urllib.request for retrieving data from URLs and smtplib for sending mail:

>>>

>>> from urllib.request import urlopen

>>> with urlopen('http://tycho.usno.navy.mil/cgi-bin/timer.pl') as response:

...  for line in response:

...      line = line.decode('utf-8')  # Decoding the binary data to text.

...      if 'EST' in line or 'EDT' in line:  # look for Eastern Time

...             print(line)

<BR>Nov. 25, 09:43:32 PM EST