Iterators and Generators

Iteration in Python

- use *for* to iterate over an object

```
>>> for a in [1, 2, 3]:
        print(a)
>>> for b in (1, 2, 3):
        print(b)
>>> for c in {1, 2, 3}:
        print(c)
>>> for d in 'Mozzarella':
        print(d)
>>> for e in {1: 'Python’, 2: ‘C++’}:
        print(e)
>>> for f in open(‘datafile.txt’):
        print(f)
```

- these objects are called *iterable*

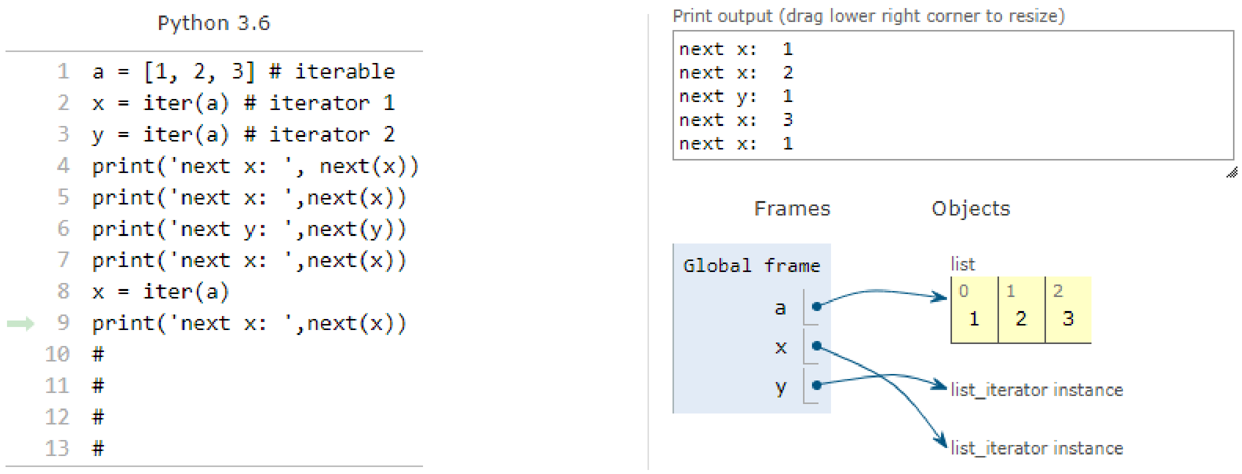
Use of Iterable Objects

- many functions ‘use’ iterable objects

```
>>> ‘$’.join([1, 2, 3])
‘1$2$3’
>>> list(‘Mozzarella’)
[‘M’, ’o’, ’z’, ’z’, ’a’, ’r’, ’e’, ’l’, ’l’, ’a’]
>>> list(range(0, 10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- Python defines an iteration protocol
- *__iter__()* method

Iterable and Iterator Example



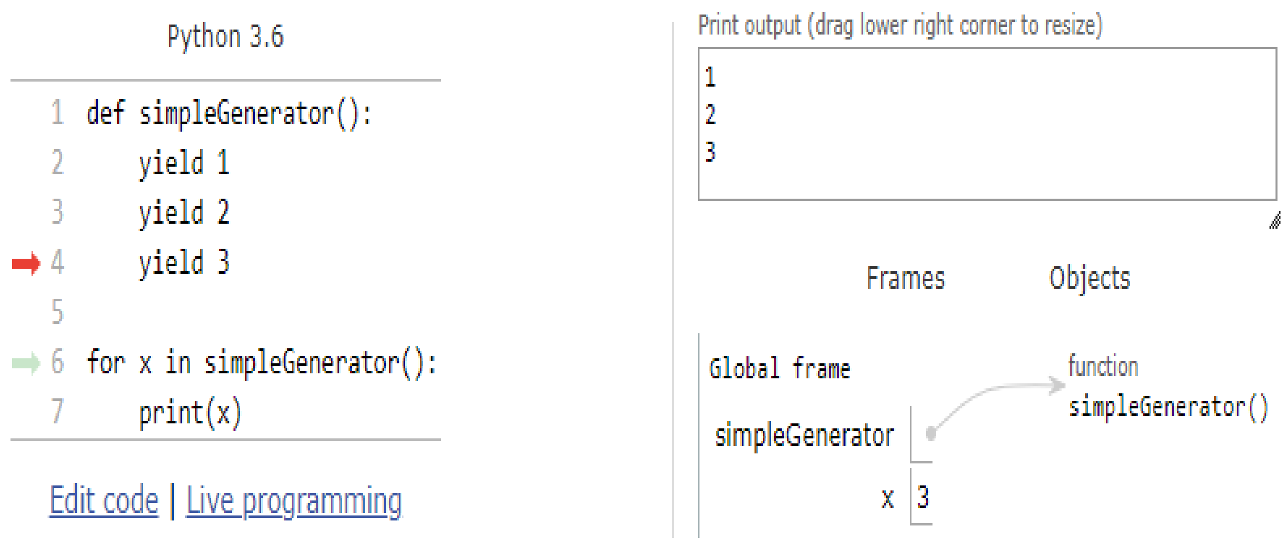
```
a = [1, 2, 3] # iterable
x = iter(a) # iterator 1
y = iter(a) # iterator 2
print('next x: ', next(x))
print('next x: ',next(x))
print('next y: ',next(y))
print('next x: ',next(x))
x = iter(a)
print('next x: ',next(x))
#
#
#
#
```

```
next x:  1
next x:  2
next y:  1
next x:  3
next x:  1
```

- __iter__ method makes an object iterable – it returns an iterator
- iterator (list, tuple) can be iterated over many times
- iterator is an object that iterates
- iterator has __next__() method
- iteration is process of calling __next__()
- iterator raises StopIteration exception when there are no more elements

Generators

- functions that behave like iterators
- iterators implement __iter__() and __next__()
- iterators do not compute all values
- computation on-demand (“lazy evaluation”)
- useful for very large data sets
- generators use yield function



```
def simpleGenerator():
    yield 1
    yield 2
    yield 3

for x in simpleGenerator():
    print(x)
```

```
1
2
3
```

Iterable and Iterator

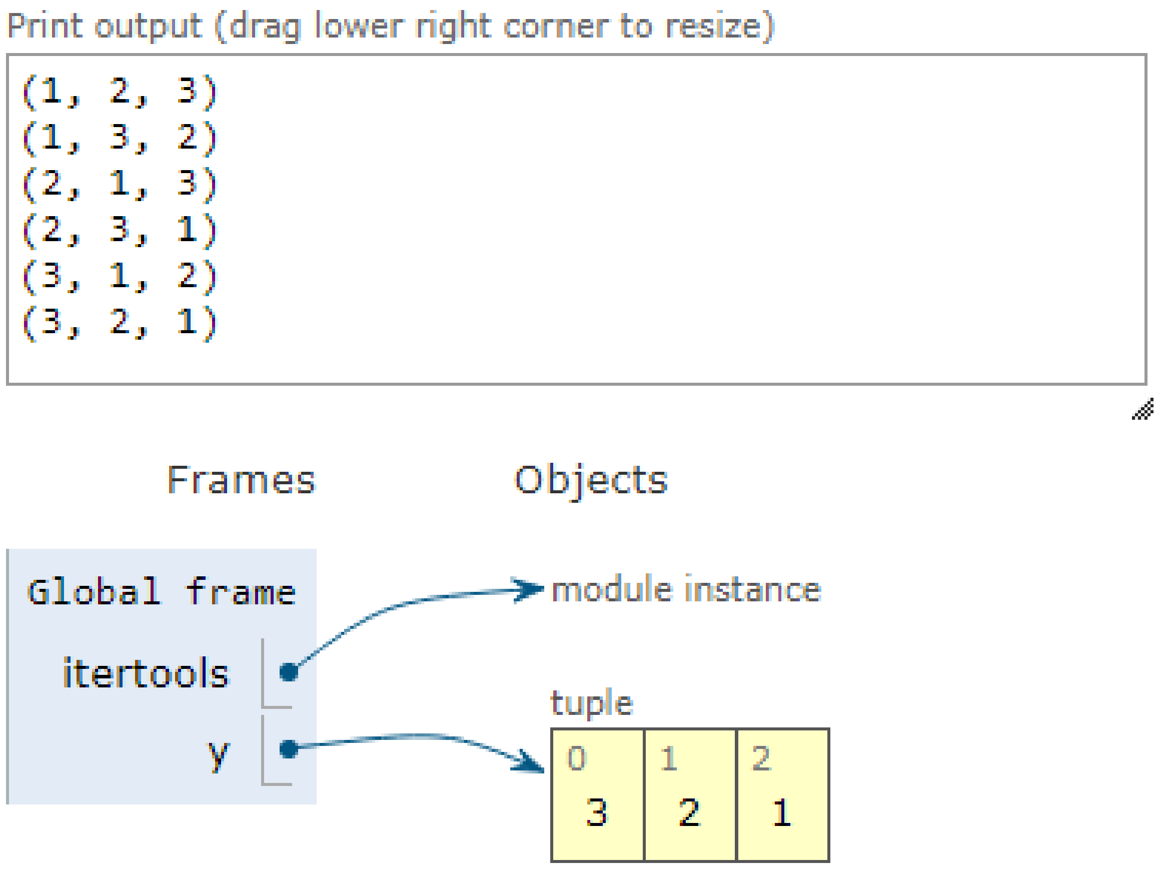
- iterations refers to getting items
- iterable is an object with __iter__ method
- this method defines __getitem__
- __getitem__ returns next sequential element
- iterator is an object with __next__ method

- Python defines an iteration protocol
- *__iter__()* method

itertools Package

- special module to manipulate iterables

```
import itertools
for y in itertools.permutations([1,2,3]):
    print(y)
```



Print output (drag lower right corner to resize)

```
(1, 2, 3)
(1, 3, 2)
(2, 1, 3)
(2, 3, 1)
(3, 1, 2)
(3, 2, 1)
```