Introduction to Generator and Iterator

- At the end of this class, students will be able to-
  - Classify Generator and Iterators.
  - Compare and Contrast between **Generator function and a Normal function**
  -

PES University

Generators

- A generator is a function that returns an object (iterator) which we can iterate over (one value at a time).
- generators are a simple way of creating iterators.

PES University

How to create a generator

- It is as easy as defining a normal function with yield statement instead of a return statement.

```
def my_gen():
    n = 1
    print('This is printed first')
    yield n
    n += 1
    print('This is printed second')
    yield n
    n += 1
    print('This is printed at last')
    yield n
a=my_gen()
print(next(a))
```

PES University

next()

- next() is a special function that asks, "What's the next item in the iteration?“
- next() is the precise function that is called in the for loop.

PES University

Differences between Generator function and a Normal function

- Generator function contains one or more yield statement.
- When called, it returns an object (iterator) but does not start execution immediately.
- Once the function yields, the function is paused and the control is transferred to the caller.
- Local variables and their states are remembered between successive calls.

PES University

- Write a program to print 25 prime numbers

PES University

```
def is_prime(m):
    i = 2
    while m % i != 0 :
            i += 1
    return i == m
def gen():
    yield 2
    yield 3
    m = 5
    while True:
            if is_prime(m) :
                    yield m
            m += 2
```

```
g = gen()
# get next n primes
n = 25
for i in range(n):
    print(next(g))
```

PES University

Iterators

- An iterator is an object that will allow you to iterate over a container.
- Iterator in Python is simply an object that can be iterated upon. An object which will return data, one element at a time.
- Python lists, tuples, dicts and sets are all examples of inbuilt iterators.
- Python **iterator object** must implement two special methods, __iter__()and __next__(), collectively called the **iterator protocol**.

PES University

Iterator protocol

- These types are iterators because they implement following methods. In fact, any object that wants to be an iterator must implement following methods.
- **__iter__**
- **next (__next__)**

PES University

__iter__

- This method that is called on initialization of an iterator.
- The **__iter__** method is required for your container to provide iteration support. It will return the iterator object itself. But if you want to create an iterator object, then you will need to define **__next__** as well, which will return the next item in the container.

PES University

next (__next__)

- The iterator next method should return the next value for the iterable. When an iterator is used with a ‘for in’ loop, the for loop implicitly calls next() on the iterator object. This method should raise a StopIteration to signal the end of the iteration.

PES University

Building Your Own Iterator in Python

```
class PowTwo:
    """Class to implement an iterator
    of powers of two"""“
    def __init__(self, m = 0):
        self.max = m
    def __iter__(self):
        self.n = 0
        return self
    def __next__(self):
        if self.n <= self.max:
            result = 2 ** self.n
            self.n += 1
            return result
        else:
            raise StopIteration
```

```
print(PowTwo.__doc__)
for i in PowTwo(5):
        print(i)
```

Class to implement an iterator
of powers of two
1
2
4
8
16
32

PES University

Without for loop

```
s=PowTwo(3)
first=iter(s)
print(first)
print(next(first))
print(next(first))
print(next(first))
print(next(first))
print(next(first))
```

```
<__main__.PowTwo object at
0x000001F7711EF470>
1
2
4
8
Traceback (most recent call last):
  File StopIteration
```

PES University

```
class MyContainer:
    def __init__(self, mylist):
        self.mylist = mylist
    def __iter__(self):
        self.i = 0
        return self
    def __next__(self):
        self.i += 1
        if self.i <= len(self.mylist):
            return self.mylist[self.i - 1]
        else:
            raise StopIteration
```

PES University

```
a = [ 'apple', 'banana', 'carrot', 'date', 'eff', 'fish' ]
c = MyContainer(a)
it1 = iter(c)
it2 = iter(c)
print(next(it1)) # apple
print(next(it2)) # expect apple, we will get banana
```

PES University