Python

1. Find expected output:

```
l = [10,20,30,40]
itr = iter(l)
print(next(itr), next(itr))
itr = iter(l)
print(next(itr), next(itr))
```

```
itr =  [10,20,30,40].__iter__()
print(itr.__ next__(), itr.__ next__())
itr = reversed([10,20,30,40])
print(itr.__ next__(), itr.__ next__())
```

```
itr1 = range(10,20)
itr2 = range(1,10,4)
print(next(itr1))
print(next(itr2))
itr3 = iter(itr1)
print(next(itr3))
print(next(itr2))
print(next(itr1))
```

```
def num_gen(start, end, diff=1):
        while start < end:
                yield start
                start = start + diff

g1 = num_gen(10,20)
g2 = num_gen(1,10,4)
print(next(g1))
print(next(g2))
g3 = iter(g1)
print(next(g3))
print(next(g2))
print(next(g1))
```

2. Which exception is raised upon reaching the last element of on iterable via its iterator.
3. Name the two methods that are required for the iterator protocol.
4. Write a Function that takes a type or object or variable as argument and returns True or False depending on whether the argument is an iterable or not. (**Hint:** Use the dir method to get a list of supported operations)

```
def is_iterable(in_type):
        # write your code here to return True or False


print(is_iterable(list)) # should print True

print(is_iterable(int)) # should print False
```

5. Write a generator, that generates Fibonacci numbers. The function takes a number as argument and generates numbers less than or equal to that number.
6. Write a generator that takes a list and a predicate function (or lambda) as arguments and gives values after applying the lambda to the elements of the list. (The elements present in the list itself should not change)