Pydon'ts
Write elegant Python code

by Rodrigo Girão Serrão

PyCon Sri Lanka 2022

About me

Rodrigo Girão Serrão

Formal education: maths

Writing Python for 9 years

Training/teaching:

- APL (Dyalog Ltd.)
- Python, maths, etc (mathspp.com)

@mathsppblog

Pydon'ts
Write elegant Python code

Intro

```
>>> s = "rod"
```

Intro

```
>>> s = "rod"
>>> for idx in range(len(s)):
...     print(s[idx])
...
r
o
d
```

Intro

```
>>> s = "rod"
>>> for char in s:
...     print(char)
...
r
o
d
```

Intro

```
>>> s = "rod"
>>> for idx in range(len(s)):
...     print(f"Letter {idx} is {s[idx]}")
...
Letter 0 is r
Letter 1 is o
Letter 2 is d
```

Intro

```
>>> s = "rod"
>>> for idx in range(len(s)):
...     print(f"Letter {idx} is {s[idx]}")
...
Letter 0 is r
Letter 1 is o
Letter 2 is d
```

Intro

```
>>> s = "rod"
>>> for idx, letter in enumerate(s):
...     print(f"Letter {idx} is {letter}")
...
Letter 0 is r
Letter 1 is o
Letter 2 is d
```

A first stab

Intro

```
>>> s = "rod"
>>> for idx, letter in enumerate(s):
...     print(f"Letter {idx} is {letter}.")
...
Letter 0 is r
Letter 1 is o
Letter 2 is d
```

A first stab

```
>>> s = "rod"
>>> for element in enumerate(s):
...     print(element)
...
(0, 'r')
(1, 'o')
(2, 'd')
```

A first stab

```
>>> s = "rod"
>>> for element in ...:
...     print(element)
...
(0, 'r')
(1, 'o')
(2, 'd')
```

A first stab

```
>>> s = "rod"
>>> for element in [(0, 'r'), (1, 'o'), (2, 'd')]:
...     print(element)
...
(0, 'r')
(1, 'o')
(2, 'd')
```

A first stab

```
>>> enumerate_("rod")
[(0, 'r'), (1, 'o'), (2, 'd')]
```

A first stab

```python
def enumerate_(iterable):
    result = []
    idx = 0
    for elem in iterable:
        result.append((idx, elem))
        idx += 1
    return result
```

A first stab

- Useful model
- (BUT!) Not accurate

…Why?

(Lazy) generators

(Lazy) generators

```
>>> ...
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

(Lazy) generators

```
>>> range(10)  # ..?
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

(Lazy) generators

```
>>> range(10)
range(0, 10)
```

(Lazy) generators

```
>>> range(10)
range(0, 10)

>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

(Lazy) generators

```
>>> enumerate("rod")
<enumerate object at 0x000001F616DE4540>
```

(Lazy) generators

```
>>> enumerate("rod")
<enumerate object at 0x000001F616DE4540>

# vs

>>> list(enumerate("rod"))
[(0, 'r'), (1, 'o'), (2, 'd')]
```

(Lazy) generators

- Generators give items 1 by 1
- Lazy: only work when needed

A first stab

```python
def enumerate_(iterable):
    result = []
    idx = 0
    for elem in iterable:
        result.append((idx, elem))
        idx += 1
    return result
```

(Lazy) generators

```python
def enumerate_(iterable):
    idx = 0
    for elem in iterable:
        return idx, elem
        idx += 1
```

(Lazy) generators

```python
def enumerate_(iterable):
    idx = 0
    for elem in iterable:
        return idx, elem
        idx += 1
```

```
>>> enumerate_("rod")
(0, 'r')
```

(Lazy) generators

```
def enumerate_(iterable):
    idx = 0
    for elem in iterable:
        <lazy-result-kwd> idx, elem
        idx += 1
```

(Lazy) generators

```python
def enumerate_(iterable):
    idx = 0
    for elem in iterable:
        yield idx, elem
        idx += 1
```

(Lazy) generators

```
>>> enumerate_("rod")
<generator object enumerate_ at 0x000...>
```

(Lazy) generators

```
>>> enumerate_("rod")
<generator object enumerate_ at 0x000...>

# vs

>>> list(enumerate_("rod"))
[(0, 'r'), (1, 'o'), (2, 'd')]
```

(Lazy) generators

- Our version is now lazy ☺
- Still inaccurate ☹

Optional parameter

`start`

Optional parameter `start`

```
>>> list(enumerate("rod"))
[(0, 'r'), (1, 'o'), (2, 'd')]
```

Optional parameter `start`

```
>>> list(enumerate("rod"))
[(0, 'r'), (1, 'o'), (2, 'd')]

>>> list(enumerate("rod", 5))
[(5, 'r'), (6, 'o'), (7, 'd')]
```

Optional parameter `start`

```python
def enumerate_(iterable):
    idx = 0
    for elem in iterable:
        yield idx, elem
        idx += 1
```

Optional parameter `start`

```python
def enumerate_(iterable, start=0):
    idx = start
    for elem in iterable:
        yield idx, elem
        idx += 1
```

Optional parameter `start`

- Simple
- Full-featured

Can we improve it..?

Bookkeeping the indices

Optional parameter start

```python
def enumerate_(iterable, start=0):
    idx = start
    for elem in iterable:
        yield idx, elem
        idx += 1
```

Bookkeeping the indices

```
>>> list(enumerate("rod", 5))
[(5, 'r'), (6, 'o'), (7, 'd')]
```

Bookkeeping the indices

```
>>> list(enumerate("rod", 5))
[(5, 'r'), (6, 'o'), (7, 'd')]

>>> ...
[5, 6, 7]
```

Bookkeeping the indices

```
>>> list(enumerate("rod", 5))
[(5, 'r'), (6, 'o'), (7, 'd')]

>>> list(range(5, 5 + 3))
[5, 6, 7]
```

Bookkeeping the indices

```python
def enumerate_(iterable, start=0):
    idxs = range(start, start + len(iterable))
    for i in range(len(iterable)):
        yield idxs[i], iterable[i]
```

Bookkeeping the indices

```python
def enumerate_(iterable, start=0):
    idxs = range(start, start + len(iterable))
    for i in range(len(iterable)):
        yield idxs[i], iterable[i]
```

Bookkeeping the indices

```
>>> list(zip(range(3), "rod"))
[(0, 'r'), (1, 'o'), (2, 'd')]
```

Bookkeeping the indices

```python
def enumerate_(iterable, start=0):
    idxs = range(start, start + len(iterable))
    for i in range(len(iterable)):
        yield idxs[i], iterable[i]
```

Bookkeeping the indices

```python
def enumerate_(iterable, start=0):
    idxs = range(start, start + len(iterable))
    for idx, elem in zip(idxs, iterable):
        yield idx, elem
```

Bookkeeping the indices

- `zip` idiom joins indices & elements

Did we just break something..?

Iterables, not sequences

Iterables, not sequences

```
>>> list(enumerate_("rod"))
[(0, 'r'), (1, 'o'), (2, 'd')]
```

Iterables, not sequences

```
>>> list(enumerate_("rod"))
[(0, 'r'), (1, 'o'), (2, 'd')]

>>> list(enumerate_(["hey", "world", "!"]))
[(0, 'hey'), (1, 'world'), (2, '!')]
```

Iterables, not sequences

```
>>> list(enumerate_("rod"))
[(0, 'r'), (1, 'o'), (2, 'd')]

>>> list(enumerate_(["hey", "world", "!"]))
[(0, 'hey'), (1, 'world'), (2, '!')]

>>> list(enumerate_(range(0, 30, 10)))
[(0, 0), (1, 10), (2, 20)]
```

Iterables, not sequences

```
>>> len("rod")
3

>>> len(["hello", "world", "!"])
3

>>> len(range(0, 30, 10))
3
```

Iterables, not sequences

```
>>> firsts = ["Harry", "Ron", "Hermione"]
>>> lasts = ["Potter", "Weasly", "Granger"]
```

Iterables, not sequences

```
>>> firsts = ["Harry", "Ron", "Hermione"]
>>> lasts = ["Potter", "Weasly", "Granger"]

>>> list(enumerate(zip(firsts, lasts)))
[
    (0, ('Harry', 'Potter')),
    (1, ('Ron', 'Weasly')),
    (2, ('Hermione', 'Granger'))
]
```

Iterables, not sequences

```
>>> firsts = ["Harry", "Ron", "Hermione"]
>>> lasts = ["Potter", "Weasly", "Granger"]

>>> list(enumerate_(zip(firsts, lasts)))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in enumerate_
TypeError: object of type 'zip' has no len()
```

Bookkeeping the indices

```python
def enumerate_(iterable, start=0):
    idxs = range(start, start + len(iterable))
    for idx, elem in zip(idxs, iterable):
        yield idx, elem
```

Not all good things need to end

Not all good things need to end

```python
def gen_indices(start):
    idx = start
    while True:
        yield idx
        idx += 1
```

Not all good things need to end

```
>>> for idx in gen_indices(0):
...     print(idx, end=" ")
...
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 1
21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
38 39 40 41 42 43 44 45 46 47 48 49 50 51 52
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
KeyboardInterrupt
```

Not all good things need to end

```
>>> for idx in gen_indices(0):
...     print(idx, end=" ")
...
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 1
21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
38 39 40 41 42 43 44 45 46 47 48 49 50 51 52
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
KeyboardInterrupt
```

next

Not all good things need to end

```
>>> count_from_42 = gen_indices(42)
```

Not all good things need to end

```
>>> count_from_42 = gen_indices(42)
>>> count_from_42
<generator object gen_indices at 0x00...>
```

Not all good things need to end

```
>>> count_from_42 = gen_indices(42)
```

Not all good things need to end

```
>>> count_from_42 = gen_indices(42)
>>> next(count_from_42)
42
```

Not all good things need to end

```
>>> count_from_42 = gen_indices(42)
>>> next(count_from_42)
42
>>> next(count_from_42)
43
```

Not all good things need to end

```
>>> count_from_42 = gen_indices(42)
>>> next(count_from_42)
42
>>> next(count_from_42)
43
>>> next(count_from_42)
44
```

Not all good things need to end

```python
def enumerate_(iterable, start=0):
    idxs = range(start, start + len(iterable))
    for idx, elem in zip(idxs, iterable):
        yield idx, elem
```

Not all good things need to end

```python
def gen_indices(start):
    ...

def enumerate_(iterable, start=0):
    idxs = gen_indices(start)
    for idx, elem in zip(idxs, iterable):
        yield idx, elem
```

Not all good things need to end

- `range` was too constrained
- `gen_indices` is infinite
    - `zip` stops it

The right tool for the job

The right tool for the job

```python
from itertools import count
```

The right tool for the job

```python
from itertools import count
```

```
>>> help(count)
# ...
 |  Equivalent to:
 |      def count(firstval=0, step=1):
 |          x = firstval
 |          while 1:
 |              yield x
 |              x += step
```

The right tool for the job

```python
def gen_indices(start):
    ...

def enumerate_(iterable, start=0):
    idxs = gen_indices(start)
    for idx, elem in zip(idxs, iterable):
        yield idx, elem
```

The right tool for the job

```python
from itertools import count

def enumerate_(iterable, start=0):
    idxs = count(start)
    for idx, elem in zip(idxs, iterable):
        yield idx, elem
```

The right tool for the job

- Right tool = expressive code
- Rely on the PSL

Are we there yet..?

Yielding from another iterable

Yielding from another iterable

```python
from itertools import count

def enumerate_(iterable, start=0):
    idxs = count(start)
    for idx, elem in zip(idxs, iterable):
        yield idx, elem
```

Yielding from another iterable

```python
from itertools import count

def enumerate_(iterable, start=0):
    idxs = count(start)
    for idx, elem in zip(idxs, iterable):
        yield idx, elem
```

Yielding from another iterable

```python
from itertools import count

def enumerate_(iterable, start=0):
    idxs = count(start)
    for idx, elem in zip(idxs, iterable):
        yield idx, elem
```

Yielding from another iterable

```python
from itertools import count

def enumerate_(iterable, start=0):
    idxs = count(start)
    for t in zip(idxs, iterable):
        yield t
```

Yielding from another iterable

```python
from itertools import count

def enumerate_(iterable, start=0):
    for t in zip(count(start), iterable):
        yield t
```

Yielding from another iterable

```python
from itertools import count

def enumerate_(iterable, start=0):
    yield from zip(count(start), iterable)
```

Recap

Recap

```python
def enumerate_(iterable):
    result = []
    idx = 0
    for elem in iterable:
        result.append((idx, elem))
        idx += 1
    return result
```

Recap

```python
def enumerate_(iterable):
    idx = 0
    for elem in iterable:
        yield idx, elem
        idx += 1
```

Recap

```python
def enumerate_(iterable, start=0):
    idx = start
    for elem in iterable:
        yield idx, elem
        idx += 1
```

Recap

```python
def enumerate_(iterable, start=0):
    idxs = range(start, start + len(iterable))
    for idx, elem in zip(idxs, iterable):
        yield idx, elem
```

Recap

```python
def gen_indices(start):
    ...

def enumerate_(iterable, start=0):
    data = zip(gen_indices(start), iterable)
    for idx, elem in data:
        yield idx, elem
```

Recap

```python
from itertools import count

def enumerate_(iterable, start=0):
    data = zip(count(start), iterable)
    for idx, elem in data:
        yield idx, elem
```

Recap

```python
from itertools import count

def enumerate_(iterable, start=0):
    yield from zip(count(start), iterable)
```

Recap

- Sometimes we make mistakes
- Always experiment
- Aim for ~~perfection~~ sustained improvements

The end..?

References

- Pydon'ts, https://mathspp.com/blog/pydonts
    - Zip-up, https://mathspp.com/blog/pydonts/zip-up
    - Bite-sized refactoring, https://mathspp.com/blog/pydonts/bite-sized-refactoring
    - Why mastering Python is impossible, and why that's ok, https://mathspp.com/blog/pydonts/why-mastering-python-is-impossible
- Enumerate from first principles, https://mathspp.com/blog/enumerate-from-first-principles
- Original twitter thread: https://twitter.com/mathsppblog/status/1455444589603557378

pydonts.com

@mathsppblog

mathspp.com/subscribe

/mathspp/talks

$$\overbrace{\underbrace{\text{rodrigo}}_{\text{name}}\text{@}\underbrace{\text{mathspp.com}}_{\text{site}}}^{\text{email}}$$