**Python - 3**

**Exercises Answers**

**1. What is the return type of "range()" function?**

List

**2. Show your understanding of iterables vs iterators with code examples**

## **What is an Iterable?**

In Python, an Iterable is an object that implements the \_\_iter\_\_() method and returns an iterator object or an object that implements \_\_getitem\_\_() method (and should raise an IndexError when indices are exhausted). Built-in iterable objects include Lists, Sets and Strings as such sequences can be iterated over -say- in a for-loop.

**What is an iterators?**

On the other hand, an Iterator in Python is an object that implements the \_\_next\_\_() method in a way that

* the next value of an iterable object is being returned and the state of the iterator gets updated so that it points to the next value
* raises a StopIteration when the elements of the iterable object are exhausted

Additionally, an Iterator is an Iterable itself as it must also implement \_\_iter\_\_() method where it simply returns self.

>>> my\_lst = [5, 10, 15]

>>> my\_iter = iter(my\_lst)

>>> type(my\_iter)

list\_iterator

>>> my\_iter.\_\_next\_\_()

5

**Buildind custom iterators**

class PowTwo:

"""Class to implement an iterator

of powers of two"""

def \_\_init\_\_(self, max=0):

self.max = max

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

# create an object

numbers = PowTwo(3)

# create an iterable from the object

i = iter(numbers)

# Using next to get to the next iterator element

print(next(i))

print(next(i))

print(next(i))

print(next(i))

print(next(i))

**3. What is the difference between yield and return?**

<https://www.programiz.com/python-programming/generator>

Here is how a generator function differs from a normal [function](https://www.programiz.com/python-programming/function).

* Generator function contains one or more yield statements.
* When called, it returns an object (iterator) but does not start execution immediately.
* Methods like \_\_iter\_\_() and \_\_next\_\_() are implemented automatically. So we can iterate through the items using next().
* 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.
* Finally, when the function terminates, StopIteration is raised automatically on further calls.

#A simple generator function

def my\_gen():

n = 1

print('This is printed first')

# Generator function contains yield statements

yield n

n += 1

print('This is printed second')

yield n

n += 1

print('This is printed at last')

yield n

**4. When to use yield?**

We should use yield when we want to iterate over a sequence, but don’t want to store the entire sequence in memory.

**5. Show the usage of \_\_iter\_\_ and \_\_next\_\_ methods**

The \_\_iter\_\_() method returns the iterator object itself. If required, some initialization can be performed.

The \_\_next\_\_() method must return the next item in the sequence. On reaching the end, and in subsequent calls, it must raise StopIteration.