List Comprehensions versus For Loops

Usual articles will perform the following case: create a list using a for loop versus a list comprehension. So let's do it and time it.

```
import time
iterations = 100000000


start = time.time()
mylist = []
for i in range(iterations):
    mylist.append(i+1)
end = time.time()
print(end - start)
>> 9.90 seconds


start = time.time()
mylist = [i+1 for i in range(iterations)]
end = time.time()
print(end - start)
>> 8.20 seconds
```

As we can see, the for loop is slower than the list comprehension (9.9 seconds vs. 8.2 seconds).

> List comprehensions are faster than for loops to create lists.

But, this is because we are creating a list by **appending** new elements to it at each iteration. This is slow.

Side note: It would even be worse if it was a Numpy Array and not a list. The for loop would take minutes to run. But you shouldn't blame the for loop for this: we are simply not using the right tool.

```
myarray = np.array([])
for i in range(iterations):
    myarray = np.append(myarray, i+1)
```

Do not create numpy arrays by appending values in a loop.

For Loops are Faster than List Comprehensions

② Suppose we only want to perform some computations (or call an independent function multiple times) and do not want to create a list.

```
start = time.time()
for i in range(iterations):
    i+1
end = time.time()
print(end - start)
>> 6.16 seconds


start = time.time()
[i+1 for i in range(iterations)]
end = time.time()
print(end - start)
>> 7.80 seconds
```

In that case, we see that the list comprehension is 25% slower than the for loop.

For loops are faster than list comprehensions to run functions.

Array Computations are Faster than For Loops

One element is missing here: what's faster than a for loop or a list comprehension? Array computations! Actually, it is a bad practice in Python to use for loops, list comprehensions, or .apply() in pandas. Instead, you should always prefer array computations.

As you can see, we can create our list even faster using **list(range())**

```
start = time.time()
mylist = list(range(iterations))
end = time.time()
print(end - start)
>> 4.84 seconds
```

It only took 4.84 seconds! This is 40% less time-intensive than our previous list comprehension (8.2 seconds). Moreover, creating a list using list(range(iterations)) is **even faster** than performing simple computations (6.16 seconds with for loops).

If you want to perform some computation on a list of numbers, the best practice is **not** to use a list comprehension or a for loop **but to perform array computations.**

Array computations are faster than loops.

Conclusion

Are list comprehensions faster than for loops?

List comprehensions are the right tool to create lists — it is nevertheless better to use *list(range())*. For loops are the right tool to perform computations or run functions. In any case, avoid using for loops and list comprehensions altogether: use array computations instead.

Once you have your numpy array you'll be able to perform lightning-fast array computations.