Generators in Python

Or: next, please

Hack Reactor

Iterables, Iterators, and Generators

What's the difference?

We have seen **Iterables**, these are sequence types such as `List`, `Range`, `String`, etc. that can be looped over with `for… in…`

Iterators are a type of object used behind the scenes when traversing the elements of iterables. Explicitly produce **iterators** from iterables with `iter()`, Iterators support the `next()` function.

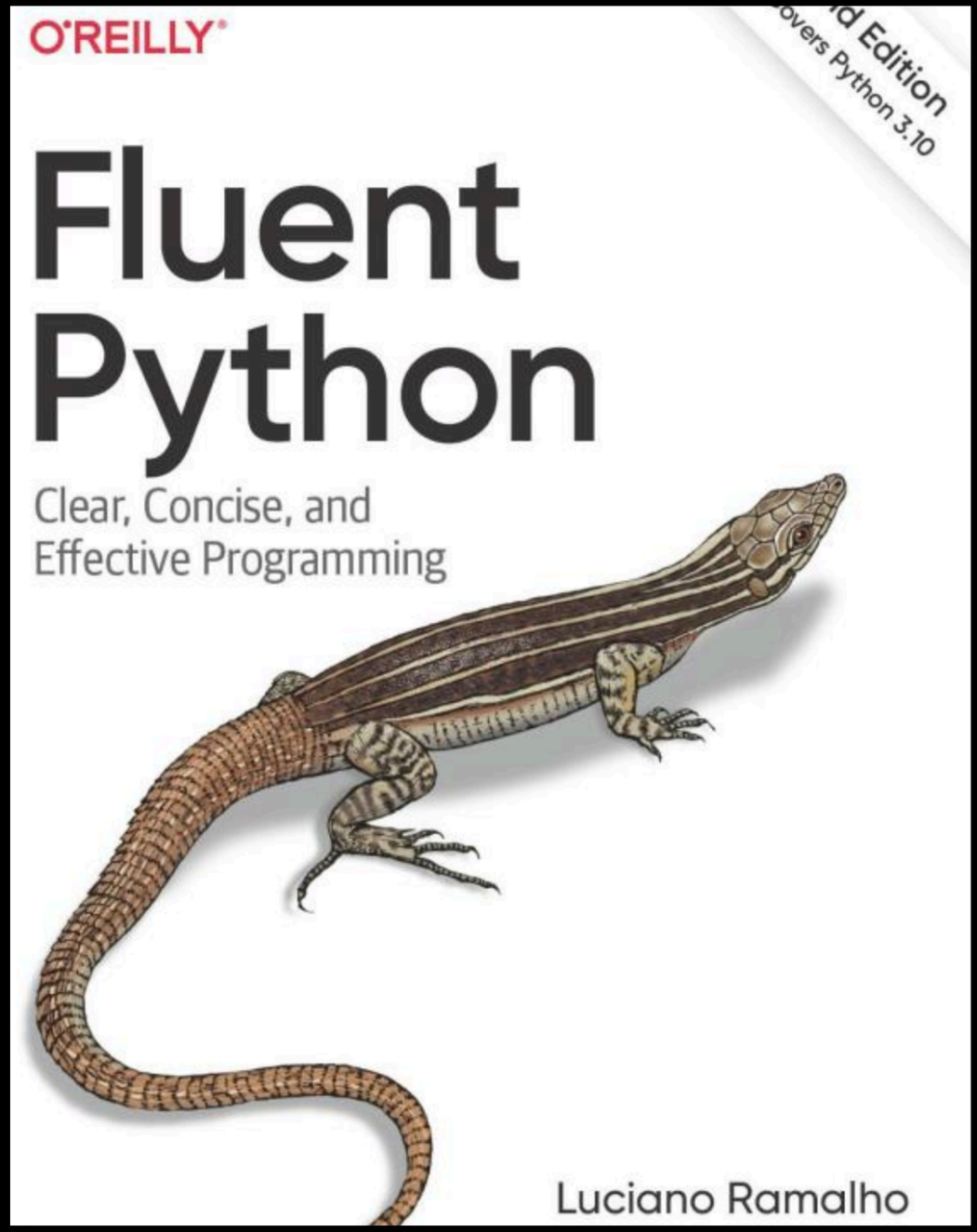
Generators are another type of iterator, produced by a generator expression or generator function.

Iterators and Generators signal termination (if it ever happens) with a **StopIteration** sentinel exception. `for` loops handle this automatically, but it will be raised from a final call to `next()`

MUCH More information

Chapter 14 of Luciano Ramalho's excellent and very thorough *Fluent Python*

Also covers how to design iterables and iterators as classes using __dunder_methods__



Generator Expressions

One-at-a-time “comprehensions”

A generator expression looks a lot like a list comprehension, but is expressed inside of parentheses ( ) instead of brackets [ ]

```
>>> gen = (x*x for x in range(1, 1000001))
>>> next(gen)
1
>>> next(gen)
4
>>> # ...999998 more next calls…
>>> next(gen)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

Unlike a list comprehension this expression does not calculate all 1,000,000 values at once, but delays calculation until the next element is requested

Problem 1

Consecutive values in a list that sum to a target

Write a function that takes as its argument a list of numeric values and a target sum and returns a generator expression that produces the consecutive pairs from the list that sum to the target

```
pairs_that_sum_generator([1,9,2,4,7,4], 11)
```

Would produce a generator that yields the values `(9,2)` `(4,7)` and `(7,4)`

Generator Functions

“Functions” that yield values in sequence

A function that “returns” multiple values in sequence with `yield` instead of `return`
returns a Generator object

```
def simple_generator(x):
  for i in range(x):
    yield i
```

Will produce a generator that produces the same sequence as

```
>>> gen = (y for y in range(x))
```

Problem 2

Pairwise sums from a list

Write a generator function that uses `yield` to produce values. The function takes as its argument a list of numeric values and produces a list of the sums of each consecutive pair of values.

```
pairs_that_sum_generator([1, 9, 2, 4, 1, 4])
```

Would return a generator that yields the values `10, 11, 6, 5, 5`

*The generator will yield one less value than the length of the list argument

Problem 3

Row-wise values of the transpose of a matrix

Homework, see the provided code

Thank You For Your Time

Any questions?