

# Functional Programming

Functional Programming is a different paradigm.

The idea is to treat all computations as mathematical function -- no side effects.

- Easier to predict outputs from inputs
- Easier to debug
- Easier to parallelize

Great intro here: <https://www.codenewbie.org/blogs/object-oriented-programming-vs-functional-programming> (<https://www.codenewbie.org/blogs/object-oriented-programming-vs-functional-programming>)

## Map

```
In [1]: from math import sqrt  
[ sqrt(i) for i in [1, 4, 9, 16] ]
```

```
Out[1]: [1.0, 2.0, 3.0, 4.0]
```

Map applies a **unary** function to each element in the sequence and returns a new sequence containing the results, in the same order.

```
In [2]: from math import sqrt  
map(sqrt, [1, 4, 9, 16])
```

```
Out[2]: <map at 0x10f2ea4a8>
```

So, this is like a generator.

```
In [3]: x = map(sqrt, [1, 4, 9, 16])  
list(x)
```

```
Out[3]: [1.0, 2.0, 3.0, 4.0]
```

```
In [4]: def mymap(f, seq):  
        result = []  
        for elt in seq:  
            result.append( f(elt) )  
        return result
```

```
In [5]: mymap(sqrt, [1, 4, 9, 16])
```

```
Out[5]: [1.0, 2.0, 3.0, 4.0]
```

```
In [6]: def powerOfTwo(k):  
        return 2**k  
  
powerOfTwo(3)
```

```
Out[6]: 8
```

```
In [7]: list(map(powerOfTwo, [1, 2, 3, 4]))
```

```
Out[7]: [2, 4, 8, 16]
```

Beyond Basic Programming - Intermediate Python

recluze.net/learn

```
In [8]: # Short
list( map(lambda k: 2**k, [1, 2, 3, 4]) )

Out[8]: [2, 4, 8, 16]
```

## Filter

```
In [9]: x = filter(str.isalpha, ['x', 'y', '2', '3', 'a'])
print(x)

<filter object at 0x10f3be470>
```

```
In [10]: list(x)
```

```
Out[10]: ['x', 'y', 'a']
```

## Reduce

```
In [11]: from functools import reduce # Need to import reduce
```

```
In [12]: def add(x, y):
          return x + y
```

```
In [14]: reduce(add, [1, 10, 4], 0) # Should give start value
```

```
Out[14]: 15
```

## But Why?

```
In [24]: lines = [
          "A cow is a domestic animal. A cow is a very useful animal.",
          "A cow is kept in barns. Cow milk is very healthy.",
          "Another cow."
          ]
```

Let's count words in all these lines.

```
In [16]: from collections import defaultdict

def count_words(s): # Takes in a single string
    counts = defaultdict(int) # Initializes keys not already present

    for word in s.split():
        counts[word] += 1

    return dict(counts) # don't want to send back the defaultdict

# See more about collections here:
# https://docs.python.org/dev/library/collections.html
```

```
In [17]: dict(count_words(lines[0]))
```

```
Out[17]: {'A': 2,
          'a': 2,
          'animal.': 2,
          'cow': 2,
          'domestic': 1,
          'is': 2,
          'useful': 1,
          'very': 1}
```

## Beyond Basic Programming - Intermediate Python

recluze.net/learn

```
In [27]: list(map(count_words, lines))
```

```
Out[27]: [{'A': 2,
          'a': 2,
          'animal.': 2,
          'cow': 2,
          'domestic': 1,
          'is': 2,
          'useful': 1,
          'very': 1},
          {'A': 1,
          'Cow': 1,
          'barns.': 1,
          'cow': 1,
          'healty.': 1,
          'in': 1,
          'is': 2,
          'kept': 1,
          'milk': 1,
          'very': 1},
          {'Another': 1, 'cow.': 1}]
```

```
In [28]: counts_map = list(map(count_words, lines))
```

```
In [20]: def reduce_counts(x, y):
          print("x:", x)
          print("y:", y)
          print("---")
          return {'word': 0}
```

```
In [29]: reduce(reduce_counts, counts_map, {})
```

```
Out[29]: {'A': 3,
          'Another': 1,
          'Cow': 1,
          'a': 2,
          'animal.': 2,
          'barns.': 1,
          'cow': 3,
          'cow.': 1,
          'domestic': 1,
          'healty.': 1,
          'in': 1,
          'is': 4,
          'kept': 1,
          'milk': 1,
          'useful': 1,
          'very': 2}
```

```
In [22]: from collections import Counter
```

```
def reduce_counts(x, y):
    counter = Counter()      # {Key: Value} where Value is the count

    counter.update(x)        # Get numbers from x
    counter.update(y)        # Add counts from y

    return dict(counter)
```

Beyond Basic Programming - Intermediate Python

recluze.net/learn

In [23]: `reduce(reduce_counts, counts_map, {})`

```
Out[23]: {'A': 3,
          'Cow': 1,
          'a': 2,
          'animal.': 2,
          'barns.': 1,
          'cow': 3,
          'domestic': 1,
          'healthy.': 1,
          'in': 1,
          'is': 4,
          'kept': 1,
          'milk': 1,
          'useful': 1,
          'very': 2}
```

This makes parallelization very easy! That's what MapReduce (and Hadoop/Spark) is built on top of!

Imagine a scenario where you have 1 billion files and a Hadoop cluster of 5,000 machines.

- Take a million files and pass to one machine (Since they are independent, no network overhead)
- Each machine computes their own sum
- Add them all together once!
- Almost 5000x speedup (more if you use threads on one machine)

## Hadoop and Spark

If you're interested in MapReduce for [big data processing](#):

See here: <https://www.cloudera.com/developers/get-started-with-hadoop-tutorial.html>

(<https://www.cloudera.com/developers/get-started-with-hadoop-tutorial.html>)

And here: <https://spark.apache.org/docs/latest/quick-start.html> (<https://spark.apache.org/docs/latest/quick-start.html>)

In [ ]: