Iterators vs generators

As per understanding, Iterable is an object which actually has elements stored inside it (E.g. a list). They follow an iteration protocol where they implement __iter__() method which returns an Iterator object which helps in iterating the Iterable.

As per my understanding Generators helps in generating the data on the fly instead of creating a big data structure in memory and returning it. We can achieve similar goal by the use of Iterators as well.

Now my doubt, If we already had Iterators what was the need of Generators, since both helps achieving a similar goal of generating data on the fly. Is that just to simplify the syntax or is there any other reason why Generators exist

```
Iterators

   for variable in iterable:

         statement(s)

   iterable are nothing but collection from where we can

   read one by one value.Eg:  list | set | str | range | tuple

   dict | file | cursor | callable_iterator...

'''

import time

lst=[10,20,30,40]


for i in lst:
    time.sleep(.5)
    print(i)

class Shashi:
    pass


s=Shashi()
for i in s:
    pass



''' lst is an object of <class 'list'> '''

print(dir(lst))



''' Note : if any class is Overridden with

__iter__(self) and __next__(self)

then those closes are acts as iterable classes



__iter__(self) method should return Same class object

__next__(self) method should return next item

       form iterable. whenever __next__() is complited
       its execute then it should raise StopItratorError

import time
class Shashi:
    def __init__(self):
        self.courses=["Java","Python","DM"]
        self.index=-1

    def __iter__(self):
        return self

    def __next__(self):
        self.index=self.index+1
        if self.index>=len(self.courses):
            raise StopIteration
        return self.courses[self.index]

#calling
s=Shashi( )
for i in s:
    time.sleep(1)
    print(i)
```

Example 2:

```
import time

print("Predefined Range Object")
for i in range(1,10):
    time.sleep(.2)
    print(i)
print("- "*30)


class MyRange:
    def __init__(self,start,end):
        self.value=start
        self.end=end

    def __iter__(self):
        return self

    def __next__(self):
        if self.value>self.end:
            raise StopIteration
        curval=self.value
        self.value=self.value+1
        return curval

print("User defied range object ")
for i in MyRange(10,20):
    time.sleep(.5)
    print(i)
```

Generators

```
'''
Generator is an alternative way for
                defining our own iterators.

Note: if  u want define class level iterators we have to
follow some protocals. i.e the class must be overridden
with __iter__(self) and __next__(self)

Defining the generator is nothing defining a function
with yield keyword.

if any function defined by using yield keyword then
that function return generator

generator is a iterator
'''
import time
def myfun():
    yield 10
    yield 20
    yield 30
    yield 40

m=myfun()
print("type of m is : ",type(m))

for i in m:
    time.sleep(1)
    print(i)

'''Note : Generator are best than iterator Reason

  Case 1: Iterator will take more space in the memory
                Generator will take less space in the memory.

  case 2: Generator are executes faster than iterators '''

Example :
import time

def myRange(start,end):
    i=start
    while i<=end:
        yield i
        i=i+1

m=myRange(10,20)
print("Type is : ",type(m))

for i in m:
    time.sleep(.5)
    print(i)


Note:
''' Comprehension
      List comprehension return list collection | where
      list collection is iterator

      Tuple comprehension returns generator | generator
      also an iterator

      Note: If u want know how many bytes are taken
      by an object then we have to use getsizeof( ) from
      sys module.

      '''

import sys

lst=[i for i in range(1,1000000) ]
print("Type is : ",type(lst))
size=sys.getsizeof(lst)
print("Memory taken for Iterator  is :",size)

print("- "*30)
t=(i for i in range(1,1000000))
print("Type is : ",type(t))
size2=sys.getsizeof(t)
print("Memory taken for generator is : ",size2)
```

Note:

Testing Efficiency

```
import timeit
'''
   timeit.timeit(stmt='function',number=int) -> time
'''

def myIterator():
    lst=[i for i in range(1,1000000)]

def myGenerator():
    t=(i for i in range(1,1000000))

ttitr=timeit.timeit(stmt='''def myIterator():
    lst=[i for i in range(1,100000)]''',number=(100))

print("Time taken for iterator ? : ",ttitr)
print("- "*30)

ttgen=timeit.timeit(stmt='''def myGenerator():
    t=(i for i in range(1,100000))''',number=(100))

print("Time Taken for generator ? : ",ttgen)
```