for...in Loops!

Iteration: Looping Toward a Goal

- Iteration is the superpower of many algorithmic problems:
  - Given a sequence you want to process, you work through them item-by-item

- Common pattern when iterating on a Sequence:

```
seq: range = range(1, 100, 2)
i: int = 0
while i < len(seq):
    item: int = seq[i]
    # process item
    i += 1
```

`for...in` - Iterate through a collection of items.

```
seq: range = range(1, 100, 2)
i: int = 0
while i < len(seq):
    item: int = seq[i]
    # process item
    i += 1
```

```
seq: range = range(1, 100, 2)
for item in odds:
    # process item
```

- The left pattern is so common when iterating through *each item* in a sequence, the *for...in* statement is a simpler abstraction for doing same.
- The *for..in* statement is often broadly called a "for each" construct and has variations in most languages.
- This pattern works generally on any kind of sequence.
  - Also works for other *iterable* collection types we'll explore in the days ahead

`for...in` - Semantics

```
for [identifier] in [sequence]:
    # In the repeat block:
    # identifier is now a variable
    # assigned to the next item
    # in the iteration.
```

- identifier is a variable name you choose for *each item* in repeat block
- The assignment of each item and progression through each item in the sequence is handled for you as part of the construct.
- The loop completes after the repeat block evaluates for the last item.
- This works for algorithms where you want to process *every item.*
  - This gives you *less control* than with a while loop.
  - Many algorithms involve more clever use of indexing.