# Iteration



## What is Iteration

It 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.

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

for i in num:
        print(i)
# 1
# 2
# 3
```

## What is Iterator?

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

```
x = range(1, 100) --> # this is iterator

L = [x for x in range(1, 100)]

# L took 1mb memory of ram and iterator took 0.75 kb memory of ram.
```

## What is Iterable

It is an object, which one can iterate over ( loop over)

It generates an iterator when passed to `iter()` method

```
L = [1,2,3]
type(L) --> list # which is an Iterable
```

```
type(iter(L)) --> list_iterator # which is an Iterator
```

# Point to remember

- Every `Iterator` is also an `Iterable`
- Not all `Iterables` are `Iterators`.

# Tricks

### Iterable

- Use `dir()` function on variable, to check if it is an iterable, if u see `__iter__` in `dir()` of that variable then it is iterable.

### Iterator

- To check if variable is iterator, if u see `__iter__` & `__next__` both in `dir()` of that variable then it is iterator.

# Making our own for Loop

```
def my_loop(iterable):

        iterator = iter(iterable)

        while True:

                try:
                        print(next(iterator))

                except StopIteration:
                        break

a = [1,3,4]
# 1
# 3
# 4
```

# Confusing point

```
num = [1,2,3]
iter_obj1 = iter(num)

print(id(iter_obj1))  # let say it is 1069

iter_obj2 = iter(obj_1)

print(id(iter_obj2)) # it will be same 1069
```

```
# because iterator just change its name not the address in memory location
```

## Let’s create our own range() function

```
class mera_range:

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

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

```
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
```

```
for i in mera_range(1,3):
        print(i)

# 1
# 2
# 3
```