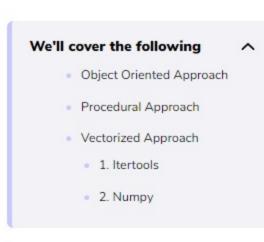


Views and Copies

Coding Example: How to find if one vector is view of the other?

NumPy Vectorization

This lesson teaches Numpy vectorization and explains it with a simple example using object-oriented, procedural and vectorized approach.



Vectorization, in simple words, means optimizing the algorithm so that it can run multiple operations from a single instruction. NumPy is all about vectorization. If you are familiar with Python, this is the main difficulty you'll face because you'll need to change your way of thinking and your new friends (among others) are named "vectors", "arrays", "views" or "ufuncs".

Note: A custom magic command timeit is used in all codes. It's a tool for measuring the execution time of small code snippets.

Object Oriented Approach

Let's take a very simple example, a random walk. One possible object-oriented approach would be to define a RandomWalker class and write a walk method that would return the current position after each (random) step. It's nice, it's readable, but it is slow:

```
main.py
tools.py
                                  def sysinfo():
                                      import sys
                                      import time
                                      import numpy as np
                                      import scipy as sp
                                      import matplotlib
                                                         %s" % (time.strftime("%D")))
                                      print("Date:
                                      version = sys.version_info
                                      major, minor, micro = version.major, version.minor, version.micro
                                      print("Python:
                                                       %d.%d.%d" % (major, minor, micro))
                                                        ", np.__version__)
                                      print("Numpy:
                                                        ", sp.__version__)
                                      print("Scipy:
                                      print("Matplotlib:", matplotlib.__version__)
                              23 def timeit(stmt, globals):
                                      import timeit as _timeit
                                      import numpy as np
                                      trial = _timeit.timeit(stmt, globals=globals, number=1)
    RUN
                                                                                       SAVE
                                                                                                   RESET
                                                                                                             ×
Output
                                                                                                        2.2515
 10 loops, best of 3: 31.6 msec per loop
```

Here loops are the total number of CPU cycles required during a random walk and the time in msec indicates time per cycle.

Procedural Approach

For such a simple problem, we can probably save the class definition and concentrate only on the walk method that computes successive positions after each random step. This new method saves some CPU cycles but not that much because of this function is pretty much the same as in the object-oriented approach and the few cycles we saved probably come from the inner Python object-oriented machinery.

```
main.py
tools.py
                                  def sysinfo():
                                      import sys
                                     import time
                                     import numpy as np
                                     import scipy as sp
                                     import matplotlib
                                     print("Date:
                                                        %s" % (time.strftime("%D")))
                                     version = sys.version_info
                                     major, minor, micro = version.major, version.minor, version.micro
                                     print("Python:
                                                      %d.%d.%d" % (major, minor, micro))
                                                       ", np.__version__)
                                     print("Numpy:
                                      print("Scipy: ", sp.__version__)
                                      print("Matplotlib:", matplotlib.__version__)
                             23 def timeit(stmt, globals):
                                     import timeit as timeit
                                      import numpy as np
                                      # Rough approximation of a single run
                                      trial = _timeit.timeit(stmt, globals=globals, number=1)
    RUN
                                                                                      SAVE
                                                                                                  RESET
                                                                                                           ×
Output
 10 loops, best of 3: 29.6 msec per loop
```

approach. Vectorized Approach

observe that a random walk is an accumulation of steps, we can rewrite the function by first generating

Here we can see that the time taken by the procedural approach is less than that of the object-oriented

For the vectorized approach, we can use Itertools or NumPy.

main.py

1. Itertools Itertools is a python module that offers a set of functions creating iterators for efficient looping. If we

all the steps and accumulate them without any loop:

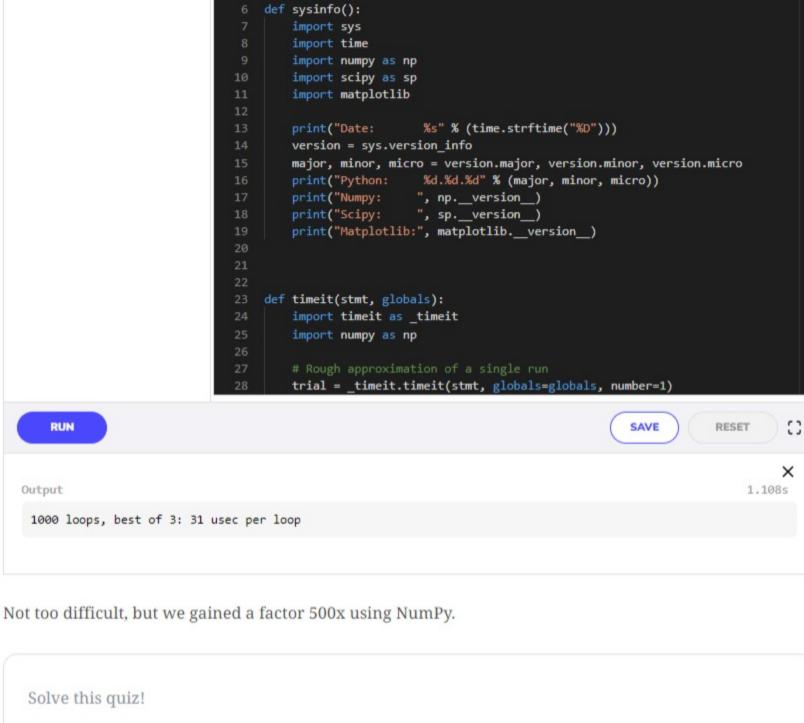
```
tools.py
                                     def sysinfo():
                                         import sys
                                         import time
                                         import numpy as np
                                       import scipy as sp
                                      import matplotlib
                                         print("Date:
                                                           %s" % (time.strftime("%D")))
                                         version = sys.version_info
                                          major, minor, micro = version.major, version.minor, version.micro
                                         print("Python: %d.%d.%d" % (major, minor, micro))
print("Numpy: ", np.__version__)
print("Scipy: ", sp.__version__)
                                          print("Matplotlib:", matplotlib.__version__)
                                 23 def timeit(stmt, globals):
                                       import timeit as _timeit
                                        import numpy as np
                                          trial = _timeit.timeit(stmt, globals=globals, number=1)
       RUN
                                                                                            SAVE
                                                                                                        RESET
                                                                                                                  ×
  Output
                                                                                                             1.180s
   10 loops, best of 3: 4.39 msec per loop
In fact, we've just vectorized our function. Instead of looping for picking sequential steps and add them to
the current position, we first generated all the steps at once and used the accumulate function to
compute all the positions. We got rid of the loop and this makes things faster.
```

2. Numpy We gained 85% of computation-time compared to the previous version, not so bad. But the advantage of

this new version is that it makes NumPy vectorization super simple. We just have to translate itertools

call into NumPy ones:

main.py tools.py



```
What's a good alternative in Numpy for the "accumulate" method from Itertools?
       A) Numpy.sum()
        B) Numpy.cumsum()
       C) Numpy.add()
        D) None of the above
 COMPLETED 0%
                                                                                1 of 2
This course is about vectorization, be it at the code or problem level. We'll see this difference is important
```

before looking at custom vectorization. In the next lesson, we'll learn about "Readability vs. Speed".

