Generators

A Primes Generator

- Task: produce a stream of all prime numbers, one by one.
- The algorithm that we will use for that is termed sieve (נָפָה), which dates back to the 3rd century BC, and works as follows:
  - 2 is prime
  - Iterate endlessly from N=3 onward
    - For each number N, check if is it not divided by any previously found prime. If so, store it in a list of primes
  - See Wikipedia

A Primes Generator – Sieve

```
def primes1():
    """ a generator for all prime numbers """
    primes_lst = []
    N = 2
    while True:
        isPrime = True
        for p in primes_lst:
            if N % p == 0:
                isPrime = False
        if isPrime:
            primes_lst.append(N)
            yield N
        N += 1
```

A Better Primes Generator

- This algorithms is costly in terms of memory. To extract a prime we need to store all smaller primes
- Alternatives?

```
def primes2():
    """ a generator for all prime numbers
        using good old Fermat's test """
    N = 2
    while True:
        if is_prime(N):
            yield N
        N += 1
```

Extracting the n’th Prime

```
>>> prim = primes()
>>> type(prim)
<class 'generator'>
>>> for i in range(8):
        next(prim)
2
3
5
7
11
13
17
19
>>> for i in range(8):
        next(prim)
23
29
31
37
41
43
47
53
```

Extracting the n’th Prime

```
def get_prime(n):
    g = primes2()
    for i in range(n-1):
        next(g) #no need to store it
    return next(g)
```

Generators: exam question

שאלה 5 (20 נק')

שאלה זאת עוסקת בפונקציות גנרטור.

<u>תזכורת</u>: קריאה לפונקצית גנרטור fgen מחזירה גנרטור (סוג של איטרטור), gen, וכל קריאה ל next(gen) מחזירה ערך כלשהו.

נאמר שפונקצית הגנרטור fgen **מייצרת** ערך x אם x מתקבל (מוחזר) אחרי מס' **סופי** של קריאות ל next(gen).

נאמר ש fgen **מייצרת** קבוצה (אולי אינסופית) של ערכים אם היא מייצרת כל ערך בקבוצה.

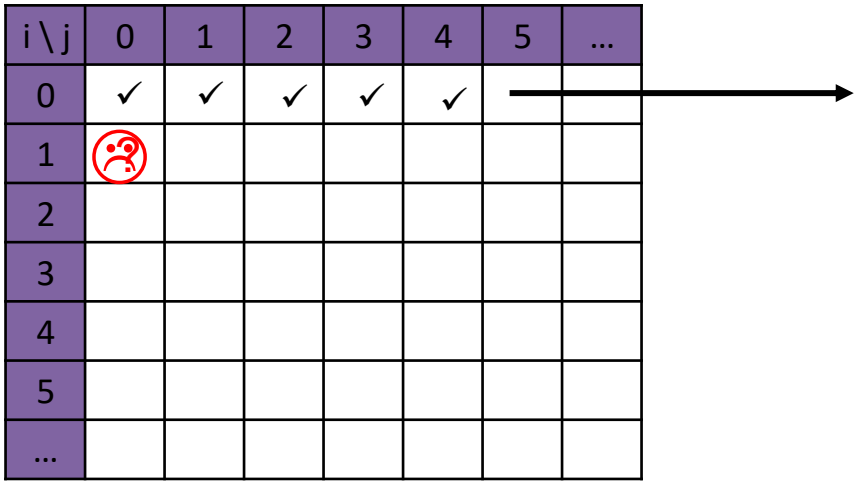
<u>בשאלה זו אין חשיבות לסדר ייצור הערכים ע"י הגנרטור</u>

Generate all pairs of naturals

רוני התבקש לכתוב פונקציית גנרטור שמייצרת את כל (אינסוף) הזוגות הסדורים של מספרים טבעיים (i,j), ללא חשיבות לסדר ייצורם וללא חזרות. שימו לב שהזוגות הם סדורים, כלומר (2,3) שונה מ- (3,2), והזוגות הנדרשים כוללים גם זוגות בהם i=j (למשל הזוג (2,2)).

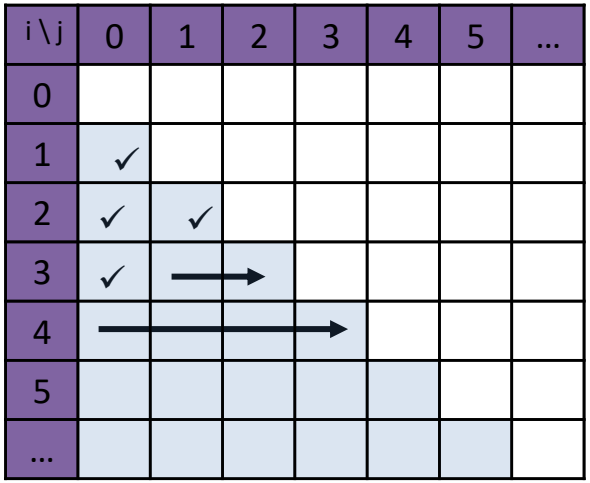
הוא כתב את הקוד הבא:

```
def AllPairs():
 i=0
 while True:
    j=0
    while True:
          yield(i,j)
          j=j+1
    i=i+1
```



Generate all pairs (i,j) where $j < i$

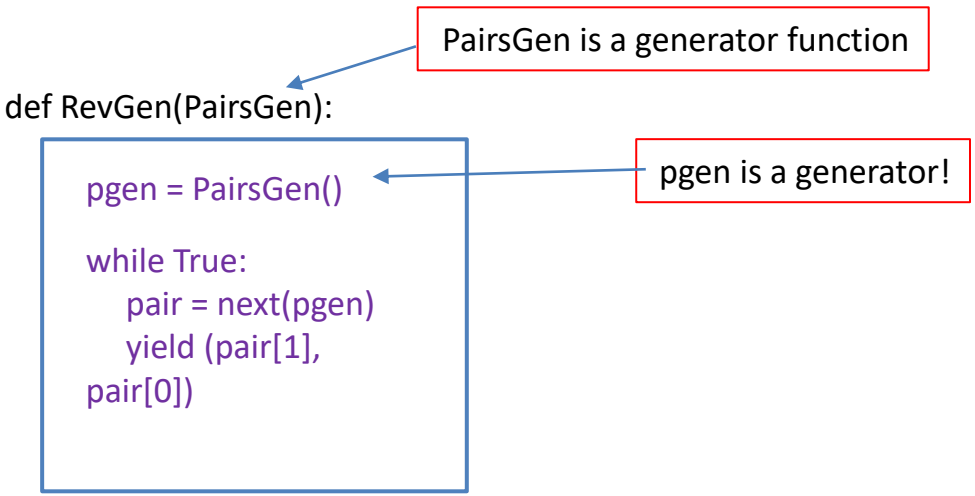
```
def SomePairs():
    i=0
    while True:
        for j in range(i):
            yield (i, j)
        i = i + 1
```




```
>>> gen = SomePairs()
>>> for k in range(5):
print(next(gen))
(1,0)
(2,0)
(2,1)
(3,0)
(3,1)
```

SomePairs is a generator function → when called, it returns a generator but does not start execution immediately.

RevGen




```
>>> gen = RevGen(SomePairs)
>>> for k in range(5):
print(next(gen))
(0,1)
(0,2)
(1,2)
(0,3)
(1,3)
```

RevGen is a generator function, whose input is a generator function generating an infinite number of pairs

UnionGenerators

gen1, gen2 are generators!

```
def UnionGenerators(gen1, gen2):
    while True:
        yield (next(gen1))
        yield (next(gen2))
```

```
>>> g1 = SomePairs()
>>> g2 = RevGen(SomePairs)
>>> gen = UnionGenerators(g1, g2)
>>> for k in range(5):
print(next(gen))
(1,0)
(0,1)
(2,0)
(0,2)
(2,1)
```

UnionGenerators is a generator function, whose input is two generators , each generating an infinite number of values. No value is generated by both.

EqPairs: generates all (i,i) pairs

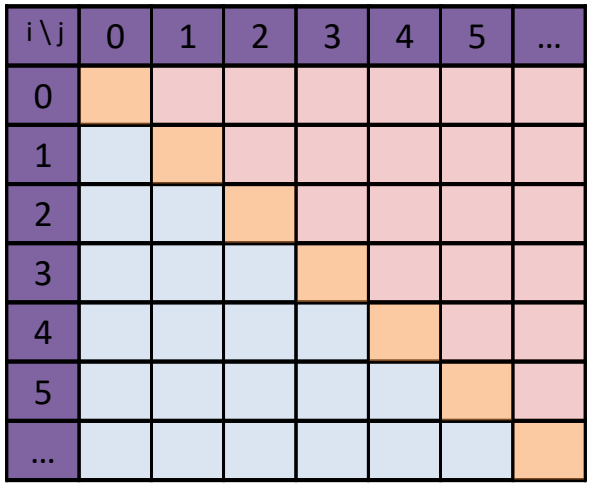
def EqPairs():

```
i=0
while True:
    yield (i, i)
    i = i + 1
```

```
>>> gen = EqPairs()
>>> for k in range(5):
print(next(gen))
(0,0)
(1,1)
(2,2)
(3,3)
(4,4)
```

EqPairs is a generator function

Combining all into: AllPairs



gen = UnionGenerators(g1_2, g3)

g1_2 = UnionGenerators(g1, g2)

g1 = EqPairs()

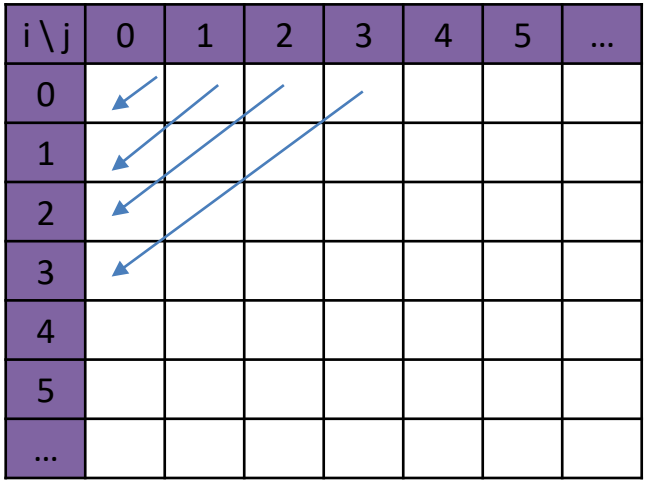
g2 = RevGen(SomePairs)

g3 = SomePairs()

```
>>> gen = AllPairs()
>>> for k in range(7):
print(next(gen))
(0,0)
(1,0)
(0,1)
(2,0)
(1,1)
(2,1)
(0,2)
```

AllPairs is a generator function → when called, it returns a generator but does not start execution immediately.

Bonus: All in one




```
def AllPairs_v2():
    sum = 0
    while True:
        for i in range(sum + 1):
            yield i, sum - i
        sum += 1
```

```
gen = AllPairs_v2()
[next(gen) for x in range(10)]
```

```
[(0, 0),
 (0, 1),
 (1, 0),
 (0, 2),
 (1, 1),
 (2, 0),
 (0, 3),
 (1, 2),
 (2, 1),
 (3, 0)]
```

Lazy Evaluation

- Generators are good not only for infinite sequences, but also for reducing computation.

```
import time
start = time.time()

dim = 10000000
x = [random.random() for i in range(dim)] # O(dim) operation
y = [random.random() for i in range(dim)] # O(dim) operation

for i, (a, b) in enumerate(zip(x, y)):
    if abs(a - b) < 0.05:
        print('Absolute difference is', abs(a - b), '< 0.05
        break

end = time.time()
print('Elapsed seconds', end - start)
```

zip reads a pair of elements from *x* & *y* into a tuple *(a, b)* in each iteration.

enumerate adds the index of the iteration *i* (counter).

```
Absolute difference is 0.00916299450466107 < 0.05 at index 4
Elapsed seconds 2.525766372680664
```

Lazy Evaluation (cont.)

- In Python 3 `zip`, `enumerate`, `range` and many other builtin functions are generators.

```
import time
start = time.time()

dim = 10000000
x = (random.random() for i in range(dim)) # O(1) operations
y = (random.random() for i in range(dim)) # O(1) operations

for i, (a, b) in enumerate(zip(x, y)):
    if abs(a - b) < 0.05:
        print('Absolute difference is', abs(a - b), '< 0.05 at index', i)
        break

end = time.time()
print('Elapsed seconds', end - start)
```

```
Absolute difference is 0.039369254226192 53 < 0.05 at index 9
Elapsed seconds 0.00076413154602050 78
```