# Iterator and iterable

*# The* ***iter()*** *method* ***returns an iterator for the given object****.*

*# The* ***iter()*** *method creates an object which can be iterated one element at a time.*   
*# An* ***iterator*** *is an object with a state that it remembers where it is during an iteration.*

*# Iterators also know how to get their next value and they get their next value with \_\_next\_\_ method.*

*# if something is iterable then it needs to have \_\_iter\_\_() method* ***or in other word*** *something can be looped over if it has \_\_iter\_\_() method in it.*

*# a list is an iterable but it is not iterator*

nums = [12, 13, 15, 16]  
  
*# i\_nums = nums.\_\_iter\_\_()*i\_nums = iter(nums) # *<list\_iterator object at 0x00000185724B4A58>*  
print(next(i\_nums)) *# 12*print(next(i\_nums)) *# 13*print(next(i\_nums)) *# 15*print(next(i\_nums)) *# 16  
# print(next(i\_nums)) # raises* ***StopIteration*** *error*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*# This simple while loop works kinda same as below*

num = [1, 2, 3, 4, 5]  
i = iter(num) *# this will return an iterator object of num var*  
  
while True:  
 try:  
 print(next(i))  
 except StopIteration: *# stop iteration or it will go till forever*  
 break

` \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*# MyRange class is iterable because we can iterate over it with*

*# for loop this class is also iterator because it has dunder next*

*# method*

class MyRange:  
 def \_\_init\_\_(self, start, end):  
 self.start = start  
 self.end = end  
  
 def \_\_iter\_\_(self):  
 return self  
  
 def \_\_next\_\_(self):  
 if self.start >= self.end:  
 raise StopIteration  
 current = self.start  
 self.start += 1  
 return current

val1 = MyRange(5,10)for i in val1:  
 print(i)  
*# Output : 5 6 7 8 9*

*# You can use generators as well and that would be clean code than above code. So, generators are iterators as well, but \_\_iter\_\_() and \_\_next\_\_() methods are created automatically.*def my\_range(start, end):  
 current = start  
 while current < end:  
 yield current  
 current += 1

*# this generator is identical to the one above*def my\_range(start, end):  
 while start < end:  
 yield start  
 start += 1

sequence = my\_range(5, 10)  
for i in sequence:  
 print(i)  
*# 5 6 7 8 9*

# Problem - Creating your own iterators

class Sentence:  
 def \_\_init\_\_(self, sentence):  
 self.sentence = sentence  
 self.words = self.sentence.split()  
 self.index = 0  
  
 def \_\_iter\_\_(self):  
 return self  
  
 def \_\_next\_\_(self):  
 if self.index >= len(self.words):  
 raise StopIteration  
 index = self.index  
 self.index += 1  
 return self.words[index]  
  
  
txt = Sentence("Hello world! This is Basir Payenda.")  
  
for i in txt:  
 print(i)

# output:

# Hello

# world!

# This

# is

# Basir

# Payenda.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Or you can use a generator to acquire the same result above

**def** sentence(txt):  
 **for** word **in** txt.split():  
 **yield** word  
  
  
txt = sentence(**"Hello world! This is Basir Payenda."**)  
  
**for** i **in** txt:  
 print(i)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

nums = [12, 13, 14, 15, 16]

for i in nums:

print(i)

# how for loop works under the hood:

# 1. it calls iter() method and this method returns an iterator -> iter(nums)

# 2. the iterator loops every single item using next() method -> next(iter(nums))

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_