LESSON 11 : Iterators, generators

Start by going over / using the TrackingDiceSet subclass of DiceSet:

```
class TrackingDiceSet(DiceSet):

    def __init__(self, number, sides, base=1):
        self.history = []
        super().__init__(number, sides, base=base)

    def roll(self):
        val = super().roll()
        self.history.append(val)
        return val
```

Now we want to add functionality so you can iterate over the history, as in a for loop.

- Enter iterator protocol → show page for iterators in python library reference

Make an iterator class:

```
class _~~TDS_Iter~~_TDSIterator:

    def __init__(self, tds)
        self.tds = tds
        self.idx = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.idx < len(self.tds.history):
            self.idx += 1
            return self.tds.history[self.idx - 1]
        raise StopIteration()
```

Now make an __iter__ in TrackingDiceSet!

```
def __iter__(self):
    return _TDSIterator(self)
```

And ~~use~~ iterate over the TDS object in a for loop!

Making a class designed just as an iterator!

Fibonacci Example:

```
class Fib:
    def __init__(self, n):
        self.n = n
        self.a, self.b = 0, 1
        self.idx = 0
    def __iter__(self):
        return self

    def __next__(self):
        self.idx += 1
        if self.idx > self.n:
            raise StopIteration

        self.a, self.b = self.b, self.a + self.b
        return self.a
```

Now introduce a <u>generator function</u> by example:

run this function "by hand" by calling, assigning object, calling iter and next on it:

```
def gfib(n):
    a, b = 0, 1
    for _ in range(n):
        a, b = b, a+b
        yield a
```

← Explain `yield` keyword

Now use gfib in a for loop.

Next revise TrackingDiceSet `__iter__` method as a generator function, remove need for _TDSIterator class.

Now discuss generator expressions, difference in syntax from list comprehensions, ~~lazy versus~~ inherent lazy execution.

Ex `odd_squares = (n**2 for n in range(1000) if n%2==1)`

play with this object in a for loop and by calling iter/next on it.

Emulating sequence types:

Try to access roll history of TDS by subscript notation

- implement `__len__` and `__getitem__` functions.

- Discuss sequence types in language reference - note there is a lot to support.

↳ Now do an alternate implementation of TDS by inheriting from list, exposing full sequence type protocol "for free".