What is an Iteration

Iteration is a general term for taking each item of something, one after another. Any time you use a loop, explicit or implicit, to go over a group of items, that is iteration.

In [19]:
```
# Example
num = [1,2,3]

for i in num:
    print(i)
```

```
1
2
3
```

What is Iterator

An Iterator is an object that allows the programmer to traverse through a sequence of data without having to store the entire data in the memory

In [28]:
```
# Example
L = [x for x in range(1,10000)]

#for i in L:
    #print(i*2)

import sys

print(sys.getsizeof(L)/64)

x = range(1,10000000000)

#for i in x:
    #print(i*2)

print(sys.getsizeof(x)/64)
```

```
1369.0
0.75
```

What is Iterable

Iterable is an object, which one can iterate over

It generates an Iterator when passed to iter() method.

In [32]:
```
# Example

L = [1,2,3]
type(L)


# L is an iterable
```

```
# L is an iterable
type(iter(L))

# iter(L) --> iterator
```

Out[32]: list_iterator

Point to remember

- Every **Iterator** is also and **Iterable**
- Not all **Iterables** are **Iterators**

Trick

- Every Iterable has an **iter function**
- Every Iterator has both **iter function** as well as a **next function**

In [35]:
```
a = 2
a

#for i in a:
    #print(i)

dir(a)
```

Out[35]:
```
['__abs__',
 '__add__',
 '__and__',
 '__bool__',
 '__ceil__',
 '__class__',
 '__delattr__',
 '__dir__',
 '__divmod__',
 '__doc__',
 '__eq__',
 '__float__',
 '__floor__',
 '__floordiv__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getnewargs__',
 '__gt__',
 '__hash__',
 '__index__',
 '__init__',
 '__init_subclass__',
 '__int__',
 '__invert__',
 '__le__',
 '__lshift__',
 '__lt__',
 '__mod__',
 '__mul__',
 '__ne__',
 '__neg__',
 '__new__',
```

```
'__or__',
'__pos__',
'__pow__',
'__radd__',
'__rand__',
'__rdivmod__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__rfloordiv__',
'__rlshift__',
'__rmod__',
'__rmul__',
'__ror__',
'__round__',
'__rpow__',
'__rrshift__',
'__rshift__',
'__rsub__',
'__rtruediv__',
'__rxor__',
'__setattr__',
'__sizeof__',
'__str__',
'__sub__',
'__subclasshook__',
'__truediv__',
'__trunc__',
'__xor__',
'as_integer_ratio',
'bit_length',
'conjugate',
'denominator',
'from_bytes',
'imag',
'numerator',
'real',
'to_bytes']
```

In [38]:

```
T = {1:2,3:4}
dir(T)
```

Out[38]:

```
['__class__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__ne__',
 '__new__',
 '__reduce__',
```

```
'__reduce_ex__',
'__repr__',
'__reversed__',
'__setattr__',
'__setitem__',
'__sizeof__',
'__str__',
'__subclasshook__',
'clear',
'copy',
'fromkeys',
'get',
'items',
'keys',
'pop',
'popitem',
'setdefault',
'update',
'values']
```

In [41]:

```
L = [1,2,3]

# L is not an iterator
iter_L = iter(L)

# iter_L is an iterator
```

Out[41]:

```
['__class__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__length_hint__',
 '__lt__',
 '__ne__',
 '__new__',
 '__next__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__setstate__',
 '__sizeof__',
 '__str__',
 '__subclasshook__']
```

Understanding how for loop works

In [42]:

```
num = [1,2,3]

for i in num:
    print(i)
```

```
1
2
3
```

In [47]:

```
num = [1,2,3]

# fetch the iterator
iter_num = iter(num)

# step2 --> next
next(iter_num)
next(iter_num)
next(iter_num)
next(iter_num)
```

```
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-47-0793ebf651f8> in <module>
      8 next(iter_num)
      9 next(iter_num)
---> 10 next(iter_num)

StopIteration:
```

Making our own for loop

In [48]:

```
def mera_khudka_for_loop(iterable):

    iterator = iter(iterable)

    while True:

        try:
            print(next(iterator))
        except StopIteration:
            break
```

In [53]:

```
a = [1,2,3]
b = range(1,11)
c = (1,2,3)
d = {1,2,3}
e = {0:1,1:1}

mera_khudka_for_loop(e)
```

```
0
1
```

A confusing point

In [64]:

```
num = [1,2,3]
iter_obj = iter(num)

print(id(iter_obj),'Address of iterator 1')
```

```
iter_obj2 = iter(iter_obj)
print(id(iter_obj2),'Address of iterator 2')
```

```
2280889893936 Address of iterator 1
2280889893936 Address of iterator 2
```

Let's create our own range() function

In [67]:
```
class mera_range:

    def __init__(self,start,end):
        self.start = start
        self.end = end

    def __iter__(self):
        return mera_range_iterator(self)
```

In [68]:
```
class mera_range_iterator:

    def __init__(self,iterable_obj):
        self.iterable = iterable_obj

    def __iter__(self):
        return self

    def __next__(self):

        if self.iterable.start >= self.iterable.end:
            raise StopIteration

        current = self.iterable.start
        self.iterable.start+=1
        return current
```

In [70]:
```
x = mera_range(1,11)
```

In [71]:
```
type(x)
```

Out[71]:
```
__main__.mera_range
```

In [72]:
```
iter(x)
```

Out[72]:
```
<__main__.mera_range_iterator at 0x2130fd362b0>
```

In [ ]: