Quickly Analyzing Data With Parallel Processing: Takeaways

Syntax

- Creating a pool of threads:

```
import concurrent.futures
def word_length(word):
    return len(word)
pool = concurrent.futures.ThreadPoolExecutor(max_workers=10)
lengths = pool.map(word_length, ["Hello", "are", "you", "thinking", "of", "becoming", "a",
"polar", "bear", "?"])
```

- Creating a pool of processes:

```
import concurrent.futures
def word_length(word):
    return len(word)
pool = concurrent.futures.ProcessPoolExecutor(max_workers=10)
lengths = pool.map(word_length, ["Hello", "are", "you", "thinking", "of", "becoming", "a",
"polar", "bear", "?"])
```

Concepts

- The `threading` and `multiprocessing` packages are widely used and give you more low-level control.
- The `concurrent.futures` package allows for a simple and consistent interface for both threads and processes.
- `concurrent.features.ThreadPoolExecutor.map()` method returns a generator, but you can just call it using `list` to force it to evaluate.
- Threads and processes are using a paradigm called MapReduce, which is utilized in data processing tools like Apache Hadoop and Apache Spark.

Resources

- Debugging using the multiprocessing module
- Documentation for concurrent.features module