Iterators

Python provides a great module for creating your own iterators. The module I am referring to is **itertools**. The tools provided by itertools are fast and memory efficient. You will be able to take these building blocks to create your own specialized iterators that can be used for efficient looping. In this chapter, we will be looking at examples of each building block so that by the end you will understand how to use them for your own code bases.

Let's get started by looking at some infinite iterators!

The Infinite Iterators

The itertools package comes with three iterators that can iterate infinitely. What this means is that when you use them, you need to understand that you will need to break out of these iterators eventually or you'll have an infinite loop.

These can be useful for generating numbers or cycling over iterables of unknown length, for example. Let's get started learning about these interesting iterables!

count(start=0, step=1)

The **count** iterator will return evenly spaced values starting with the number you pass in as its **start** parameter. Count also accept a **step** parameter. Let's take a look at a simple example:

```
from itertools import count
for i in count(10):
    if i > 20:
        break
    else:
        print(i)

#10
#11
```

```
#12
#13
#14

#15
#16
#17
#18
#19
#20
```

Here we import **count** from itertools and we create a **for** loop. We add a conditional check that will break out of the loop should the iterator exceed 20, otherwise it prints out where we are in the iterator. You will note that the output starts at 10 as that was what we passed to **count** as our start value.

Another way to limit the output of this infinite iterator is to use another sub-module from itertools, namely **islice**. Here's how:

```
from itertools import count
from itertools import islice
for i in islice(count(10), 5):
    print(i)

#10
#11
#12
#13
#14
```

In this example we import **islice** and we loop over **count** starting at 10 and ending after 5 items. As you may have guessed, the second argument to islice is when to stop iterating. But it doesn't mean "stop when I reach the number 5". Instead, it means "stop when we've reached five iterations".

cycle(iterable)

The **cycle** iterator from itertools allows you to create an iterator that will cycle through a series of values infinitely. Let's pass it a 3 letter string and see what happens:

```
from itertools import cycle
```

```
count = 0
for item in cycle('XYZ'):
    if count > 7:
        break
    print(item)
    count += 1

#X
#Y
#Z
#X
#Y
#Z
#X
#Y
```

Here we create a **for** loop to loop over the infinite cycle of the three letter: XYZ. Of course, we don't want to actually cycle forever, so we add a simple counter to break out of the loop with.

You can also use Python's **next** built-in to iterate over the iterators you create with itertools:

```
from itertools import cycle
polys = ['triangle', 'square', 'pentagon', 'rectangle']
iterator = cycle(polys)
print (next(iterator))
#'triangle'

print (next(iterator))
#'square'

print (next(iterator))
#'pentagon'

print (next(iterator))
#'rectangle'

print (next(iterator))
#'triangle'

print (next(iterator))
#'square'
```

In the code above, we create a simple list of polygons and pass them to **cycle**. We save our new iterator to a variable and then we pass that variable to the

next function. Every time we call next, it returns the next value in the

iterator. Since this iterator is infinite, we can call next all day long and never run out of items.

repeat(object)

The **repeat** iterators will return an object an object over and over again forever unless you set its **times** argument. It is quite similar to **cycle** except that it doesn't cycle over a set of values repeatedly. Let's take a look at a simple example:

```
from itertools import repeat
repeat(5, 5)
repeat(5, 5)

iterator = repeat(5, 5)
print (next(iterator))
#5

print (next(iterator))
#5

print (next(iterator))
#5

print (next(iterator))
#5

print (next(iterator))
#5

print (next(iterator))
#Traceback (most recent call last):
#  File "/usercode/__ed_file.py", line 21, in <module>
# print (next(iterator))
#StopIteration:
```

Here we import **repeat** and tell it to repeat the number 5 five times. Then we call **next** on our new iterator six times to see if it works correctly. When you run this code, you will see that **StopIteration** gets raised because we have run out of values in our iterator.