# Introduction

This lesson gives a brief introduction to code vectorization and explains an example using pure Python and NumPy.

WE'LL COVER THE FOLLOWING



## What is Code vectorization?

Code vectorization means that the problem you're trying to solve is inherently vectorizable and only requires a few NumPy tricks to make it faster. Of course, it does not mean it is easy or straightforward, but at least it does not necessitate totally rethinking your problem (as it will be the case in the **Problem vectorization** chapter). Still, it may require some experience to see where code can be vectorized.

### Example

Let's illustrate this through a simple example where we want to **sum up two lists of integers**.

### Solution 1: Pure Python

One simple way using pure Python is:

```
def add_python(Z1,Z2):
  return [z1+z2 for (z1,z2) in zip(Z1,Z2)]
```

### Solution 2: Use np.add(list1,list2)

This first naive solution can be vectorized very easily using NumPy:

```
def add_numpy(Z1,Z2):
        return np.add(Z1,Z2)
```

## Compare time of two approaches

Without any surprise, benchmarking the two approaches shows that the second method is the fastest with one order of magnitude.

main.py

tools.py

```
import random
import numpy as np
from tools import timeit

def add_python(Z1,Z2):
  return [z1+z2 for (z1,z2) in zip(Z1,Z2)]

def add_numpy(Z1,Z2):
        return np.add(Z1,Z2)

Z1 = random.sample(range(1000), 100)
Z2 = random.sample(range(1000), 100)

timeit("add_python(Z1, Z2)", globals())
timeit("add_numpy(Z1, Z2)", globals())
```

Not only is the second approach faster, but it also naturally adapts to the shape of `Z1` and `Z2`. This is the reason why we did not write `Z1 + Z2` because it would not work if `Z1` and `Z2` were both lists. In the first Python method, the inner `+` is interpreted differently depending on the nature of the two objects such that if we consider two nested lists, we get the following outputs:

```
import numpy as np

def add_python(Z1,Z2):
    return [z1+z2 for (z1,z2) in zip(Z1,Z2)]
def add_numpy(Z1,Z2):
    return np.add(Z1,Z2)

Z1 = [[1, 2], [3, 4]]
Z2 = [[5, 6], [7, 8]]
print("Using concatenation:",Z1 + Z2)
   #[[1, 2], [3, 4], [5, 6], [7, 8]]
print("Using pure python:",add_python(Z1, Z2))
   #[[1, 2, 5, 6], [3, 4, 7, 8]]
```

```
print("Using numpy.add:",add_numpy(Z1, Z2))
   #[[ 6  8][10 12]]
```

Summing up, the first method concatenates the two lists together, the second method concatenates the internal lists together and the last one computes what is (numerically) expected.

Solve this Quiz!

Q

Which of the following is a good approach when designing a solution to a problem?

COMPLETED 0%

1 of 1

Now that you have learned code vectorization, let's move on to an exercise in the next lesson.