Looping in Python

TLDR: I will take points off if I find any looping on `range(len(iterable))`!

If you've come to Python from another programming language, you will be tempted to loop through items in an iterable like this:

```
for i in range(len(some_iterable)):
    # do something with some_iterable[i]
```

Noooooo! We're using Python! Change this to:

```
for item in some_iterable:
    # do something with item
```

This will loop over the elements of `some_iterable`, which is the equivalent of `some_iterable[i]` from above.

Why do it this way, you ask? Here's why:

1. Not all iterables can be indexed.
2. Looks bad. It adds clutter and isn't clean code for Python.
3. It's inefficient.

Python's "for loops" are really "for each" loops. They are **not** like the "for loops" in C/C++ or Java. You don't need an index to loop through an iterable. When you have the urge to use an index, *think seriously about what you are trying to accomplish.* You will probably find you don't need the index after all. In addition, Python code is optimized to be faster with "for each" loops.

If you need the index so you can index another iterable at the same time - ie, you need to access `some_iterable[i]` and also `other_iterable[i]` together, use the built-in `zip()` function:

```
for some_item, other_item in zip(some_iterable, other_iterable):
    # now you can do something with some_item and other_item together
```

The `zip` function takes one element from each iterable and makes it into a tuple, for each element of the iterables. When we loop over the `zip` object, we get tuples, which can be unpacked into the individual items.

```
>>> names = ['Diane', 'Maria', 'Rani']
>>> animals = ['Pugs', 'Cats', 'Pandas']
>>> for name, animal in zip(names, animals):
...     print(f"{name} likes {animal}")
...
Diane likes Pugs
Maria likes Cats
Rani likes Pandas
```

If you are *absolutely 100% sure* you **must** have an index, do it like this:

```
for i, item in enumerate(some_iterable):
    # do something with i and item
```

The `enumerate` function returns an index and the corresponding item. So we can use it even for iterables that are not

sequences and therefore cannot be indexed.

```
>>> names = {'Diane', 'Rani', 'Maria'}
>>> names[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object does not support indexing
>>> for i, item in enumerate(names):
...     print(f'number: {i} is {item}')
...
number: 0 is Diane
number: 1 is Maria
number: 2 is Rani
>>>
```

Indexing returned from `enumerate` begins with `0`. If you want to start the enumerations with a different index, add the `start` keyword parameter to the `enumerate` call:

```
>>> names = {'Diane', 'Rani', 'Maria'}
>>> for i, item in enumerate(names, start=1):
...     print(f'number: {i} is {item}')
...
number: 1 is Diane
number: 2 is Maria
number: 3 is Rani
>>>
```

If any of this is unclear, please ask.

