

# Pipeline Tasks: Takeaways

by Dataquest Labs, Inc. - All rights reserved © 2021

## Syntax

- Calculating the squares of each number using a generator:

```
def squares(N):  
    for i in range(N):  
        yield i * i
```

- Fetching the next element in an iterable:

```
next(iterable)
```

- Turning a list comprehension into a generator comprehension:

```
# list comprehension  
squared_list = [i * i for i in range(20)]  
  
# generator comprehension  
squared_gen = (i * i for i in range(20))
```

- Writing to a file using the csv module:

```
import csv  
  
rows = [('a', 'b', 'c'), ('a1', 'b1', 'c1')]  
  
# Open file with read and write permissions.  
file = open('example_file.csv', 'r+')  
writer = csv.writer(file, delimiter=',')  
writer.writerows(rows)
```

- Combining iterables:

```
import itertools  
import random  
  
nums = [1, 2]  
letters = ('a', 'b')  
  
# Random number generator.  
randoms = (random.random() for _ in range(2))  
  
for ele in itertools.chain(nums, letters, randoms):  
    print(ele)
```

## Concepts

- File streaming works by breaking a file into small sections, and then loaded one at a time into memory.
- A generator is an iterable object that is created from a generator function.
- A generator differs from a regular function in two important ways:
  - A generator uses `yield` instead of `return`.
  - Local variables are kept in memory until the generator completes.

- The yield expression:
  - Lets the Python interpreter know that the function is a generator.
  - Suspends the function execution, keeping the local variables in memory until the next call.
- Once the final yield in the generator is executed, the generator will have exhausted all of its elements.

## Resources

- [Python Generators Tutorial](#)
- [itertools module](#)

Takeaways by Dataquest Labs, Inc. - All rights reserved © 2021