Iteration,
iteration,
iteration.

John
Sutherland.


@sneeu


FanDuel

Iteration,
iteration,
iteration.

Factorial.

$$4! = 4 \times 3 \times 2 \times 1$$

while

```
def factorial(n):
    fact = 1

    while n >= 1:
        fact = fact * n
        n = n - 1

    return fact
```

for

```
def factorial(n):
    fact = 1

    for i in range(n, 1, -1):
        fact *= i

    return fact
```

Recursion.

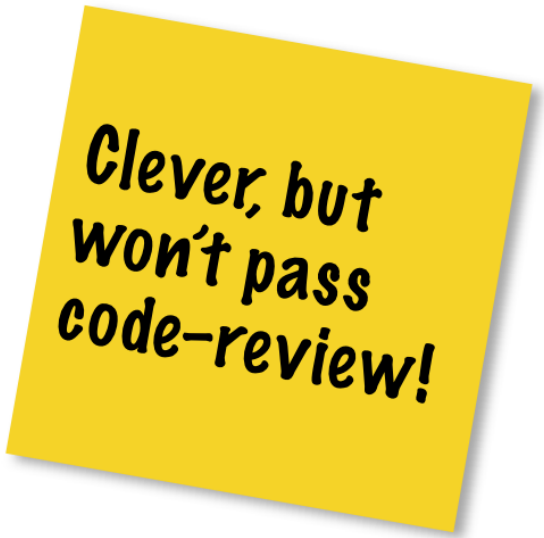
$$4! = 4 \times 3!$$

$$\mathbf{n! = n \times (n-1)!}$$

```
def factorial(n):
    if n <= 1:
        return 1
    return n * factorial(n - 1)
```

reduce

```
import functools
import operator

def factorial(n):
    return functools.reduce(
        operator.mul,
        range(1, n + 1))
```



Iterators.

```
>>> iterator = iter([2, 3, 5])
>>> next(iterator)
2
>>> next(iterator)
3
>>> next(iterator)
5
>>> next(iterator)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

```
>>> for n in iter([2, 3, 5]):
...     print(n)
...
2
3
5
```

```
class Fibonacci:
    def __init__(self):
        self.a, self.b = 1, 1

    def __iter__(self):
        return self

    def __next__(self):
        r = self.a
        self.a = self.b
        self.b = r + self.b
        return r
```

itertools

```
>>> import itertools
>>> forever = itertools.count(0)
>>> next(forever)
0
>>> next(forever)
1
>>> next(forever)
2
>>> next(forever)
3
```

```
>>> list(itertools.islice(zip(
...     itertools.count(0),
...     itertools.cycle("ABC")), 4))
[(0, 'A'), (1, 'B'), (2, 'C'), (3, 'A')]
```

List Compre-hensions.

```
>>> people = Person.objects.all()
>>> [p.name for p in people]
['Ro', 'John', 'Lucy', 'Tom']
```

```
>>> [p.name for p in people
...          if not p.is_boss]
['John', 'Tom']
```

```
>>> [m + n for m in range(1, 7)
...           for n in range(1, 7)]
[2, 3, 4, 5, 6, 7,
 3, 4, 5, 6, 7, 8,
 4, 5, 6, 7, 8, 9,
 5, 6, 7, 8, 9, 10,
 6, 7, 8, 9, 10, 11,
 7, 8, 9, 10, 11, 12]
```

Generator expressions.

```
>>> (p.name for p in people)
<generator object <genexpr> at
0x1032dd9e8>
```

```
>>> list(p.name for p in people)
['Ro', 'John', 'Lucy', 'Tom']
```

Generators.

```
def fibonacci():
    a, b = 1, 1

    while True:
        yield a
        a, b = b, a + b
```

```
class Fibonacci:
    def __init__(self):
        self.a, self.b = 1, 1

    def __iter__(self):
        return self

    def __next__(self):
        r = self.a
        self.a = self.b
        self.b = r + self.b
        return r
```

```
>>> fibs = fibonacci()
>>> next(fibs)
1
>>> next(fibs)
1
>>> next(fibs)
2
>>> next(fibs)
3
>>> next(fibs)
5
>>> next(fibs)
8
```

```
>>> list(fibonacci())
^CTraceback (most recent call last):
  File "<stdin>", line 1, in <module>
```

yield from

```
def bosses():
    people = query(Person.is_boss == True)

    for person in people:
        yield person
```

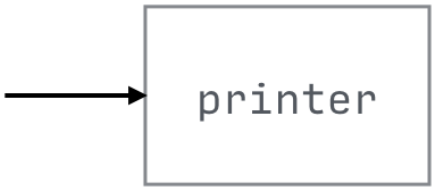
```
def bosses():
    people = query(Person.is_boss == True)

    yield from people
```

Coroutines.


= yield

```
def printer():
    line_number = 0
    while True:
        item = yield
        print(f"{line_number} {item}")
        line_number += 1
```

```
>>> p = printer()
>>> next(p)
>>> p.send("Ro")
0 Ro
>>> p.send("Lucy")
1 Lucy
>>> p.send("Tom")
2 Tom
```


printer

```
import re

def filter_printer(pred):
    while True:
        item = yield
        if pred(item):
            print(item)
```

```
>>> pred = lambda n: n[1] == 'u'
>>>
>>> p = filter_printer(pred)
>>> next(None)
>>> p.send("Ro")
>>> p.send("Lucy")
Lucy
>>> p.send("Tom")
```


filter_printer

```
@coroutine
def filter(pred, sink):
    while True:
        item = yield
        if pred(item):
            sink.send(item)
```

```
import functools

def coroutine(cr):
    @functools.wraps(cr)
    def f(*args, **kwargs):
        primed = cr(*args, **kwargs)
        next(primed)
        return primed
    return f
```

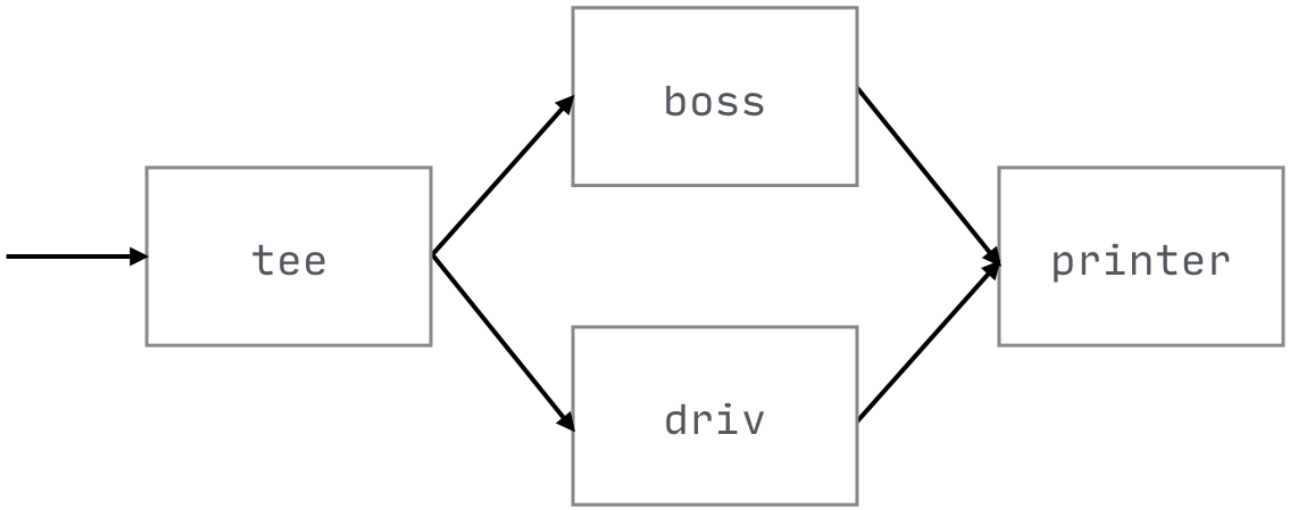

filter

```
>>> pr = printer()
>>>
>>> taxi_drivers = filter(
...     lambda p: p.can_drive, pr)
>>>
>>> for p in people:
...     taxi_drivers.send(p)
...
0 Ro
1 John
```


filter
printer

```
@coroutine
def tee(sinks):
    while True:
        item = yield
        for s in sinks:
            s.send(item)
```

```
>>> log = printer()
>>> boss = filter(lambda p: p.is_boss, log)
>>> driv = filter(lambda p: p.can_drive, log)
>>>
>>> pipeline = tee([boss, driv])
>>> for p in people:
...     pipeline.send(p)
0 Ro
1 Ro
2 John
3 Lucy
```


boss
tee
printer
driv

https://flic.kr/p/aubWis

__pow__

```
>>> pipeline = a(…, b(…, c(…, d(…))))
```

$ a … | b … | c … | d …

```
import functools

class PowerfulCombiner(object):
    def __init__(self, partial_coro):
        self.partial_coro = partial_coro

    def __pow__(self, other):
        return self.partial_coro(other)

def powerful(coro):
    def powerful_inner(*args, **kwargs):
        return PowerfulCombiner(
            functools.partial(coro, *args, **kwargs))

    return powerful_inner
```

```
>>> pipeline = a(…) ** b(…) ** c(…)
```

```
>>> rnge(10, 15) ** filter(even) ** printer()
0 10
1 12
2 14
```

Iteration,
iteration,
iteration.

Thanks!

@sneeu